Università degli Studi di Verona

DIPARTIMENTO DI INFORMATICA
Corso di Laurea in Informatica

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

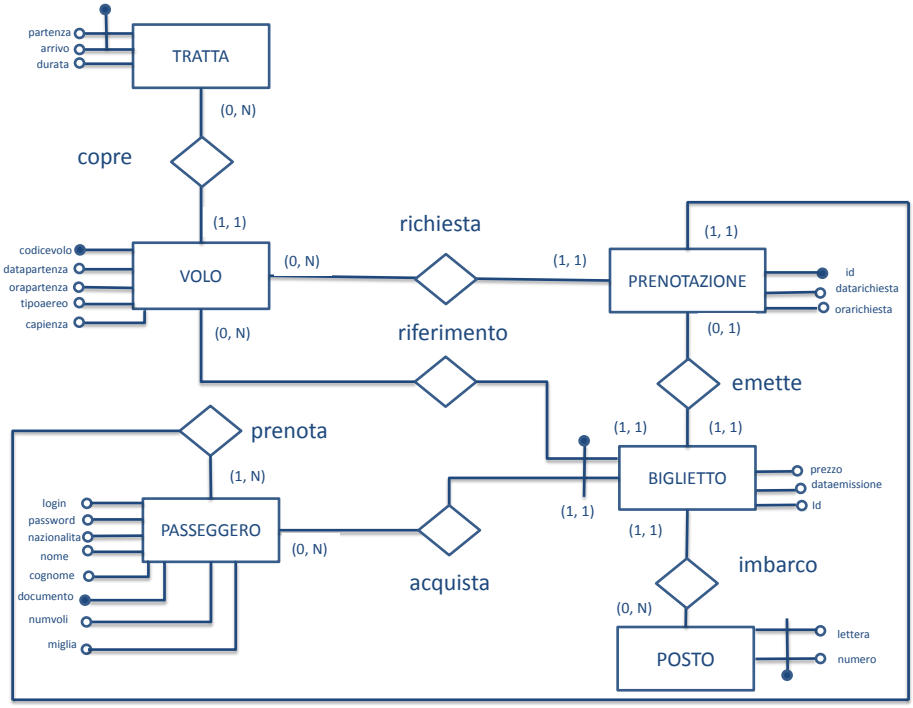
Anno Accademico 2013-2014

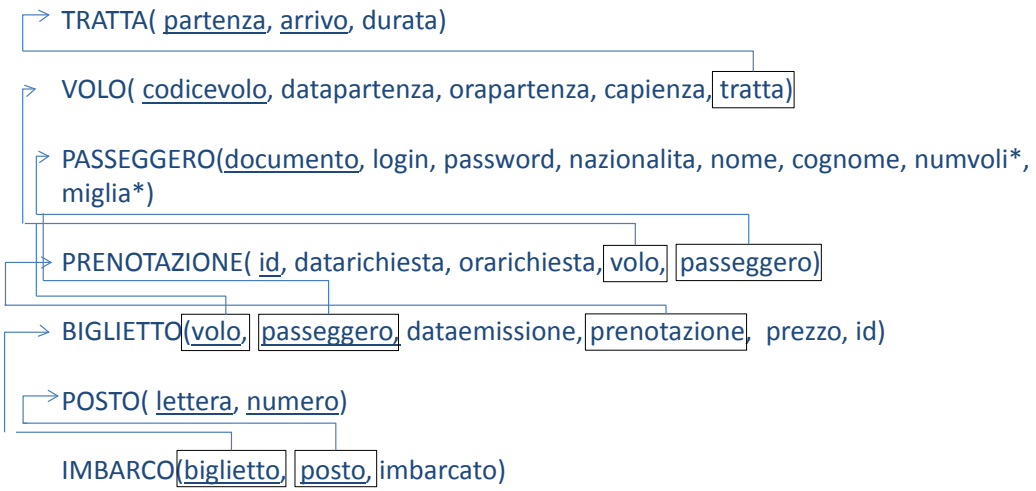
Contents

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.

```
CREATE  TABLE tratta(
  partenza  VARCHAR(100) NOT NULL,
  arrivo    VARCHAR(100) NOT NULL,
  durata    INTEGER NOT NULL,
  distanza  NUMERIC(10,1) NOT NULL DEFAULT '0.0',
  PRIMARY KEY ( partenza, arrivo ),
  UNIQUE ( partenza, arrivo )
);
CREATE TABLE volo(
  codicevolo  VARCHAR(10) NOT NULL,
  partenza  VARCHAR(100) NOT NULL,
  arrivo    VARCHAR(100) NOT NULL,
  datapartenza  DATE NOT NULL,
  orapartenza TIME NOT NULL,
  tipoaereo VARCHAR(50) NOT NULL ,
  capienza  INTEGER NOT NULL,
  PRIMARY KEY( codicevolo ),
  UNIQUE ( codicevolo ),
  FOREIGN KEY( partenza, arrivo )
```

```
      REFERENCES tratta( partenza, arrivo )
    ON UPDATE CASCADE
    ON DELETE CASCADE
);
CREATE TABLE passeggero(
  login    VARCHAR(100) NOT NULL,
  password  VARCHAR(100) NOT NULL,
  nazionalita VARCHAR(100) NOT  NULL,
  nome     VARCHAR(100) NOT NULL,
  cognome    VARCHAR(100) NOT NULL,
  documento VARCHAR(50) NOT NULL,
  picture    BYTEA,

  numvoli    INTEGER DEFAULT 0,
  miglia     FLOAT DEFAULT 0,
  tessera    BOOLEAN DEFAULT FALSE,
  PRIMARY KEY( documento ),
  UNIQUE ( login )
);
CREATE TABLE prenotazione(
  id     SERIAL,
  codicevolo  VARCHAR(10) NOT NULL
      REFERENCES volo( codicevolo )
        ON UPDATE CASCADE
        ON DELETE CASCADE,
  documento VARCHAR(50) NOT NULL
      REFERENCES passeggero( documento )
      ON UPDATE CASCADE
      ON DELETE CASCADE,
  datarichiesta DATE,
  orarichiesta  TIME,
  PRIMARY KEY( id ),
  UNIQUE ( codicevolo, documento )
);
CREATE TABLE biglietto(
  id     SERIAL,
  codicevolo  VARCHAR(10) NOT NULL
      REFERENCES volo( codicevolo )
      ON UPDATE CASCADE
      ON DELETE CASCADE,
  documento VARCHAR(50) NOT NULL
      REFERENCES passeggero( documento )
      ON UPDATE CASCADE
      ON DELETE CASCADE,
  id_prenotazione INTEGER NOT NULL
        REFERENCES prenotazione( id )
        ON UPDATE CASCADE
        ON DELETE CASCADE,
  dataemissione DATE NOT NULL,
  prezzo     NUMERIC(10,2) NOT NULL,

  PRIMARY KEY( codicevolo, documento ),
  UNIQUE ( codicevolo, documento, id )
);

CREATE TABLE posto(
  lettera    CHAR NOT NULL,
  numero     INTEGER DEFAULT 0,
  PRIMARY KEY ( lettera, numero ),
  UNIQUE ( lettera, numero )
);
CREATE TABLE imbarco(
```

```
  lettera    CHAR NOT NULL,
  numero     INTEGER DEFAULT 0,
  imbarcato BOOLEAN DEFAULT FALSE,
  -- PRIMARY KEY DEL BIGLIETTO
  codicevolo  VARCHAR(10) NOT NULL,
  documento VARCHAR(50) NOT NULL,
  -- IDENTIFICATO DAL BIGLIETTO E DAL POSTO
  PRIMARY KEY ( codicevolo, documento, lettera, numero),

  FOREIGN KEY( lettera, numero )
      REFERENCES posto( lettera,numero )
    ON UPDATE CASCADE
    ON DELETE CASCADE,
  FOREIGN KEY( codicevolo, documento )
      REFERENCES biglietto( codicevolo,documento )
    ON UPDATE CASCADE
    ON DELETE CASCADE
);
```

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.

```
CREATE OR REPLACE FUNCTION update_time_richiesta()
RETURNS TRIGGER AS '
BEGIN
  IF NEW.datarichiesta IS NULL THEN
    NEW.datarichiesta := current_date;
  END IF;
  IF NEW.orarichiesta IS NULL THEN
    NEW.orarichiesta := current_time;
    END IF;
  RETURN NEW;
END' LANGUAGE 'plpgsql';



CREATE OR REPLACE FUNCTION update_time_biglietto()
RETURNS TRIGGER AS '
BEGIN
  IF NEW.dataemissione IS NULL THEN
    NEW.dataemissione := current_date;
  END IF;
  RETURN NEW;
END' LANGUAGE 'plpgsql';



CREATE OR REPLACE FUNCTION update_numvoli()
RETURNS TRIGGER AS '
BEGIN
  IF NEW.dataemissione IS NULL THEN
    NEW.dataemissione := current_date;
  END IF;
  RETURN NEW;
END' LANGUAGE 'plpgsql';

CREATE OR REPLACE FUNCTION update_tessera() RETURNS trigger AS $$
BEGIN
  UPDATE passeggero
  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 seguiti 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 rihieste 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 javabeans.

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. Per ogni richiesta viene passato il parametro *ps* con un determinato valore e può assumere questi valori:

- uploadimage
- downloadimage

La servlet *main.java* controlla le richieste da parte delle JSPs.

- **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*.
- **volipage** - Ricerca un volo a partire dai valori specificati nei parametri passati e passa il controllo a *voliPage.jsp* se esiste almeno una corrispondeza, 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
- **prenotazione**
- **nuovaprenotazione**
- **logout**
- **login**
- **contatti**
- **emettibiglietto**
- **newbiglietto**
- **chisiamo**
- **ajaxricercavolo**
- **checkusername**
- **checkdocumento**

9 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`

Part IV
Scelte progettuali

10 Hibernate

11 AJAX

12 Session