

CONSEGNA G4 - SISTEMA INFORMATIVO PER LA GESTIONE DELL'EMISSIONE DI BIGLIETTI AEREI

## Relazione finale del progetto di Basi di Dati

Candidato:

**Giovanni Liboni**

Matricola VR363021

## Contents

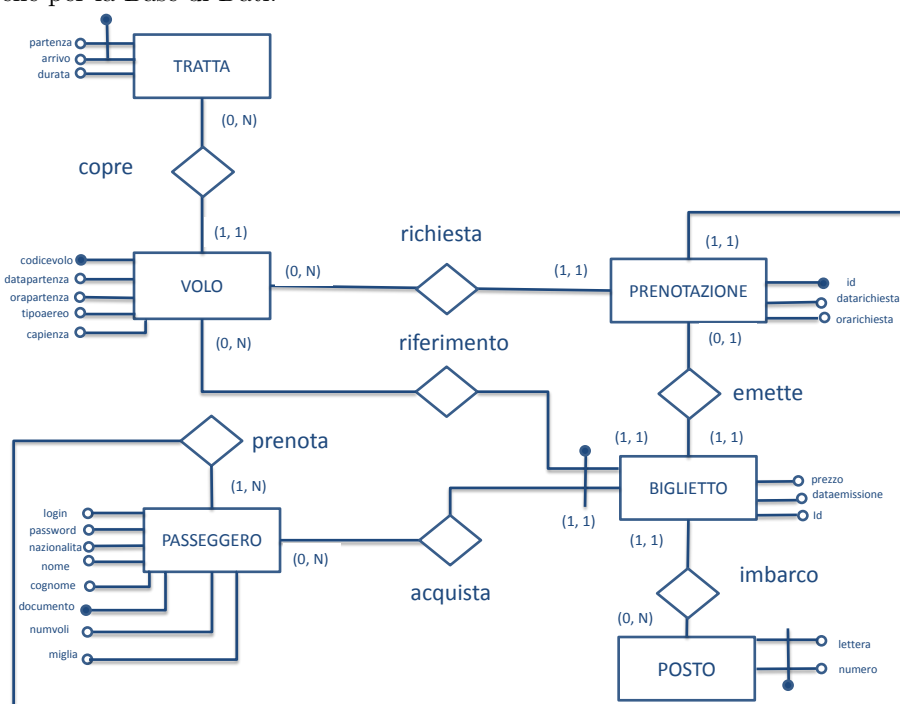
<b>I</b>	<b>Progetto della basi di dati</b>	<b>2</b>
1	Progetto concettuale	2
2	Progetto logico	3
3	Popolamento della base di dati	6
<b>II</b>	<b>Progetto del sito web</b>	<b>6</b>
4	Progettazione logica	6
5	Struttura dell'applicazione web	7
<b>III</b>	<b>Architettura MVC2</b>	<b>7</b>
6	Model	7
7	View	7
8	Controller	7
9	Gestione di errori	10
10	Pagine JSPs e parametri passati	10
11	Path e Context	11
12	Login e sessioni	11
<b>IV</b>	<b>Scelte progettuali</b>	<b>11</b>
13	Hibernate	11
14	AJAX	11

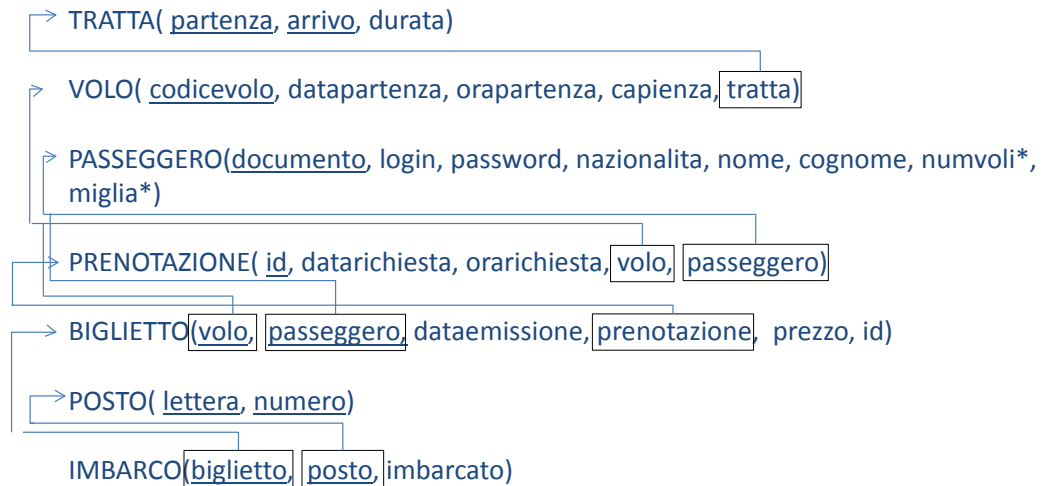
## Part I

# Progetto della basi di dati

## 1 Progetto concettuale

Qui di seguito si riporta il diagramma Entità - Relazione e lo schema concettuale ricavati dalle specifiche del progetto. A partire da questo schema si sono scritte le tabelle per la Base di Dati.





Descrivere il diagramma e lo schema logico.

## 2 Progetto logico

Abbiamo usato un singolo script per la creazione delle tabelle sul database. Il codice viene riportato qui di seguito.

```

1 CREATE TABLE tratta(
2   partenza VARCHAR(100) NOT NULL,
3   arrivo   VARCHAR(100) NOT NULL,
4   durata   INTEGER NOT NULL,
5   distanza NUMERIC(10,1) NOT NULL DEFAULT '0.0',
6   PRIMARY KEY ( partenza, arrivo ),
7   UNIQUE ( partenza, arrivo )
8 );
9 CREATE TABLE volo(
10  codicevolo VARCHAR(10) NOT NULL,
11  partenza   VARCHAR(100) NOT NULL,
12  arrivo     VARCHAR(100) NOT NULL,
13  datapartenza DATE NOT NULL,
14  orapartenza TIME NOT NULL,
15  tipoaereo   VARCHAR(50) NOT NULL,
16  capienza    INTEGER NOT NULL,
17  PRIMARY KEY( codicevolo ),
18  UNIQUE ( codicevolo ),
19  FOREIGN KEY( partenza, arrivo )

```

```

20 REFERENCES tratta( partenza , arrivo )
21 ON UPDATE CASCADE
22 ON DELETE CASCADE
23 );
24 CREATE TABLE passeggero(
25 login VARCHAR(100) NOT NULL,
26 password VARCHAR(100) NOT NULL,
27 nazionalita VARCHAR(100) NOT NULL,
28 nome VARCHAR(100) NOT NULL,
29 cognome VARCHAR(100) NOT NULL,
30 documento VARCHAR(50) NOT NULL,
31 picture BYTEA,
32
33 numvoli INTEGER DEFAULT 0,
34 miglia FLOAT DEFAULT 0,
35 tessera BOOLEAN DEFAULT FALSE,
36 PRIMARY KEY( documento ),
37 UNIQUE ( login )
38 );
39 CREATE TABLE prenotazione(
40 id SERIAL,
41 codicevolo VARCHAR(10) NOT NULL
42 REFERENCES volo( codicevolo )
43 ON UPDATE CASCADE
44 ON DELETE CASCADE,
45 documento VARCHAR(50) NOT NULL
46 REFERENCES passeggero( documento )
47 ON UPDATE CASCADE
48 ON DELETE CASCADE,
49 datarichiesta DATE,
50 orarichiesta TIME,
51 PRIMARY KEY( id ),
52 UNIQUE ( codicevolo , documento )
53 );
54 CREATE TABLE biglietto(
55 id SERIAL,
56 codicevolo VARCHAR(10) NOT NULL
57 REFERENCES volo( codicevolo )
58 ON UPDATE CASCADE
59 ON DELETE CASCADE,
60 documento VARCHAR(50) NOT NULL
61 REFERENCES passeggero( documento )
62 ON UPDATE CASCADE
63 ON DELETE CASCADE,
64 id_prenotazione INTEGER NOT NULL
65 REFERENCES prenotazione( id )
66 ON UPDATE CASCADE
67 ON DELETE CASCADE,
68 dataemissione DATE NOT NULL,
69 prezzo NUMERIC(10,2) NOT NULL,
70
71 PRIMARY KEY( codicevolo , documento ),
72 UNIQUE ( codicevolo , documento , id )
73 );
74
75 CREATE TABLE posto(
76 lettera CHAR NOT NULL,
77 numero INTEGER DEFAULT 0,
78 PRIMARY KEY ( lettera , numero ),
79 UNIQUE ( lettera , numero )
80 );
81 CREATE TABLE imbarco(

```

```

82 lettera CHAR NOT NULL,
83 numero INTEGER DEFAULT 0,
84 imbarcato BOOLEAN DEFAULT FALSE,
85 -- PRIMARY KEY DEL BIGLIETTO
86 codicevolo VARCHAR(10) NOT NULL,
87 documento VARCHAR(50) NOT NULL,
88 -- IDENTIFICATO DAL BIGLIETTO E DAL POSTO
89 PRIMARY KEY ( codicevolo , documento , lettera , numero ),
90
91 FOREIGN KEY( lettera , numero )
92 REFERENCES posto( lettera , numero )
93 ON UPDATE CASCADE
94 ON DELETE CASCADE,
95 FOREIGN KEY( codicevolo , documento )
96 REFERENCES biglietto( codicevolo , documento )
97 ON UPDATE CASCADE
98 ON DELETE CASCADE
99 );

```

Abbiamo aggiunto delle funzioni scritte in *plpgsql* per eseguire azioni di routine sul base di dati e per inserire automaticamente la data e l'ora di inserimento di determinate tuple. Si riporta di seguito il codice delle funzioni create.

```

1 CREATE OR REPLACE FUNCTION update_time_richiesta()
2 RETURNS TRIGGER AS '
3 BEGIN
4     IF NEW.datarichiesta IS NULL THEN
5         NEW.datarichiesta := current_date;
6     END IF;
7     IF NEW.orarichiesta IS NULL THEN
8         NEW.orarichiesta := current_time;
9     END IF;
10    RETURN NEW;
11 END' LANGUAGE 'plpgsql';
12
13
14
15 CREATE OR REPLACE FUNCTION update_time-biglietto()
16 RETURNS TRIGGER AS '
17 BEGIN
18     IF NEW.dataemissione IS NULL THEN
19         NEW.dataemissione := current_date;
20     END IF;
21    RETURN NEW;
22 END' LANGUAGE 'plpgsql';
23
24
25
26 CREATE OR REPLACE FUNCTION update_numvoli()
27 RETURNS TRIGGER AS '
28 BEGIN
29     IF NEW.dataemissione IS NULL THEN
30         NEW.dataemissione := current_date;
31     END IF;
32    RETURN NEW;
33 END' LANGUAGE 'plpgsql';
34
35 CREATE OR REPLACE FUNCTION update_tessera() RETURNS trigger AS $$
36 BEGIN
37     UPDATE passeggero
38     SET numvoli = (SELECT count(*) FROM volo JOIN biglietto ON
                     biglietto.codicevolo = volo.codicevolo

```

```

39      JOIN tratta ON (volo.partenza = tratta.partenza
AND volo.arrivo = tratta.arrivo )
40      WHERE passeggero.tessera=true AND passeggero.documento =
biglietto.documento
41      GROUP BY passeggero.documento ),
42  miglia = (SELECT sum(distanza) FROM volo JOIN biglietto ON
biglietto.codicevolo = volo.codicevolo
43      JOIN tratta ON (volo.partenza = tratta.partenza
AND volo.arrivo = tratta.arrivo )
44      WHERE passeggero.tessera=true AND passeggero.documento =
biglietto.documento
45      GROUP BY passeggero.documento )
46  FROM biglietto
47  WHERE biglietto.dataemissione >= (SELECT date('now') - interval '
3 year') AND NEW.documento = passeggero.documento;
48  RETURN NEW;
49
50 END;
51 $$ LANGUAGE 'plpgsql';

```

Per automatizzare l'esecuzione delle funzioni si sono scritti dei trigger, azioni eseguite al verificarsi di certe condizioni. I trigger creati sono i seguenti:

```

1 CREATE TRIGGER update_time_richiesta
2 BEFORE INSERT ON prenotazione
3   FOR EACH ROW EXECUTE PROCEDURE update_time_richiesta();
4
5 CREATE TRIGGER update_tessera
6 AFTER INSERT OR DELETE ON biglietto
7   FOR EACH ROW EXECUTE PROCEDURE update_tessera();
8
9 CREATE TRIGGER update_time_biglietto
10 BEFORE INSERT ON biglietto
11   FOR EACH ROW EXECUTE PROCEDURE update_time_biglietto();

```

### 3 Popolamento della base di dati

Per il popolamento della base di dati abbiamo creato dei programmi per la generazione automatica di query. In particolare sono stati creati dei programmi in Java per popolare le tabelle *Passeggero*, *Tratta*, *Volo*. I files .sql prodotti sono pronti per essere eseguiti senza alcuna necessità di modificare il file. Le chiavi esportate vengono gestite direttamente dai programmi.

## Part II

# Progetto del sito web

### 4 Progettazione logica

Si riportano di seguito gli schemi di pagina seguiti durante la creazione del sito.

## 5 Struttura dell'applicazione web

### Part III

## Architettura MVC2

L'elaborato è stato sviluppato seguendo il design pattern MVC2 e seguendo l'approccio servlet-centric. Questo pattern è composto da tre moduli:

- **MODEL** - Comprende la classe DBMS.java e i Java Data Beans. I beans sono contenuti nel package *bean*, mentre la classe DBMS.java è contenuta nel package *database*. DBMS.java si occupa di interrogare il DB e di manipolare i dati al suo interno. La comunicazione con main.java e picture.java avviene tramite i Java Data Beans in ambo le direzioni ;
- **VIEW** - Comprende tutte le JSPs, i javascript, i css e le foto profilo salvate all'interno della base di dati;
- **CONTROLLER** - Comprende le classi main.java, picture.java. Entrambe sono servlet con compiti differenti: picture.java gestisce le richieste di upload e download delle immagini, main.java gestisce il resto delle richieste GET e POST. Utilizza DBMS.java per l'iterazione con la base di dati ed inoltre gli eventuali dati alla JSP appropriata ;

## 6 Model

Fanno parte del modulo Model il DBMS e i Java Data Beans.

## 7 View

Fanno parte del modulo View tutte la JSPs, i javascript e i css.

## 8 Controller

Fanno parte del modulo Controller le due servlet. Le due servlet che controllano il flusso sono *main.java* e *picture.java*.

La servlet *picture.java* gestisce la parte multimediale dell'applicazione, caricando e scaricando dalla base di dati le foto profilo dei passeggeri. La servlet identifica e gestisce le richieste HTTP sulla base che esse siano **GET** o **POST** e dal valore del parametro **ps** passato nella richiesta.

Per le richieste di tipo **GET**, **ps** può assumere questi valori:

- **downloadimage** - Scrive direttamente sullo stream output l'immagine relativa al passeggero specificato. I parametri richiesti sono:
  - **documento** Il documento del passeggero per il quale bisogna caricare l'immagine



- **Parametro assente o stringa vuota** - Viene passato il controllo a *index.jsp*

Per le richieste di tipo **POST**, **ps** può assumere questi valori:

- **uploadimage** - Recupera i dati del passeggero loggato e carica l'immagine passata nella base di dati. In caso di errore passa il controllo a *error.jsp* specificando un messaggio di errore, altrimenti passa il controllo a *bigliettiPage.jsp* dopo aver caricato i relativi dati. I parametri richiesti sono:
  - **image** L'immagine da caricare come foto profilo
- **Parametro assente o stringa vuota** - Viene passato il controllo a *index.jsp*

La servlet *main.java* controlla le richieste da parte delle JSPs. La servlet identifica e gestisce le richieste HTTP sulla base che esse siano **GET** o **POST** e dal valore del parametro **ps** passato nella richiesta.

Per le richieste di tipo **GET**, **ps** può assumere questi valori:

- **areapersonale** - Vengono caricati dalla base di dati i voli e le prenotazioni dell'utente specificato nel parametro *pass*. Viene passato il controllo a *areapersonale.jsp*. Non sono richiesti ulteriori parametri in quanto i dati del passeggero vengono recuperati dall'attributo di sessione *pass*.
- **ricercavolo** - Vengono caricati gli aeroporti di partenza e viene passato il controllo a *ricercavolo.jsp*.
- **prenotazione** - Recupera le informazioni del volo e del passeggero. Infine passa il controllo a *prenotazionePage.jsp*. I parametri richiesti sono:

- **codiceVolo** Codice del volo per recuperare le informazioni

Non serve specificare un parametro per il passeggero in quanto le informazioni vengo recuperate dall'attributo della sessione.

- **logout** - Viene rimosso il passeggero loggato e passa il controllo a *ricercavolo.jsp* insieme agli aeroporti di partenza.
- **contatti** - Viene passato il controllo a *contatti.jsp*
- **emettibiglietto** - Passa il numero della prenotazione a *emettiBigliettoPage.jsp* e gli passa il controllo. I parametri richiesti sono:
  - **numPrenotazione** Numero della prenotazione per la quale emettere il biglietto
- **chisiamo** - Passa il controllo a *chisiamo.jsp*
- **Parametro assente o stringa vuota** - Viene passato il controllo a *index.jsp*

Per le richieste di tipo **POST**, **ps** può assumere questi valori:

- **newbiglietto** - Viene richiesto di emettere un biglietto a partire dal numero della prenotazione. Una volta inserito il biglietto passa il controllo a *bigliettiPage.jsp* con i dati relativi al passeggero loggato, alle sue prenotazioni e ai suoi biglietti. I parametri richiesti sono:
  - **numPrenotazione** Numero della prenotazione per la quale emettere il biglietto
- **login** - Vengono ricevuti la coppia login e password dalla form *authentication* di *login.jsp*. Se la coppia è valida allora passa il controllo a *bigliettiPage.jsp* insieme ai dati del passeggero, delle prenotazioni e dei voli. Altrimenti imposta un messaggio di errore e passa il controllo a *login.jsp*. I parametri richiesti sono:
  - **username** Username del passeggero
  - **password** Password del passeggero
- **nuovaprenotazione** - Vengono inviati i dati dalla form *prenotazione* di *prenotazionePage.jsp*, se i dati sono validi allora si procede ad aggiungere una nuova prenotazione del database. Se il passeggero non esiste ed è la prima volta che effettua una prenotazione allora sarà creato un nuovo passeggero a partire dai dati passati alla servlet. Il controllo poi passa a *esitoPage.jsp* con il relativo messaggio di stato. I parametri richiesti sono:
  - **nome** Nome del passeggero
  - **cognome** Cognome del passeggero
  - **documento** Numero del documento, può essere sia il numero di una carta d'identità sia di un passaporto
  - **nazionalita** Nazionalità del passeggero
  - **username** Username scelto dal passeggero
  - **password** Password scelta dal passeggero
  - **codicevolo** Codice del volo da prenotare
  - **tessera** Richiesta della tessera
- **volipage** - Ricerca un volo a partire dai valori specificati nei parametri passati e passa il controllo a *voliPage.jsp* se esiste almeno una corrispondenza, altrimenti viene passato il controllo a *ricercavolo.jsp*. I parametri richiesti sono:
  - **partenza** Aeroporto di partenza
  - **arrivo** Aeroporto di arrivo
  - **date** Data di partenza del volo
- **Parametro assente o stringa vuota** - Viene passato il controllo a *index.jsp*

La servlet *main.java* è in grado di gestire anche richieste AJAX sempre con lo stesso parametro **ps**. Il parametro può assumere i seguenti valori:

- **ajaxricercavolo** - Data un aeroporto di partenza passato come parametro, ricava tutti i possibili aeroporti di arrivo. Dopo aver recuperato le informazioni invia un messaggio JSON alla JSP chiamante. I parametri da passare sono:
  - **part** Indica l'aeroporto di partenza
- **checkusername** - Dato il nome utente di un passeggero interroga la classe *DBMS.java* se l'username esiste all'interno della base di dati. Se esiste setta il campo **isFree** all'interno del messaggio JSON di risposta a **false**, altrimenti a **true**. I parametri da passare sono:
  - **username** Indica il nome utente da controllare
- **checkdocumento** - Dato il numero di documento interroga la classe *DBMS.java* se il numero di documento esiste all'interno della base di dati. Se esiste setta il campo **isFree** all'interno del messaggio JSON di risposta a **false**, altrimenti a **true**. I parametri da passare sono:
  - **documento** Indica il documento da controllare

## 9 Gestione di errori

Se si verifica un errore su una qualunque pagina JSP, allora viene caricata, automaticamente dal sistema, la pagina *error.jsp*. La pagina riporta la natura dell'errore. Se viene passato il parametro **msg** allora stamperà il messaggio di errore specificato.

## 10 Pagine JSPs e parametri passati

Vengono descritte ora le JSPs utilizzate con i relativi parametri.

- *index.jsp* - Home page del sito. Viene mostrato un menù con il quale è possibile muoversi. La pagina è composta da un iframe a cui interno vengono caricate le singole JSPs.
- *error.jsp* - Se si verifica un errore su una qualunque pagina JSP, allora viene caricata, automaticamente dal sistema, la pagina *error.jsp*. La pagina riporta la natura dell'errore. Se viene passato il parametro **msg** allora stamperà il messaggio di errore specificato. Parametri passati alla pagina:
  - **msg** Messaggio di errore da stampare
- *bigliettiPage.jsp* Parametri passati alla pagina:
  - **prenotazioni**
  - **biglietti**
  - **pass**
- *chisiamo.jsp*
- *contatti.jsp*

- *emettiBigliettoPage.jsp* Parametri passati alla pagina:
  - **numPrenotazione**
- *esitoPage.jsp* Parametri passati alla pagina:
  - **status**
- *login.jsp* Parametri passati alla pagina:
  - **auth**
- *prenotazionePage.jsp* Parametri passati alla pagina:
  - **volo**
  - **pass**
- *ricercavolo.jsp* Parametri passati alla pagina:
  - **partenza**
  - **arrivi**
  - **status**
- *voliPage.jsp* Parametri passati alla pagina:
  - **voli** ArrayList dei voli richiesti

## 11 Path e Context

Il context del sito web è *basi*, mentre il path relativo per raggiungere la servlet è *servlet/main*. La porta del server è la *8080*. Quindi l'url per accedere al sito web è: `http://localhost:8080/basi/servlet/main`

## 12 Login e sessioni

Quando un passeggero esegue il login viene associata una sessione al client relativo, nella quale viene mantenuto l'attributo **pass**. L'attributo contiene l'oggetto relativo al passeggero loggato. L'utilizzo di sessioni implica che il browser del client debba avere l'accettazione di COOKIE attiva per poter navigare sul sito. Ogni sessione dura 600 secondi(10 minuti), dopo di che essa verrà "eliminata" e se avviene un'altra richiesta da parte di un host a cui è scaduta la sessione, allora esso verrà reindirizzato alla *index.jsp*.

## Part IV

# Scelte progettuali

## 13 Hibernate

## 14 AJAX