



**UNIVERSITÀ DEGLI STUDI DI PADOVA**

# **Relazione Progetto Basi di Dati**

*Anno Accademico 2014-2015*



## **Studenti:**

Gianluca Pegoraro (1049140)

Davide Scarparo (1049135)

**Indirizzo email referente:** [davide.scarparo@studenti.unipd.it](mailto:davide.scarparo@studenti.unipd.it)

**URL di accesso:** <http://basidati.studenti.math.unipd.it/basidati/~dscarpar/>

**Classi di utenza:** admin (amministratore), cliente (generico utente)

- **Utente Admin:** admin@gmail.com (mail) – admin (password)
- **Utente Cliente:** cliente@gmail.com (mail) – 123456 (password)

# **Indice**

## **1 Introduzione**

- 1.1 Abstract
- 1.2 Analisi dei requisiti

## **2 Progettazione concettuale**

- 2.1 Entità e gerarchie
- 2.2 Relazioni
- 2.3 Modello ER

## **3 Progettazione logica**

- 3.1 Eliminazione gerarchie
- 3.2 Traduzione verso il modello logico
- 3.3 Schema logico

## **4 Implementazione dello schema logico**

## **5 SQL**

- 5.1 Query
- 5.2 Procedure e funzioni
- 5.3 Trigger

## **6 PHP**

## **7 Note**

# **1 Introduzione**

## **1.1 Abstract**

Il progetto SkyDreamer si propone di modellare il booking online di biglietti per la suddetta compagnia aerea.

Nel corso degli anni, volare è diventata una cosa accessibile a tutti, grazie alla nascita del concetto di volo low cost: le nuove compagnie aeree offrono a prezzi molto spesso convenienti, biglietti aerei per viaggiare in giro per il mondo.

Con l'avvento di internet, cercare e prenotare il proprio viaggio non è mai stato così facile; SkyDreamer propone un'interfaccia web semplice e intuitiva per queste funzioni fondamentali, ovvero la ricerca di voli a seconda delle destinazioni dell'utente, permettendogli di confrontare e scegliere il volo a lui più congeniale, la possibilità di prenotare biglietti e di scegliere la classe di volo, raccogliere punti fedeltà per poterne usufruire in sede di acquisto.

## **1.2 Analisi dei requisiti**

Il progetto consiste nella realizzazione di una base di dati e della relativa interfaccia web, con la quale gli utenti possono interagire ed effettuare le operazioni descritte in precedenza.

Gli utenti si distinguono in due categorie, clienti ed amministratori: questi ultimi possono effettuare operazioni di manutenzione del database (inserimento, modifica e cancellazione di record), rispetto alle più semplici operazioni concesse ai normali clienti (visualizzazione storico ordini, informazioni sul proprio profilo, ricerca e prenotazione di voli).

Ogni utente è identificato univocamente da una mail, con relativa password, e in particolar modo di un utente interessano cognome, nome, data di nascita e recapito telefonico.

Quando un utente si registra al sito, gli viene automaticamente associata una carta fedeltà, la quale possiede un numero univoco e raccoglie i punti.

Tali punti potranno essere utilizzati per ottenere uno sconto durante la prenotazione di biglietti per un dato volo.

Ogni prenotazione è identificata da un codice, e di una prenotazione interessano la data nella quale è avvenuta, il numero di biglietti acquistati, la classe di volo e il costo per singolo biglietto che si è pagato in quella prenotazione (a seconda della classe di volo e se si sono utilizzati i punti fedeltà).

Un volo è identificato univocamente da un codice, e di un volo interessano l'aeroporto di partenza, l'aeroporto di arrivo, la data e l'orario di partenza, il prezzo e la durata della tratta.

Ogni aereo è identificato da un codice univoco, e di un aereo interessano il modello e il numero di posti passeggeri.

Ogni aeroporto è identificato da un codice, e di un aeroporto interessano il nome, la città, la nazione e il numero di piste.

Infine, si noti che:

- un aeroporto può essere sia di partenza che di arrivo;
- il nome dell'aeroporto è univoco (non esistono aeroporti con lo stesso nome, al massimo hanno uguale città);
- non è possibile registrare un utente al sito se inserisce un numero di telefono che è già stato registrato;
- un amministratore non può effettuare prenotazioni di voli, quindi non possiede una carta per la raccolta punti;
- solo un utente di tipo amministratore può inserire, modificare e cancellare voli, aeroporti e aerei;
- ogni biglietto relativo ad una prenotazione subisce un incremento di prezzo a seconda della classe di volo scelta dall'utente durante la prenotazione (+50€ per la prima classe, +20€ per la classe business, nessuna variazione di prezzo per la classe economy);
- per ogni prenotazione, è possibile richiedere fino a un massimo di 5 biglietti;
- la prenotazione viene effettuata se e solo se i posti disponibili per il volo richiesto sono sufficienti;
- per ogni biglietto acquistato, il cliente guadagna 10 punti sulla carta;
- l'utente può decidere se e quando utilizzare i proprio punti accumulati: ogni 10 punti, lo sconto per ogni biglietto prenotato è di 1€;
- nel caso un volo venga cancellato, si vogliono mantenere le informazioni riguardanti la prenotazione.

## 2 Progettazione concettuale

### 2.1 Entità e gerarchie

**utente**: rappresenta tutti gli utenti registrati al sito.

mail: string <<PK>>  
password: string  
cognome: string  
nome: string  
dataNascita: date  
telefono: string

**cliente**: rappresenta gli utenti di tipo cliente che possono prenotare voli.

**amministratore**: rappresenta gli utenti di tipo amministratore che possono effettuare operazioni di manutenzione del database.

**N.B.:** cliente e amministratore sono una generalizzazione di tipo totale dell'entità utente.

**carta**: rappresenta tutte le carte dei clienti registrati al sito.

codiceCarta: string <<PK>>  
punti: int

**prenotazione**: rappresenta le prenotazioni effettuate.

idPrenotazione: int <<PK>>  
dataPrenotazione: date  
numeroBiglietti: int  
costoBiglietto: double  
classe: enum (prima, business, economy)

**volo**: rappresenta i voli disponibili (per cui si può effettuare la prenotazione di almeno 1 biglietto).

idVolo: string <<PK>>  
dataPartenza: datetime  
costo: double  
durataTratta: time

**aereo**: rappresenta gli aerei di linea che sono impiegati nei voli della compagnia aerea.

idAereo: string <<PK>>  
modello: string  
numeroPosti: int

**aeroporto:** rappresenta gli aeroporti da cui partono/arrivano voli della compagnia aerea.

codiceAeroporto: string <<PK>>  
nomeAeroporto: string  
città: string  
nazione: string  
numeroPiste: int

## 2.2 Relazioni

### **cliente-carta: possiede**

Ogni cliente possiede una carta per la raccolta di punti.  
Una carta è intestata ad uno specifico cliente.

**Molteplicità:** 1:1

**Totalità:**

Totale da cliente a carta  
Totale da carta a cliente

### **cliente-prenotazione: effettua**

Un cliente effettua zero o più prenotazioni.  
Ogni prenotazione è relativa ad un unico cliente.

**Molteplicità:** 1:N

**Totalità:**

Parziale da cliente a prenotazione  
Totale da prenotazione a cliente

### **prenotazione-volo: relativa a**

Ogni prenotazione è relativa ad un volo.  
Per un volo possono essere effettuate zero o più prenotazioni.

**Molteplicità:** 1:N

**Totalità:**

Totale da prenotazione a volo  
Parziale da volo a prenotazione

### **aereo-volo: partecipa**

Ogni aereo vola per zero o più volte (partecipa a zero o più voli).  
Ogni volo è effettuato da un unico aereo.

**Molteplicità:** 1:N

**Totalità:**

Parziale da aereo a volo  
Totale da volo a aereo

### **aereo-aeroporto: viaggia**

Ogni aereo viaggia in uno o più aeroporti.

Da ogni aeroporto volano uno o più aerei.

**Molteplicità:** M:N

**Totalità:**

Totale da aereo a aeroporto

Totale da aeroporto a aereo

### **volo-aeroporto: parte**

Ogni volo parte da un solo aeroporto.

In un aeroporto arrivano uno o più voli.

**Molteplicità:** 1:N

**Totalità:**

Totale da volo a aeroporto

Totale da aeroporto a volo

### **voli-aeroporto: arriva**

Ogni volo arriva in un solo aeroporto.

Da un aeroporto partono uno o più voli.

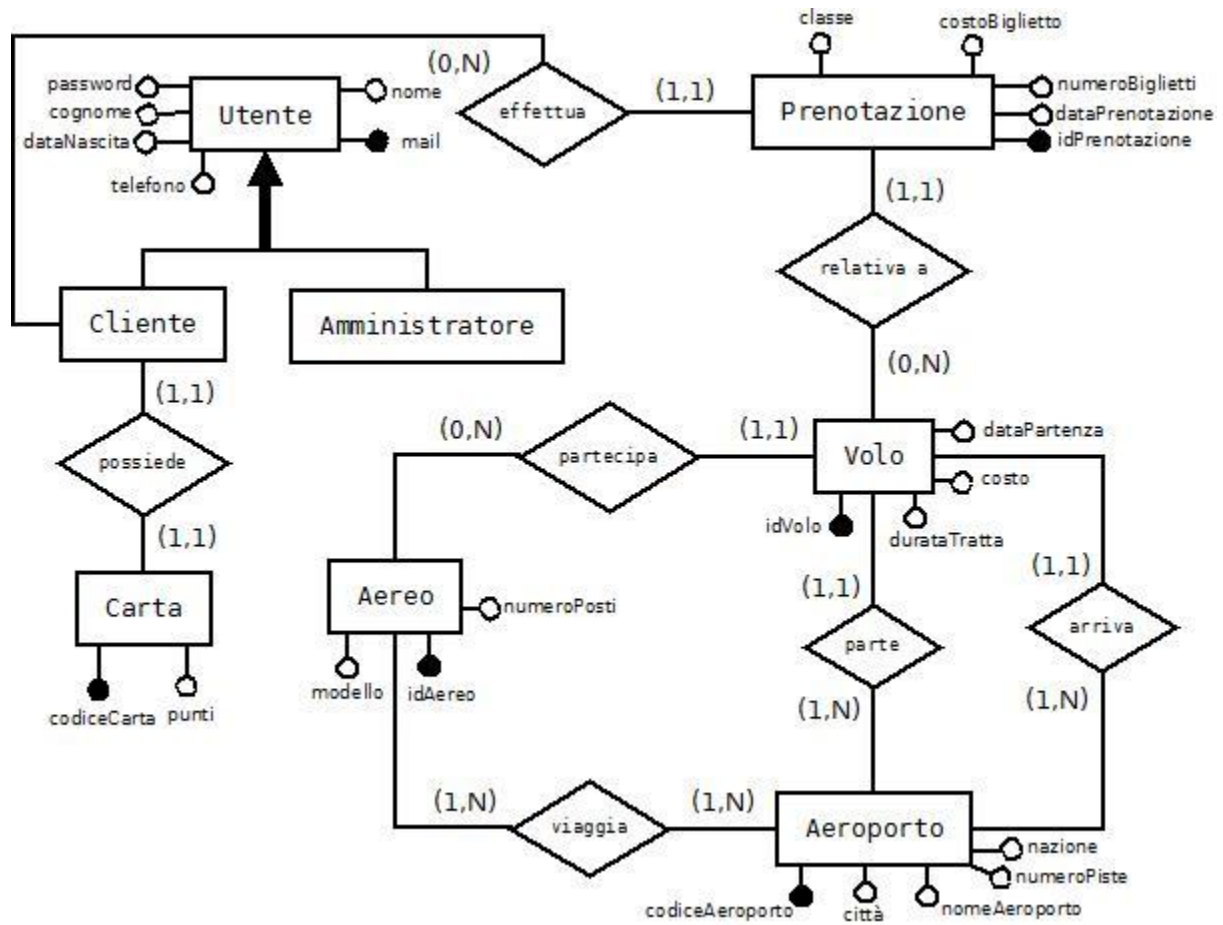
**Molteplicità:** 1:N

**Totalità:**

Totale da volo a aeroporto

Totale da aeroporto a volo

## 2.3 Modello ER



*modelloER.jpeg*



## 3 Progettazione logica

### 3.1 Eliminazione gerarchie

Per l'eliminazione della generalizzazione, si è scelta la soluzione che prevede l'accorpamento delle entità figlie nell'entità padre, perchè le entità figlie non hanno attributi specifici in più rispetto a quelli dell'entità padre, e quindi è sufficiente aggiungere nell'entità padre un attributo che discrimini le varie entità figlie.

Nel caso specifico, nell'entità utente si aggiunge l'attributo *admin* di tipo boolean, poiché la generalizzazione è di tipo totale, per cui un utente può essere o un cliente o un amministratore.

### 3.2 Traduzione verso il modello logico

Ogni entità presente nel modello ER diventa una relazione con lo stesso nome, avente per attributi i medesimi attributi dell'entità e per chiave il suo identificatore.

Un'associazione molti a molti diventa una relazione con lo stesso nome, avente per attributi gli attributi dell'associazione e gli identificatori delle entità coinvolte; tali identificatori formano la chiave primaria della relazione.

Nel caso di un'associazione uno a molti, si aggiunge un vincolo di integrità referenziale nella relazione con cardinalità massima 1 (chiave esterna).

Nel caso di un'associazione uno a uno, è possibile rappresentare l'associazione in una qualunque delle relazioni che rappresentano le due entità.

Nello specifico caso del modello E-R costruito precedentemente, si ottiene il seguente schema logico:

**utente** (mail, password, cognome, nome, dataNascita, telefono, admin)

**carta** (codiceCarta, mailUtente, punti)

La relazione **possiede** viene rappresentata attraverso un vincolo di integrità referenziale fra mailUtente in carta e mail in utente (chiave esterna in carta verso utente).

**aereo** (idAereo, modello, numeroPosti)

**aeroporto** (codiceAeroporto, nomeAeroporto, città, nazione, numeroPiste)

Vincolo di unicità su nomeAeroporto.

**volo** (idVolo, aeroportoPartenza, aeroportoArrivo, idAereoVolo, dataPartenza, costo, durataTratta)

La relazione **parte** viene rappresentata attraverso un vincolo di integrità referenziale fra aeroportoPartenza in volo e codiceAeroporto in aeroporto (chiave esterna in volo verso aeroporto).

La relazione **arriva** viene rappresentata attraverso un vincolo di integrità referenziale fra aeroportoArrivo in volo e codiceAeroporto in aeroporto (chiave esterna in volo verso aeroporto).

La relazione **partecipa** viene rappresentata attraverso un vincolo di integrità referenziale fra idAereoVolo in volo e idAereo in aereo (chiave esterna in volo verso aereo).

**prenotazione** (idPrenotazione, mailUtente, idVoloPren, dataPrenotazione, numeroBiglietti, costoBiglietto, classe)

La relazione **effettua** viene rappresentata attraverso un vincolo di integrità referenziale fra mailUtente in prenotazione e mail in utente (chiave esterna in prenotazione verso utente).

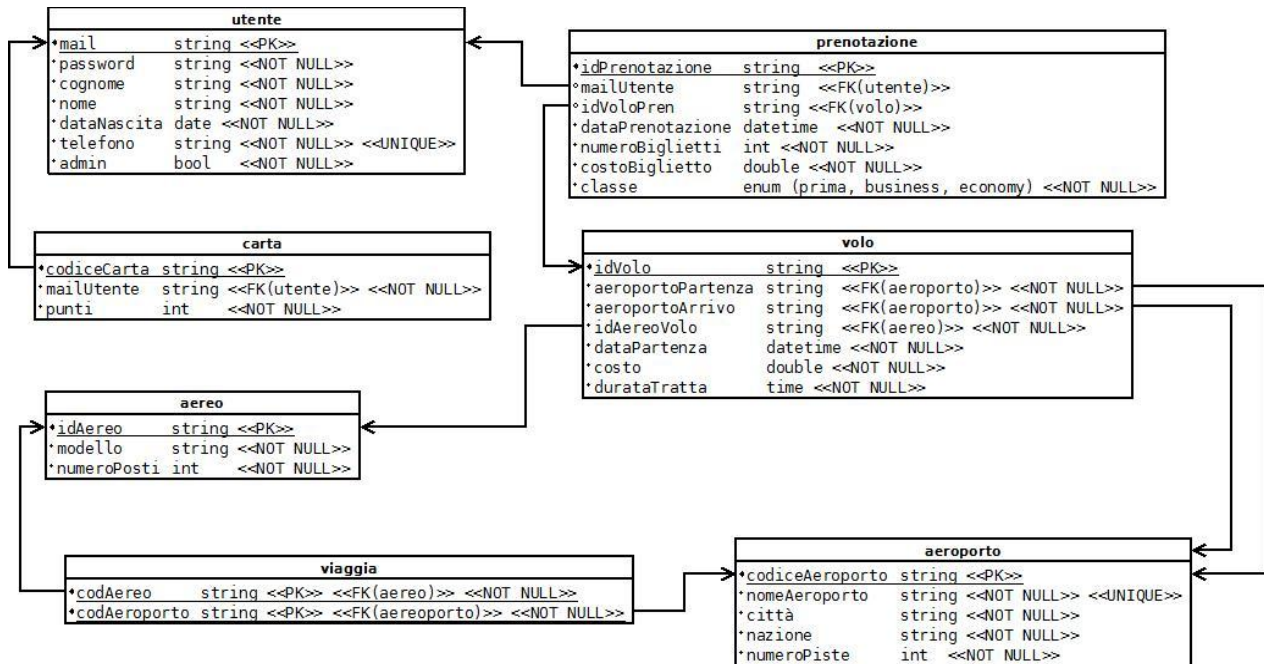
La relazione **relativa a** viene rappresentata attraverso un vincolo di integrità referenziale fra idVoloPren in prenotazione e idVolo in volo (chiave esterna in prenotazione verso volo).

**viaggia** (codAereo, codAeroporto)

La relazione **viaggia** viene rappresentata attraverso una nuova entità con lo stesso nome e attributi codAereo (vincolo di integrità referenziale fra codAereo in viaggia e idAereo in aereo --> chiave esterna in viaggia verso aereo) e vincolo di integrità referenziale fra codAeroporto in viaggia e codiceAeroporto in aeroporto --> chiave esterna in viaggia verso aeroporto).

La chiave primaria della nuova entità è la coppia dei due attributi chiave esterne.

### 3.3 Schema logico



schemaLogico.jpeg

## 4 Implementazione della base di dati

**CREATE TABLE** utente (

mail	varchar(40) <b>PRIMARY KEY</b> ,
password	varchar(20) <b>NOT NULL</b> ,
cognome	varchar(20) <b>NOT NULL</b> ,
nome	varchar(20) <b>NOT NULL</b> ,
dataNascita	date <b>NOT NULL</b> ,
telefono	varchar(10) <b>NOT NULL UNIQUE</b> ,
admin	boolean <b>NOT NULL</b>

) **ENGINE=InnoDB**;

**CREATE TABLE** carta (

codiceCarta	varchar(20) <b>PRIMARY KEY</b> ,
mailUtente	varchar(40) <b>NOT NULL</b> ,
punti	int(5),
<b>FOREIGN KEY</b> (mailUtente) <b>REFERENCES</b> utente (mail) <b>ON DELETE CASCADE</b> <b>ON UPDATE CASCADE</b>	

) **ENGINE=InnoDB**;

**CREATE TABLE** aereo (

idAereo	varchar(10) <b>PRIMARY KEY</b> ,
modello	varchar(20) <b>NOT NULL</b> ,
numeroPosti	int(3) <b>NOT NULL</b>

) **ENGINE=InnoDB**;

**CREATE TABLE** aeroporto (

codiceAeroporto	varchar(10) <b>PRIMARY KEY</b> ,
nomeAeroporto	varchar(30) <b>NOT NULL UNIQUE</b> ,
citta	varchar(20) <b>NOT NULL</b> ,
nazione	varchar(20) <b>NOT NULL</b> ,
numeroPiste	int(3) <b>NOT NULL</b>

) **ENGINE=InnoDB**;

**CREATE TABLE** viaggio (

codAereo	varchar (10),
codAeroporto	varchar (10),
<b>FOREIGN KEY</b> (codAereo) <b>REFERENCES</b> aereo (idAereo) <b>ON DELETE</b> <b>CASCADE ON UPDATE CASCADE</b> ,	
<b>FOREIGN KEY</b> (codAeroporto) <b>REFERENCES</b> aeroporto (codiceAeroporto) <b>ON</b> <b>DELETE CASCADE ON UPDATE CASCADE</b> ,	
<b>PRIMARY KEY</b> (codAereo,codAeroporto)	

) **ENGINE=InnoDB**;

```
CREATE TABLE prenotazione (  
    idPrenotazione    varchar(10) PRIMARY KEY,  
    mailUtente        varchar(40),  
    idVoloPren        varchar(10),  
    dataPrenotazione  datetime NOT NULL,  
    numeroBiglietti   int(2) NOT NULL,  
    costoBiglietto    double NOT NULL,  
    classe            enum('prima','business','economy') NOT NULL,  
    FOREIGN KEY (mailUtente) REFERENCES utente (mail) ON DELETE SET NULL  
    ON UPDATE CASCADE,  
    FOREIGN KEY (idVoloPren) REFERENCES volo (idVolo) ON DELETE SET NULL  
    ON UPDATE CASCADE  
) ENGINE=InnoDB;
```

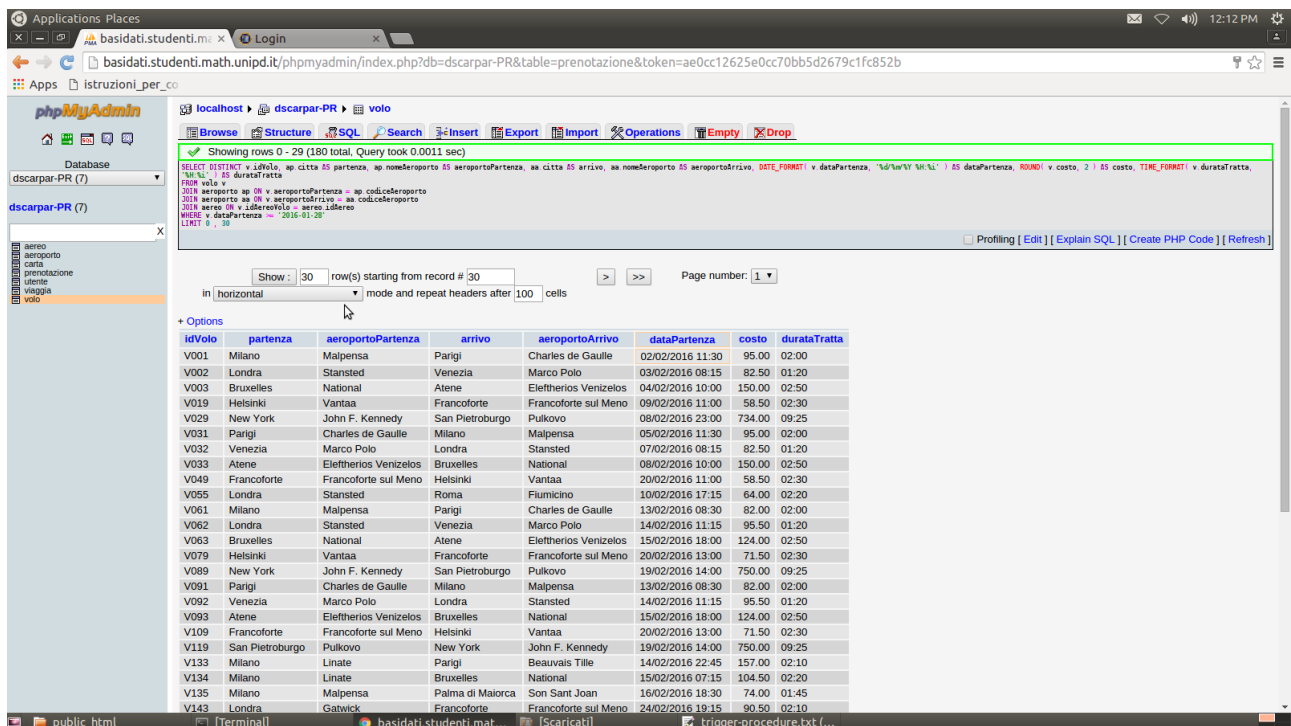
```
CREATE TABLE volo (  
    idVolo            varchar(10) PRIMARY KEY,  
    aeroportoPartenza varchar(10) NOT NULL,  
    aeroportoArrivo   varchar(10) NOT NULL,  
    idAereoVolo        varchar(10) NOT NULL,  
    dataPartenza       datetime NOT NULL,  
    costo              double NOT NULL,  
    durataTratta       time NOT NULL,  
    FOREIGN KEY (aeroportoPartenza) REFERENCES aeroporto (codiceAeroporto)  
    ON DELETE CASCADE ON UPDATE CASCADE,  
    FOREIGN KEY (aeroportoArrivo) REFERENCES aeroporto (codiceAeroporto)  
    ON DELETE CASCADE ON UPDATE CASCADE,  
    FOREIGN KEY (idAereoVolo) REFERENCES aereo (idAereo) ON DELETE  
    CASCADE ON UPDATE CASCADE  
) ENGINE=InnoDB;
```

## 5 SQL

### 5.1 Query

a) Trovare tutti i voli che partono dalla città x e arrivano alla città y, dalla data z.

```
SELECT v.idVolo, ap.nomeAeroporto AS AeroportoPartenza, aa.nomeAeroporto AS
AeroportoArrivo, DATE_FORMAT(v.dataPartenza,'%d/%m/%Y %H:%i') AS
DataPartenza, ROUND(v.costo, 2) AS Costo, TIME_FORMAT(v.durataTratta, '%H:%i')
AS Durata
FROM volo v JOIN aeroporto ap ON v.aeroportoPartenza=ap.codiceAeroporto JOIN
aeroporto aa ON v.aeroportoArrivo=aa.codiceAeroporto
WHERE ap.citta='x' AND aa.citta='y' AND v.dataPartenza>='z'
```



The screenshot shows the phpMyAdmin interface with a SQL query executed. The query is:   
SELECT DISTINCT v.idVolo, ap.citta AS partenza, ap.nomeAeroporto AS aeroportoPartenza, aa.citta AS arrivo, aa.nomeAeroporto AS aeroportoArrivo, DATE\_FORMAT(v.dataPartenza, '%d/%m/%Y %H:%i') AS dataPartenza, ROUND(v.costo, 2) AS costo, TIME\_FORMAT(v.durataTratta, '%H:%i') AS durataTratta FROM volo v JOIN aeroporto ap ON v.aeroportoPartenza = ap.codiceAeroporto JOIN aeroporto aa ON v.aeroportoArrivo = aa.codiceAeroporto WHERE ap.citta = 'x' AND aa.citta = 'y' AND v.dataPartenza >= 'z' LIMIT 0, 30  
The results are displayed in a table with the following columns: idVolo, partenza, aeroportoPartenza, arrivo, aeroportoArrivo, dataPartenza, costo, durataTratta. The table contains 29 rows of flight data.

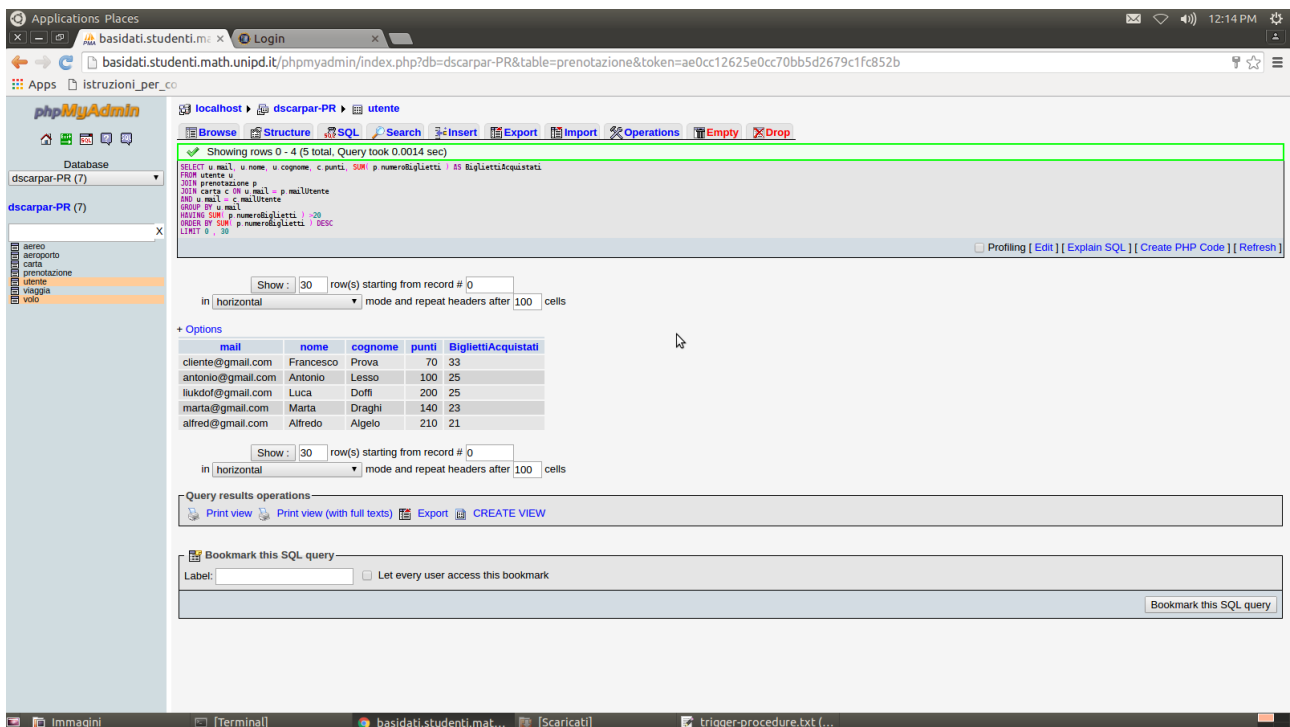
idVolo	partenza	aeroportoPartenza	arrivo	aeroportoArrivo	dataPartenza	costo	durataTratta
V001	Milano	Malpensa	Parigi	Charles de Gaulle	02/02/2016 11:30	95.00	02:00
V002	Londra	Stansted	Venezia	Marco Polo	03/02/2016 08:15	82.50	01:20
V003	Bruxelles	National	Atene	Eleftherios Venizelos	04/02/2016 10:00	150.00	02:50
V019	Helsinki	Vantaa	Francoforte	Francoforte sul Meno	09/02/2016 11:00	58.50	02:30
V029	New York	John F. Kennedy	San Pietroburgo	Pulkovo	08/02/2016 23:00	734.00	09:25
V031	Parigi	Charles de Gaulle	Milano	Malpensa	05/02/2016 11:30	95.00	02:00
V032	Venezia	Marco Polo	Londra	Stansted	07/02/2016 08:15	82.50	01:20
V033	Atene	Eleftherios Venizelos	Bruxelles	National	08/02/2016 10:00	150.00	02:50
V049	Francoforte	Francoforte sul Meno	Helsinki	Vantaa	20/02/2016 11:00	58.50	02:30
V055	Londra	Stansted	Roma	Fiumicino	10/02/2016 17:15	64.00	02:20
V061	Milano	Malpensa	Parigi	Charles de Gaulle	13/02/2016 08:30	82.00	02:00
V062	Londra	Stansted	Venezia	Marco Polo	14/02/2016 11:15	95.50	01:20
V063	Bruxelles	National	Atene	Eleftherios Venizelos	15/02/2016 18:00	124.00	02:50
V079	Helsinki	Vantaa	Francoforte	Francoforte sul Meno	20/02/2016 13:00	71.50	02:30
V089	New York	John F. Kennedy	San Pietroburgo	Pulkovo	19/02/2016 14:00	750.00	09:25
V091	Parigi	Charles de Gaulle	Milano	Malpensa	13/02/2016 08:30	82.00	02:00
V092	Venezia	Marco Polo	Londra	Stansted	14/02/2016 11:15	95.50	01:20
V093	Atene	Eleftherios Venizelos	Bruxelles	National	15/02/2016 18:00	124.00	02:50
V109	Francoforte	Francoforte sul Meno	Helsinki	Vantaa	20/02/2016 13:00	71.50	02:30
V119	San Pietroburgo	Pulkovo	New York	John F. Kennedy	19/02/2016 14:00	750.00	09:25
V133	Milano	Linate	Parigi	Beauvais Tillé	14/02/2016 22:45	157.00	02:10
V134	Milano	Linate	Bruxelles	National	15/02/2016 07:15	104.50	02:20
V135	Milano	Malpensa	Palma di Maiorca	Son Sant Joan	16/02/2016 18:30	74.00	01:45
V143	Londra	Gatwick	Francoforte	Francoforte sul Meno	24/02/2016 19:15	90.50	02:10

queryA.png

b) Trovare mail, cognome, nome e punti sulla carta dei clienti che hanno acquistato più di n biglietti (in ordine per numero di biglietti acquistati).

```
SELECT u.mail, u.nome, u.cognome, c.punti, SUM(p.numeroBiglietti) AS
BigliettiAcquistati
FROM utente u JOIN prenotazione p JOIN carta c ON u.mail=p.mailUtente AND
u.mail=c.mailUtente
GROUP BY u.mail
```

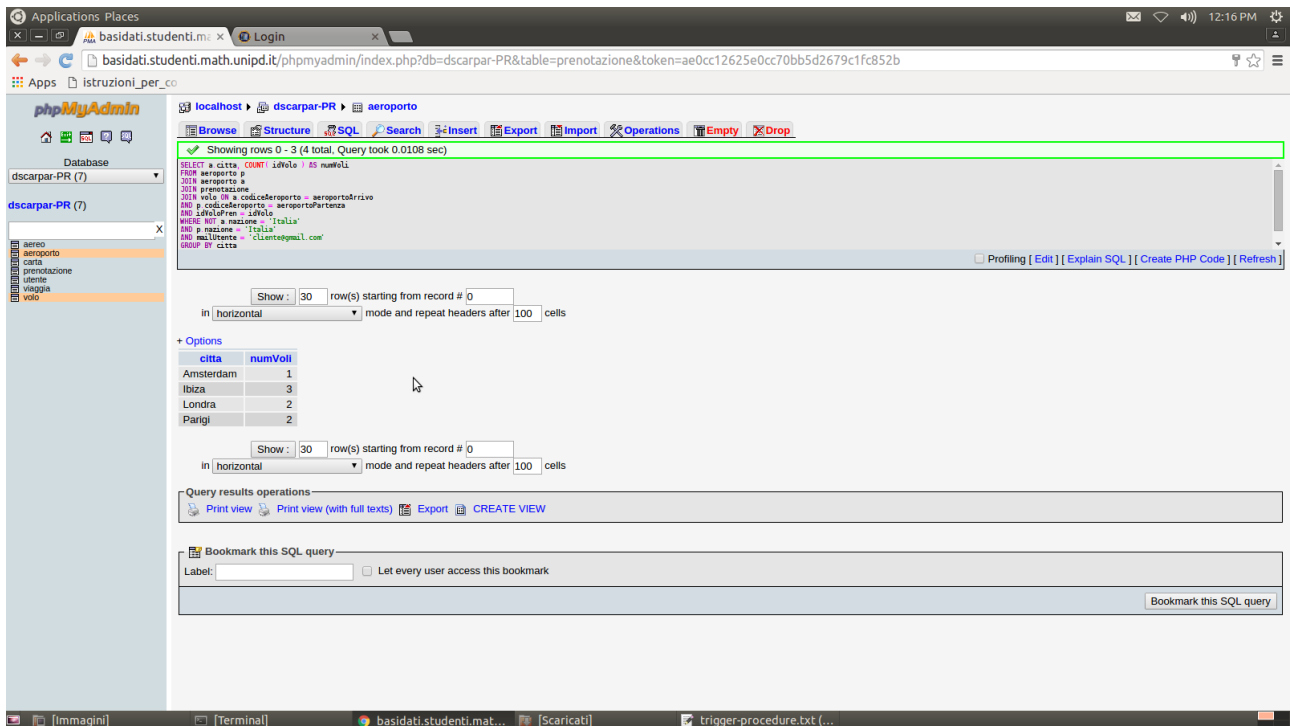
HAVING SUM(p.numeroBiglietti)>n  
ORDER BY SUM(p.numeroBiglietti)



queryB.png

**c) Trovare il numero di voli internazionali (dall'Italia all'estero) effettuati dal cliente x, divisi per città di destinazione.**

```
SELECT a.citta, COUNT(idVolo) AS numeroVoli
FROM aeroporto p JOIN aeroporto a JOIN prenotazione JOIN volo ON
p.codiceAeroporto=aeroportoPartenza AND a.codiceAeroporto=aeroportoArrivo AND
idVoloPren=idVolo
WHERE NOT a.nazione='Italia' AND p.nazione='Italia' AND mailUtente='x'
GROUP BY a.citta
```

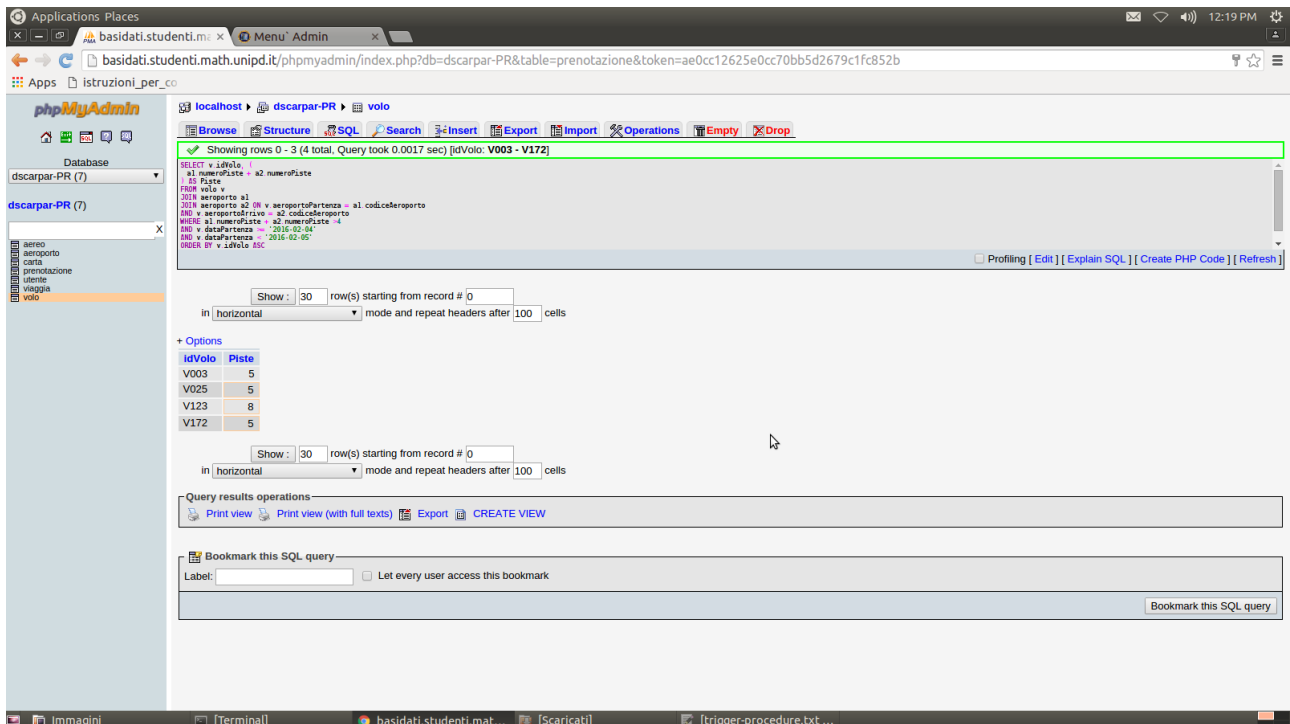


queryC.png

**d) Trovare i voli che viaggiano in data z, la cui somma del numero di piste degli aeroporti coinvolti è maggiore di x.**

```
SELECT v.idVolo, a1.nomeAeroporto AS Partenza, a2.nomeAeroporto AS Arrivo,
(a1.numeroPiste+a2.numeroPiste) AS Piste
FROM volo v JOIN aeroporto a1 JOIN aeroporto a2 ON
v.aeroportoPartenza=a1.codiceAeroporto AND v.aeroportoArrivo=a2.codiceAeroporto
WHERE a1.numeroPiste+a2.numeroPiste>x AND v.dataPartenza>='z' AND
v.dataPartenza<'z+1'
ORDER BY v.IdVolo ASC
```

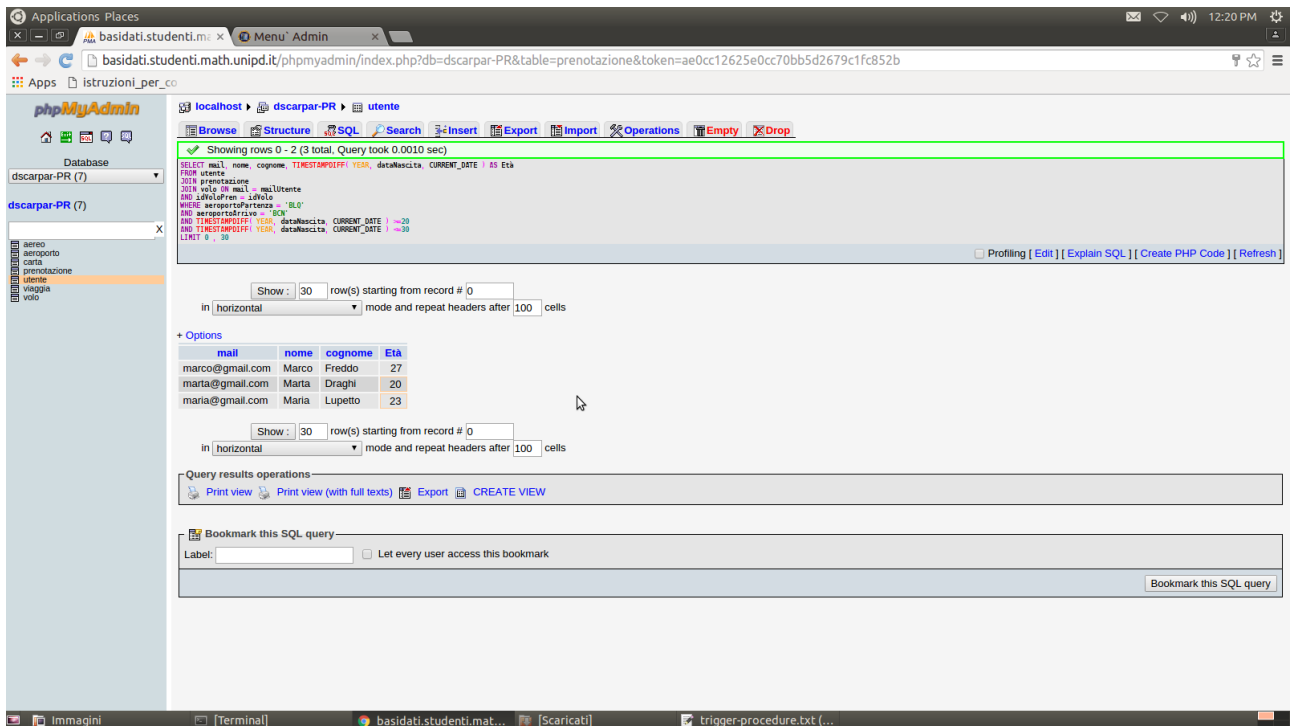




queryD.png

e) Trovare i clienti con età compresa tra  $k$  e  $n$  anni, che hanno viaggiato dall'aeroporto  $x$  all'aeroporto  $y$ .

```
SELECT mail, nome, cognome, TIMESTAMPDIFF(year, dataNascita, CURRENT_DATE)
AS Anni
FROM utente JOIN prenotazione JOIN volo ON mail=mailUtente AND
idVoloPren=idVolo
WHERE aeroportoPartenza='x' AND aeroportoArrivo='y' AND TIMESTAMPDIFF(year,
dataNascita, CURRENT_DATE)>=k AND TIMESTAMPDIFF(year, dataNascita,
CURRENT_DATE)<=n
```



*queryE.png*

**f) Trovare città, aeroporto (partenza e arrivo) e numero di biglietti dei voli che hanno venduto di piú.**

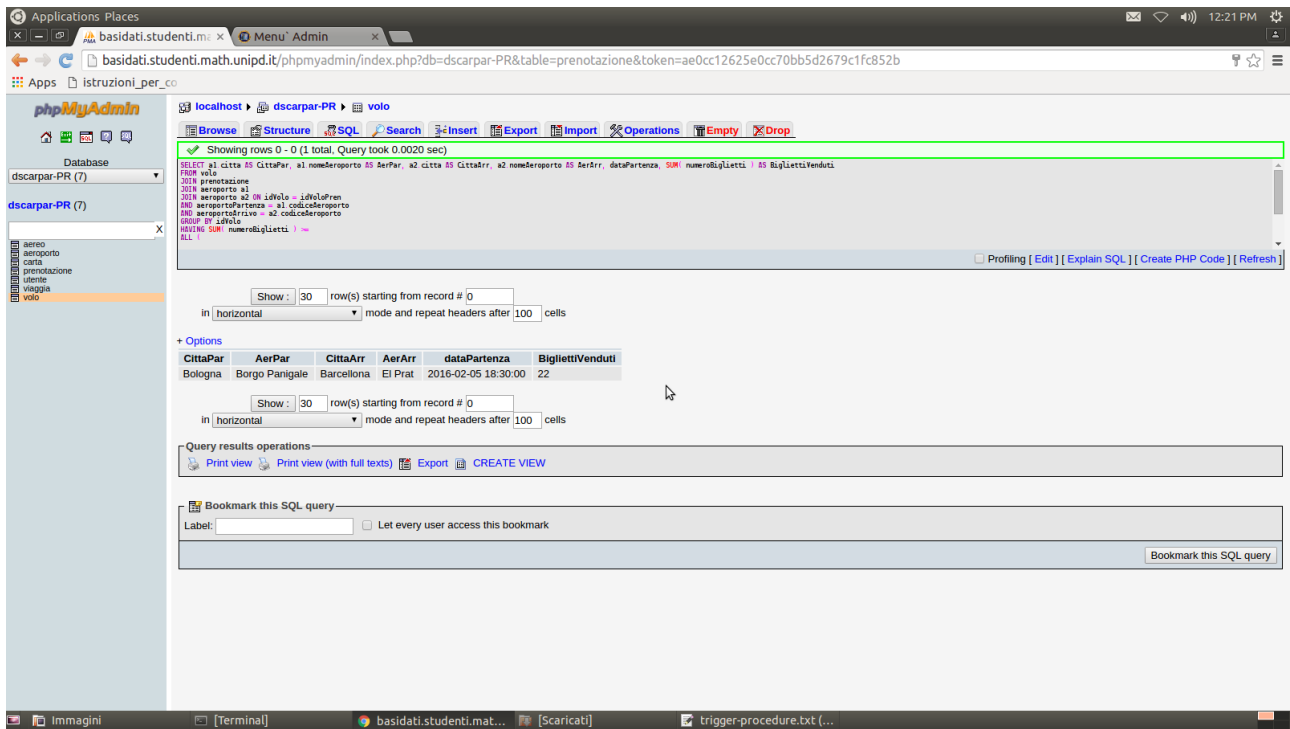
```
SELECT a1.citta AS CittaPartenza, a1.nomeAeroporto AS AeroportoPartenza, a2.citta
AS CittaArrivo, a2.nomeAeroporto AS AerArr, dataPartenza, SUM(numeroBiglietti) AS
BigliettiVenduti,
```

```
FROM volo JOIN prenotazione JOIN aeroporto a1 JOIN aeroporto a2 ON
idVolo=idVoloPren AND aeroportoPartenza=a1.codiceAeroporto AND
aeroportoArrivo=a2.codiceAeroporto
```

GROUP BY idVolo

HAVING SUM(numeroBiglietti)&gt;=ALL(SELECT SUM(numeroBiglietti)

```
FROM volo JOIN prenotazione ON idVolo=idVoloPren
GROUP BY idVolo)
```



queryF.png

## 5.2 Procedure e funzioni

### Procedure

**a) Aggiunta della carta al momento della registrazione/inserimento di un nuovo cliente.**

DROP PROCEDURE IF EXISTS aggiungiCarta;

DELIMITER !

CREATE PROCEDURE aggiungiCarta(IN cod varchar(10), IN utente varchar(20))

BEGIN

INSERT INTO carta VALUES(cod,utente,0);

END!

DELIMITER ;

**b) Aggiunta record in tabella viaggio.**

DROP PROCEDURE IF EXISTS aggiungiViaggia;

DELIMITER !

CREATE PROCEDURE aggiungiViaggia(IN codAereo varchar(10), IN codAeroporto varchar(10))

BEGIN

```
INSERT INTO viaggia VALUES(codAereo,codAeroporto);  
END!  
DELIMITER ;
```

## **Funzioni**

### **a) Funzione che calcola il numero di voli in programma con costo minore di x.**

```
DROP FUNCTION IF EXISTS fasciaVoli;  
DELIMITER !  
CREATE FUNCTION fasciaVoli(x double) RETURNS int  
BEGIN  
    DECLARE numVoli int;  
    SELECT COUNT(*) INTO numVoli  
    FROM volo  
    WHERE costo<x;  
    RETURN numVoli;  
END!  
DELIMITER ;
```

### **b) Funzione che restituisce il complessivo speso dal cliente x tra tutte le sue prenotazioni.**

```
DROP FUNCTION IF EXISTS importoTotaleCliente;  
DELIMITER !  
CREATE FUNCTION importoTotaleCliente (cliente varchar(40)) RETURNS INT  
BEGIN  
    DECLARE tot int;  
    SELECT SUM(costoBiglietto*numeroBiglietti) INTO tot  
    FROM prenotazione  
    WHERE mailUtente=cliente;  
    RETURN tot;  
END!  
DELIMITER ;
```

### **c) Funzione che ritorna il numero di clienti registrati al sito.**

```
DROP FUNCTION IF EXISTS numClienti;  
DELIMITER !  
CREATE FUNCTION numClienti() RETURNS int
```

```

BEGIN
    DECLARE numClienti int;
    SELECT COUNT(*) INTO numClienti
    FROM utente
    WHERE admin='0';
    RETURN numClienti;
END!
DELIMITER ;

```

**d) Funzione che indica se il cliente x ha mai fatto una prenotazione.**

```

DROP FUNCTION IF EXISTS booked;
DELIMITER !
CREATE FUNCTION booked(x varchar(40)) RETURNS BOOLEAN
BEGIN
    DECLARE pren BOOLEAN;
    SET pren=false;
    IF(x IN (SELECT mailUtente FROM prenotazione))
    THEN SET pren=true;
    END IF;
    RETURN pren;
END!
DELIMITER ;

```

### 5.3 Trigger

**a) Controllare se ci sono posti disponibili nell'aereo per un dato volo, in base alla richiesta del cliente (in caso negativo, il trigger genererà un errore in quanto cercherebbe di inserire una chiave nulla nella tabella prenotazione).**

```

DROP TRIGGER IF EXISTS check_posti;
DELIMITER !
CREATE TRIGGER check_posti BEFORE INSERT ON prenotazione FOR EACH ROW
BEGIN
    DECLARE postidisp int;
    SELECT numeroPosti-SUM(numeroBiglietti) INTO postidisp
    FROM aereo JOIN volo JOIN prenotazione ON idAereo=idAereoVolo AND
    idVolo=idVoloPren
    WHERE idVolo=new.idVoloPren

```

```
GROUP BY idVolo;
IF new.numeroBiglietti>postidisp THEN INSERT INTO prenotazione VALUES
(null,null,null,null,null,null,null);
END IF;
END!
DELIMITER ;
```

**b) Aggiunta/rimozione dei punti sulla carta di un cliente dopo una prenotazione (eventualmente con sconto)**

```
DROP TRIGGER IF EXISTS add_points;
DELIMITER !
CREATE TRIGGER add_points AFTER INSERT ON prenotazione FOR EACH ROW
BEGIN
    DECLARE postisel int;
    DECLARE mail_u varchar(255);
    DECLARE price double;
    DECLARE price_volo double;
    DECLARE class varchar(255);
    DECLARE sales int;
    DECLARE checksale int;
    SET mail_u=new.mailUtente;
    SET price=new.costoBiglietto;
    SET class=new.classe;
    SET postisel=new.numeroBiglietti;
    SELECT costo INTO price_volo
    FROM prenotazione JOIN volo ON new.idVoloPren=idVolo
    WHERE idPrenotazione=new.idPrenotazione;
    IF class='prima' THEN SET sales=50;
    ELSEIF class='business' THEN SET sales=20;
    ELSEIF class='economy' THEN SET sales=0;
    END IF;
    SET checksale=price_volo-(price-sales);
    UPDATE carta
    SET punti=punti+10*postisel-10*checksale
    WHERE carta.mailUtente=mail_u;
END!
DELIMITER ;
```

## 6 PHP

L'interfaccia web è stata costruita con i linguaggi HTML e PHP, che permette di connettersi al DBMS PHPMyAdmin, tramite l'estensione MySQLi.

Sono stati utilizzati elementi di HTML 5 per facilitare l'inserimento e i controlli sulla consistenza dei dati inseriti lato client.

Inoltre, si è fatto uso di JavaScript per il form di login/registrazione e per il controllo sull'inserimento dei punti per lo sconto sui biglietti relativi ad una prenotazione da parte di un cliente.

Di seguito, viene presentata la lista di tutti i file con estensione *.php*, con relativa descrizione:

- **utility.php:** contiene tutte le funzioni ausiliarie che sono necessarie alle altre pagine (connessione al database, chiusura della connessione al database, stampa dei meta-tag, stampa della sezione header della pagina, stampa della sezione footer della pagina, ecc...).
- **index.php:** pagina per l'accesso all'interfaccia web attraverso form di login con sessioni di autenticazione.
- **login.php:** script per controllare i dati di login inseriti dall'utente, se l'utente è registrato (in caso di successo, l'utente viene reindirizzato alla home, altrimenti viene segnalato un errore su index.php), altrimenti consente di inserire i dati per la registrazione e la creazione di un nuovo account.
- **insertNewUser.php:** controlla i dati inseriti dall'utente nel form di registrazione, in modo da creare un account o segnalare un errore per mail o telefono già registrati.
- **home.php:** pagina contenente il menù principale, con le voci "*Profilo*", "*Storico Prenotazioni*", "*Cerca Voli*", "*Gestione SkyDreamer*". Nel caso si tratti di un cliente, la voce "*Gestione SkyDreamer*" non è presente, poiché riservata ai soli amministratori; viceversa, "*Storico Prenotazioni*" è presente solo per i clienti, poiché gli amministratori non possono effettuare prenotazioni.
- **logout.php:** consente di effettuare l'uscita dal sito, chiudendo la sessione di autenticazione (nel caso si sia deciso di cancellare il proprio account, si procede alla rimozione del record relativo all'utente dalla tabella utente).
- **bookings.php:** visualizza lo storico delle prenotazioni fatte dal cliente.
- **profile.php:** visualizza le informazioni personali associate all'account con cui ci si è autenticati. Nel caso di un cliente, vengono visualizzate anche le informazioni relative alla propria carta fedeltà per la raccolta di punti.

- **changePsw.php:** pagina per la procedura del cambio password dell'account del cliente connesso (viene richiesta la password corrente e la nuova password ripetuta due volte).
- **checkNewPsw.php:** controlla che l'utente abbia inserito la corretta password corrente, che la nuova password sia diversa da quella corrente, e che la nuova password sia ripetuta correttamente per due volte.
- **deleteUser.php:** pagina che richiede la conferma per la cancellazione dell'account cliente.
- **search.php:** pagina dove è possibile specificare partenza, arrivo, data di partenza e numero di passeggeri, in modo da cercare il volo che si desidera.
- **searchResult.php:** visualizza tutti i voli che rispettano i parametri specificati dall'utente nella pagina search.php. Da qui un volo può essere scelto per essere prenotato semplicemente cliccando su "*Prenota*", la quale richiederà una classe di volo e il numero di passeggeri, prima di eseguire la transazione.
- **adminMenu.php:** pagina che mostra le possibili operazioni per l'utente amministratore.
- **manage.php:** reindirizzamento alla corretta pagina di gestione del database dopo aver selezionato una tabella (inserimento nuovo record o visualizzazione contenuto tabella).
- **insert.php:** pagina per l'inserimento dei dati di un nuovo record della tabella specificata in adminMenu.php.
- **execInsert.php:** operazione di inserimento di un nuovo record all'interno della tabella.
- **view.php:** pagina che visualizza l'intero contenuto della tabella selezionata in adminMenu.php. Ogni record della tabella visualizzata è selezionabile per essere modificato o cancellato.
- **delUpd.php:** recupero delle informazioni del record selezionato in view.php , con reindirizzamento alla pagina di riepilogo informazioni recordSelected.php.
- **recordSelected.php:** pagina che mostra i valori sugli attributi del record selezionato in precedenza, e in caso si sia scelta l'operazione di modifica, consente di modificare tutti e soli i campi che non sono chiave primaria della tabella.
- **execDelUpd.php:** esegue l'operazione specificata, modificando o cancellando il record selezionato.
- **getDataQuery.php:** pagina per l'inserimento dei dati necessari all'esecuzione della query selezionata dall'amministratore.



- **execQuery.php:** esegue la query selezionata con i dati inseriti in getDataQuery.php e ne mostra il risultato.
- **sceltaPren.php:** pagina che chiede all'utente le informazioni riguardanti il numero di biglietti e la classe di volo.
- **infoPren.php:** pagina che riassume i dati essenziali del volo che il cliente sta andando a prenotare. Da qui, inoltre, il cliente seleziona il numero di punti che vuole utilizzare per ottenere uno sconto sui biglietti prenotati.
- **prenota.php:** esegue l'operazione di prenotazione, se questa è possibile con i dati inseriti dal cliente.

## 7 Note

In caso di problemi nell'accesso all'URL specificato nell'intestazione di questo documento, il progetto è disponibile anche al seguente indirizzo: <http://basidati.studenti.math.unipd.it/basidati/~gpegorar/>.

Il sito implementa le query descritte al paragrafo 5.1 (vedi pagina *adminMenu.php*) e fa uso dei trigger descritti nel paragrafo 5.3, i quali sono salvati in PHPMyAdmin; inoltre, utilizza le procedure descritte al paragrafo 5.2, anch'esse salvate in PHPMyAdmin e richiamate nelle pagine php.

Il sito permette ad un utente amministratore operazioni di inserimento e modifica di record: non vi è un controllo profondo sulla consistenza dei dati inseriti dall'amministratore, il quale deve quindi avere cura nell'inserire informazioni valide.

Al fine di popolare il database per ottenere dei risultati significativi con le query descritte, sono stati creati altri profili di utenti di tipo cliente (tutti con password 123456).

E' necessario che JavaScript sia abilitato sul browser su cui si sta utilizzando il sito, per una sua corretta esecuzione.

Il database è stato popolato con 180 voli differenti, ma dato l'elevato numero di aeroporti disponibili è possibile che la ricerca di un volo produca un insieme vuoto di risultati.

Il sito è stato testato nei pc dei laboratori del dipartimento sul browser Google Chrome; pertanto i fogli di stile CSS sono settati per una corretta visualizzazione sugli schermi di tali macchine.