



CORSO DI BASI DI DATI – PROF. VINCENZO MOSCATO

---

## Progetto WaWaCar

---

Marco Bocchetti

N46 003218

Marco Barletta

N46 003441

Davide Biancardi

N46 003263

Stefano Aramu

N46 003392

Università degli Studi di Napoli Federico II  
Facoltà di Ingegneria Informatica  
Anno Accademico 2017 – 2018

# Indice

<b>1</b>	<b>Introduzione</b>	<b>2</b>
<b>2</b>	<b>Specifiche</b>	<b>3</b>
2.1	Specifiche sui dati . . . . .	3
2.2	Specifiche sulle operazioni . . . . .	3
2.3	Specifiche sugli utenti della base di dati . . . . .	4
<b>3</b>	<b>Progettazione concettuale</b>	<b>5</b>
3.1	Schema E/R portante . . . . .	5
3.2	Schema E/R completo . . . . .	5
3.3	Schema E/R trasformato . . . . .	7
<b>4</b>	<b>Progettazione logica</b>	<b>8</b>
4.1	Dallo schema E/R al modello logico-relazionale . . . . .	8
<b>5</b>	<b>Progettazione fisica</b>	<b>9</b>
5.1	Premesse . . . . .	9
5.2	Dimensionamento tabelle e tablespace . . . . .	10
5.3	Creazione dell'utente DBA, delle tabelle e delle sequenze . . . . .	13
5.4	Aggiunta delle chiavi esterne e politiche di reazione . . . . .	17
5.5	Stored procedures e <i>job</i> . . . . .	18
5.6	Triggers e compound triggers . . . . .	21
5.7	Definizione delle viste e raccolta delle statistiche . . . . .	24
5.8	Definizione delle politiche di sicurezza . . . . .	27
5.9	Definizione degli indici . . . . .	28
5.10	Gestione della concorrenza . . . . .	29
5.11	Controllo dell'affidabilità . . . . .	29
5.12	Schema finale . . . . .	30
<b>6</b>	<b>Progettazione dell'applicazione</b>	<b>31</b>
6.1	Introduzione . . . . .	31
6.2	Comunicazione tra client e database . . . . .	32
6.3	Accesso come utente passeggero o autista . . . . .	32
6.4	Ricerca dei viaggi disponibili . . . . .	33
6.5	Creazione di un viaggio . . . . .	34
6.6	Utilizzo delle viste . . . . .	35
6.7	Altre operazioni sulle base di dati . . . . .	35
<b>7</b>	<b>Livello presentazione</b>	<b>37</b>
7.1	Dal livello applicazione al livello presentazione . . . . .	37
7.2	Visualizzazione del voto medio . . . . .	37
7.3	Esempi di pagine web . . . . .	38

# 1 Introduzione

Un'azienda vuole realizzare una piattaforma informatica che consenta il car pooling tra viaggiatori sul territorio nazionale, con l'obiettivo di diffondere l'uso di una mobilità flessibile e personalizzata in termini di percorsi e costi. Si richiede che la piattaforma consenta di effettuare le seguenti operazioni:

- la registrazione di utenti
- l'inserimento di viaggi
- la consultazione e la prenotazione dei viaggi
- l'inserimento di commenti e voti per autisti e passeggeri
- l'invio di email di accettazione/rifiuto per le prenotazioni

Per l'implementazione del database è stata utilizzata la distribuzione Oracle 11g XE e un PC<sup>1</sup> dotato di un processore Intel i5 2x2.8 GHz, 8 GB di memoria RAM, e un HDD da 500 GB.

Il team di sviluppo ha proceduto nel seguente ordine:

1. progettazione e implementazione del database
2. creazione del server e del layer applicativo in JavaScript
3. creazione del layer di presentazione tramite HTML/CSS e JavaScript

La stesura della corrente documentazione è stata svolta in  $\text{\LaTeX}$ .

---

<sup>1</sup>Durante la fase di testing del servizio, il PC ha svolto contemporaneamente il ruolo di server e di client, con applicazione sul localhost.

## **2 Specifiche**

### **2.1 Specifiche sui dati**

Il committente richiede che si gestiscano le seguenti informazioni:

- per i passeggeri: username, password, generalità (nome, cognome, città, via, civico), codice del documento di identità, recapito telefonico, email, fotografia, voti, commenti
- per gli autisti: username, password, generalità (nome, cognome, città, via, civico), numero e scadenza patente di guida, dati dell'automobile utilizzata (una), recapito telefonico, email, fotografia, voti, commenti
- per ciascun viaggio: città di partenza e città di destinazione, data ed ora di partenza, contributo economico richiesto ad ogni passeggero, tempi di percorrenza stimati ed il numero di posti disponibili
- per ciascuna città: cap, nome

In seguito ad analisi da parte della dirigenza della compagnia, ci si aspetta di gestire nel primo anno di esercizio del servizio:

- l'anagrafica di circa 10.000 utenti (2000 autisti e 8000 passeggeri)
- 8000 comuni
- circa 100.000 viaggi e 500.000 prenotazioni

Al fine di evitare un rallentamento progressivo delle prestazioni dovuto all'incremento dei dati, si è concordato con il committente di utilizzare un archivio storico per la gestione delle informazioni relative ai viaggi conclusi e alle loro prenotazioni.

### **2.2 Specifiche sulle operazioni**

In seguito alla registrazione degli utenti e all'inserimento delle informazioni relative ai comuni da parte dell'azienda committente, si richiede per la piattaforma:

- che sia responsabilità dell'autista accettare le prenotazioni dei passeggeri
- che la piattaforma mostri solo i viaggi con posti ancora disponibili
- che una prenotazione non ancora accettata dall'autista non comporti alcun impegno del posto, che resta così ancora disponibile per prenotazioni di altri passeggeri
- che sia possibile visualizzare per ciascun viaggio il numero dei posti disponibili e il numero delle prenotazioni non ancora accettate
- che la piattaforma fornisca ai passeggeri la possibilità di indicare città di partenza e città di destinazione e data desiderata, presentando quindi un elenco dei soli viaggi disponibili
- che ogni passeggero possa visualizzare e inserire un giudizio relativamente ad un autista con cui ha viaggiato
- che ogni autista possa visualizzare e inserire un giudizio relativamente ad un passeggero con cui ha viaggiato

## **2.3 Specifiche sugli utenti della base di dati**

Per la tipologia del servizio richiesto, si è optato per la creazione di due utenti della base di dati:

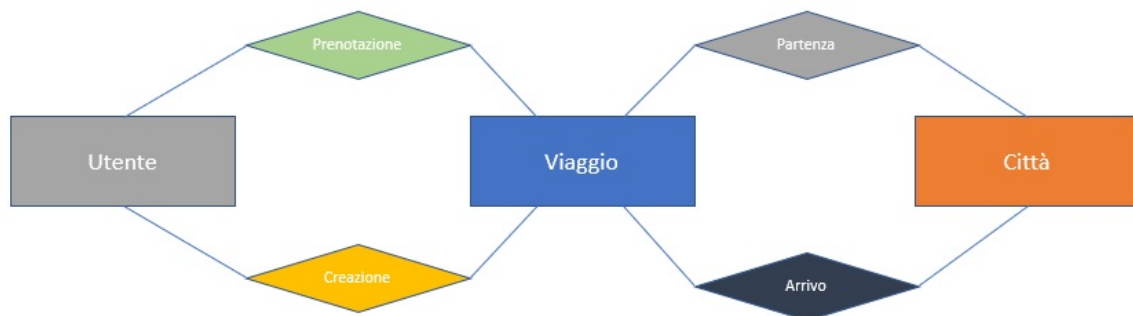
- un utente DBA, che si occupa tra le altre cose di registrare gli utenti e di inserire le informazioni relative ai comuni
- un utente per l'applicazione web, avente i soli privilegi strettamente necessari allo svolgimento delle sue operazioni sulla base di dati

### 3 Progettazione concettuale

#### 3.1 Schema E/R portante

In questa fase si è dapprima definito un modello E/R portante che cogliesse gli aspetti fondamentali del problema, e successivamente si è operato per raffinamenti successivi ed estensioni, inserendo quindi gli attributi delle entità nel dettaglio.

Nel modello E/R portante sono state individuate tre entità chiave: utente, viaggio, città. Alla luce di queste semplici considerazioni iniziali uno schema portante è il seguente:



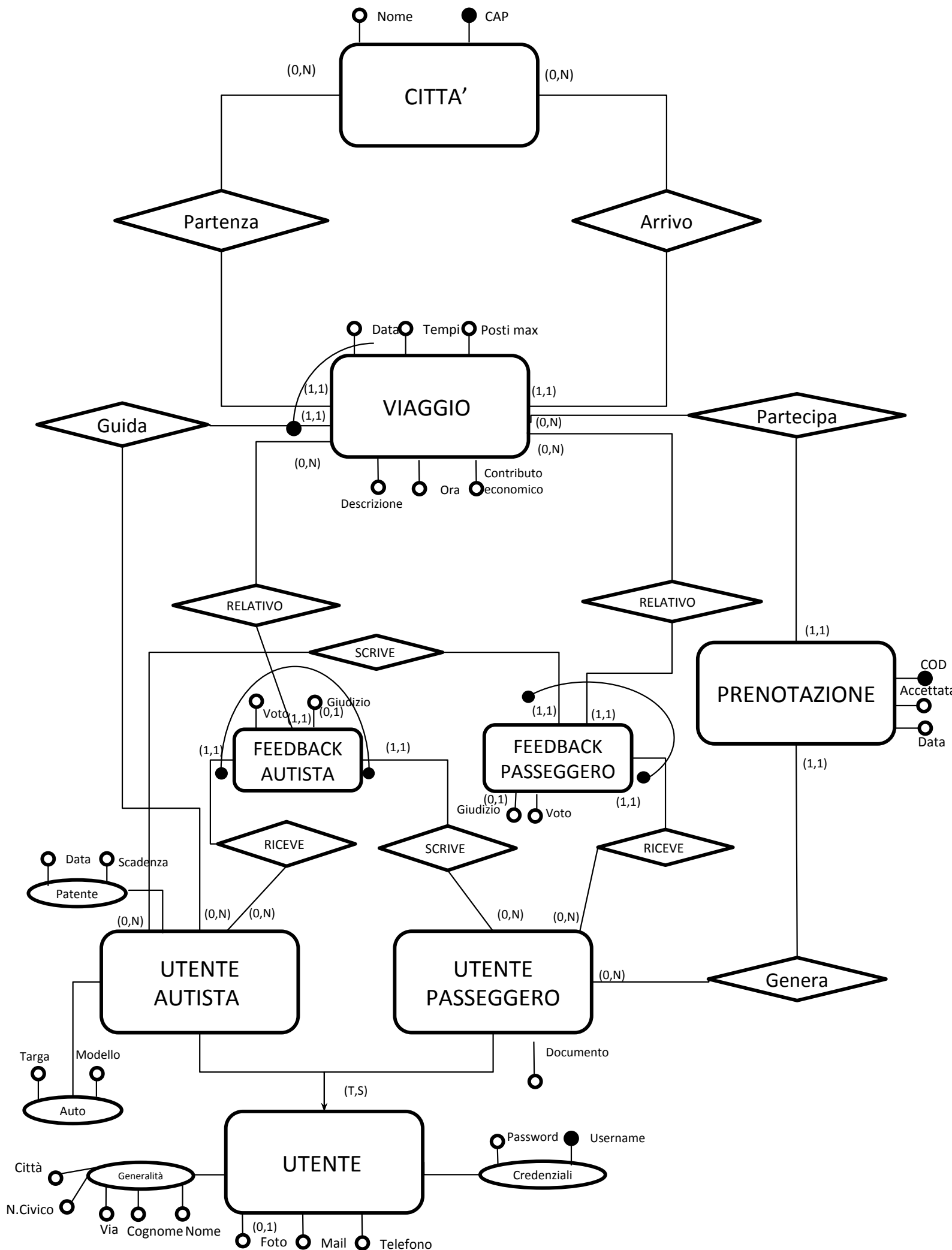
Infatti un utente può prenotarsi ad un viaggio o può crearlo, e ogni viaggio è legato ad una città per la partenza e ad un'altra città per l'arrivo. Una volta individuato il fulcro del modello E/R possiamo procedere ad una analisi più dettagliata.

#### 3.2 Schema E/R completo

Riesaminando nel dettaglio le specifiche è immediato accorgersi del fatto che ci sono utenti autisti, ovvero quegli utenti che intendono condividere un viaggio con altri, e poi ci sono utenti passeggeri, ovvero coloro che necessitano di un passaggio per andare da una città ad un'altra. È evidente, allora, che l'entità autista e l'entità passeggero sono due specializzazioni di una più generale entità utente. Tale gerarchia risulta essere totale e sovrapposta (un utente è un autista o un passeggero, ed eventualmente può essere entrambe le cose). In secondo luogo è evidente che una prenotazione presenti delle proprietà significative e goda di indipendenza, e deve essere quindi a tutti gli effetti considerata una entità separata. In particolare una prenotazione può essere effettuata da un solo utente (passeggero) ed è relativa ad un unico viaggio (se un utente vuole prenotarsi per più viaggi farà più prenotazioni, una per ciascun viaggio). Ogni passeggero può effettuare più prenotazioni, o potrebbe non effettuarne affatto (si iscrive e per vario tempo non prenota).

L'associazione Creazione riguarda solo gli autisti. È evidente che la creazione di un viaggio può essere effettuata da un unico autista, mentre un autista può creare più viaggi o può non crearne affatto. Per quanto riguarda le città è semplice osservare che ognuna può essere luogo di partenza o destinazione di molti viaggi, oppure potrebbe non essere toccata da alcun viaggio. Al contrario ogni viaggio deve avere necessariamente una e una sola città di partenza ed una e una sola città di destinazione (se l'autista intende fare più tappe, dovrà considerare ciascuna come viaggio a sé stante).

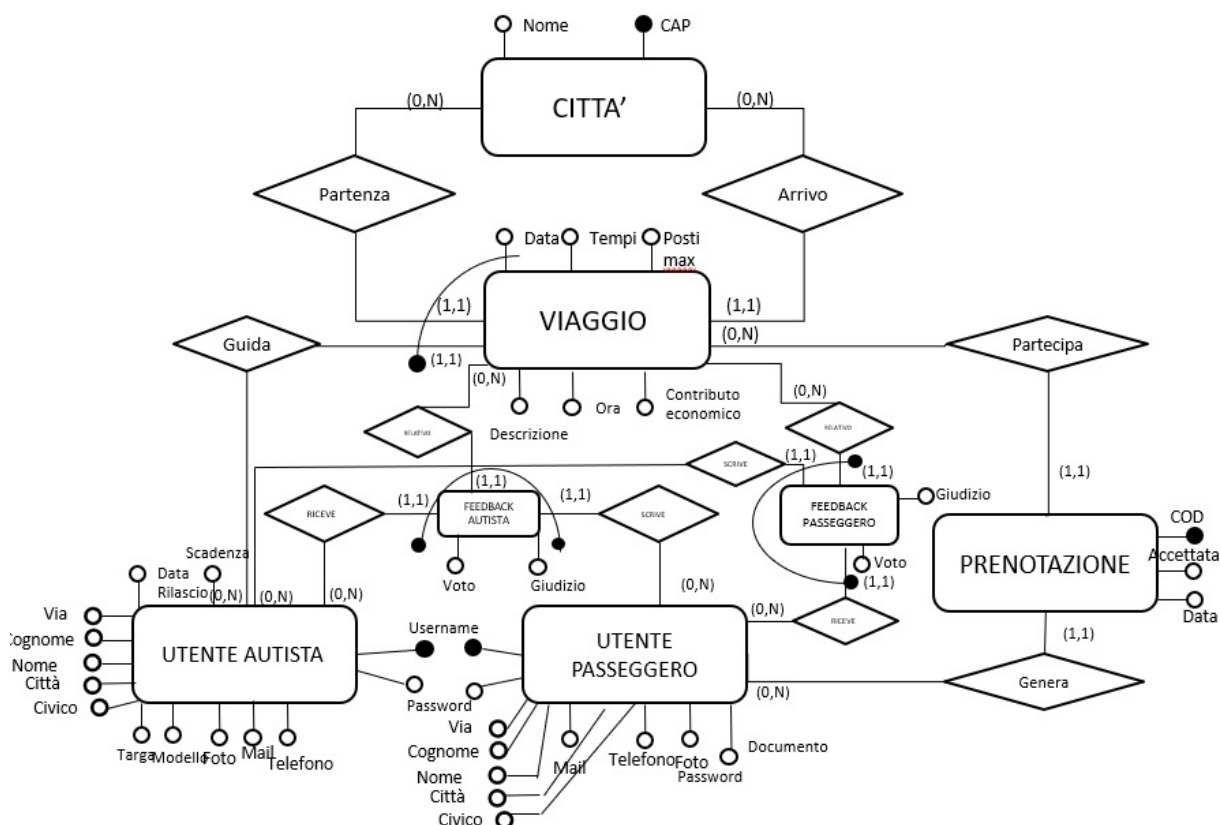
Bisogna inoltre tener presente che ogni passeggero può lasciare giudizi e commenti su un autista con cui ha viaggiato e viceversa. A tale scopo si creano due entità: feedback autisti e feedback passeggeri. Ogni feedback è relativo ad uno e un solo passeggero, ad uno e un solo autista e ad uno e un solo viaggio. Al contrario un autista e un passeggero possono lasciare più feedback (uno per utente) e ogni viaggio può essere legato a più feedback. I feedback non sono obbligatori (cardinalità minima 0).



È da notare che il viaggio, non avendo un proprio codice identificativo, deve essere identificato dalla combinazione di un autista, della città di partenza e di una data (in questo modo l'autista può decidere di fare più viaggi nello stesso giorno). Analogamente i feedback devono essere identificati dalla combinazione di un autista, di un passeggero e di un viaggio.

### 3.3 Schema E/R trasformato

A questo punto non resta che trasformare il modello E/R per poterlo successivamente tradurre nel modello logico-relazionale. Dal momento che le operazioni specificate su autisti e passeggeri sono diverse, che le associazioni sulle entità figlie sono diverse e che l'entità padre non ha associazioni proprie, si è scelto di accorpare la classe padre nelle figlie, facendo ereditare loro tutti gli attributi. Sono stati poi scomposti gli attributi composti (generalità, credenziali, auto, patente) e collegati alle relative entità. Si è deciso di considerare telefono non multivalore (un solo recapito telefonico a persona) allo scopo di velocizzare, da una parte, le interrogazioni sulla base di dati (un attributo multivalore avrebbe comportato la necessità di fare join) e, dall'altra, di semplificare l'utilizzo della piattaforma per gli utenti (più numeri di telefono confonderebbero un utente che vuole rintracciarne un altro). Il risultato è il seguente schema E/R trasformato:





## 4 Progettazione logica

### 4.1 Dallo schema E/R al modello logico-relazionale

Giunti a questo punto sono state seguite le regole di traduzione per costruire il modello logico-relazionale. Sono state create le relazioni (con i nomi al plurale delle entità), aventi come chiave primaria i relativi identificatori. Per facilitare le operazioni sulla base di dati si introduce un codice univoco per l'identificazione dei viaggi e dei feedback.

Per quanto riguarda la traduzione delle associazioni si è proceduto come segue:

- PARTENZA e ARRIVO sono state tradotte aggiungendo a VIAGGI una chiave esterna per identificare il luogo di partenza e una chiave esterna per identificare il luogo di arrivo, dal momento che le due associazioni sono uno a molti con partecipazione obbligatoria dal lato uno.
- GUIDA è stata tradotta aggiungendo a VIAGGI una chiave esterna che identifica l'autista creatore del viaggio.
- PARTECIPA è stata tradotta aggiungendo a PRENOTAZIONI una chiave esterna che referencia VIAGGI, poiché l'associazione è uno a molti con partecipazione obbligatoria dal lato uno.
- GENERA è stata tradotta aggiungendo a PRENOTAZIONI una chiave esterna che referencia PASSEGGERI.
- Le tre associazioni che riguardano i feedback (sia per i feedback passeggeri che per i feedback autisti) sono state tradotte aggiungendo a FEEDBACK tre chiavi che referenziano rispettivamente VIAGGI, AUTISTI e PASSEGGERI, poiché le associazioni sono tutte e tre uno a molti con partecipazione obbligatoria dal lato uno. Inoltre a feedback sono stati aggiunti dei campi CODICE affinché si possano mantenere i feedback in memoria qualora venga eliminato dalla base di dati un viaggio o l'autore del feedback (in tal caso si imposta Viaggio o Autista/Passeggero a NULL).

Il risultato di quanto detto è il seguente modello logico-relazionale:

- **Viaggi** (Codice, Data, Partenza: Citta, Arrivo: Citta, Autista: Autisti, Durata, PostiMax, Descrizione, Contributo, Disponibile)
- **Citta** (CAP, Nome)
- **Prenotazioni** (Codice, Accettato, Data, CodiceViaggio: Viaggi, Passeggero: Passeggeri)
- **Passeggeri** (Username, Password, Citta, Via, Civico, Nome, Cognome, Foto, E-Mail\*, Telefono\*, Documento\*)
- **Autisti** (Username, Password, Citta, Via, Civico, Nome, Cognome, Foto, E-Mail\*, Telefono\*, NumeroPatente\*, ScadenzaPatente, TargaAuto\*, ModelloAuto)
- **FeedbackPasseggeri** (Codice, Voto, Commento, Passeggero: Passeggeri, Autista: Autisti, CodiceViaggio: Viaggi)
- **FeedbackAutisti** (Codice, Voto, Commento, Passeggero: Passeggeri, Autista: Autisti, CodiceViaggio: Viaggi)

## 5 Progettazione fisica

### 5.1 Premesse

Dopo aver creato il modello logico-relazionale si è passati all'implementazione della base di dati sulla distribuzione Oracle 11g XE. Si è, dunque:

- Dimensionato le tabelle ed i tablespaces
- Creato l'utente DBA della base di dati
- Creato le tabelle e in seguito aggiunti i relativi vincoli
- Creato i trigger, le procedure e il *job* per l'archiviazione automatica dei viaggi conclusi
- Creato le viste sulle interrogazioni frequenti
- Creato il ruolo di *webapp* e l'utente per l'applicazione web
- Definito gli indici sulle tabelle
- Definito le politiche per la gestione della concorrenza e dell'affidabilità

Si è già accennato al fatto che si fa uso di due tabelle aggiuntive per la gestione dei viaggi conclusi e delle relative prenotazioni. Nello storico dei viaggi non si tiene più memoria della durata, della descrizione, dei posti max. e della disponibilità, in quanto sono attributi rilevanti ai soli viaggi futuri. Lo storico delle prenotazioni, invece, è necessario per mantenere traccia di chi ha effettivamente partecipato a tali viaggi conclusi. È ovvio, allora, che si terrà memoria solo delle prenotazioni accettate. L'introduzione dello storico dei viaggi ha reso, inoltre, necessario modificare le chiavi esterne dei feedback, poiché questi ultimi sono evidentemente relativi ai soli viaggi conclusi.

Successivamente sono state aggiunte due ulteriori tabelle, *Registrazioni\_Autisti* e *Registrazioni\_Passeggeri*, per mantenere le richieste di registrazione da parte degli utenti (ad esempio da applicazione web). Da qui il DBA stesso potrà decidere di accettare le registrazioni (spostando gli utenti, rispettivamente, nelle tabelle *Autisti* e *Passeggeri* tramite le stored procedure apposite), eventualmente mandando una email di conferma all'utente.

## 5.2 Dimensionamento tabelle e tablespace

CITTA'		
CAP	CHAR (5)	5 BYTE
NOME	VARCHAR2 (50)	50 BYTE

TOT: 55 BYTE \* 8000 = 440 KB + 10% = 500 KB

VIAGGI		
CODICE	NUMBER (6)	5 BYTE
PARTENZA	CHAR (5)	5 BYTE
ARRIVO	CHAR (5)	5 BYTE
DATA	DATE	7 BYTE
AUTISTA	VARCHAR2 (20)	20 BYTE
POSTI MAX	NUMBER (2)	3 BYTE
DURATA	CHAR(5)	5 BYTE
CONTRIBUTO ECONOMICO	NUMBER (3)	4 BYTE
DESCRIZIONE	CLOB	4 BYTE
DISPONIBILE	CHAR (1)	1 BYTE

TOT: 59 BYTE \* 100.000 = 5,9 MB + 25% = 7,5 MB

PRENOTAZIONI		
CODICE	NUMBER (6)	5 BYTE
CODICE VIAGGIO	NUMBER (6)	5 BYTE
PASSEGGERO	VARCHAR2 (20)	20 BYTE
DATA	DATE	7 BYTE
ACCETTATO	CHAR (1)	1 BYTE

TOT: 38 BYTE \* 500.000 = 19 MB + 20% = 23 MB

AUTISTI		
USERNAME	VARCHAR2 (20)	20 BYTE
PASSWORD	VARCHAR2 (20)	20 BYTE
EMAIL	VARCHAR2 (50)	50 BYTE
NOME	VARCHAR2 (100)	100 BYTE
COGNOME	VARCHAR2 (100)	100 BYTE
CITTA'	VARCHAR2 (50)	50 BYTE
VIA	VARCHAR2 (100)	100 BYTE
CIVICO	NUMBER (5)	5 BYTE
TELEFONO	CHAR (10)	10 BYTE
NUMERO PATENTE	CHAR(10)	10 BYTE
SCADENZA PATENTE	DATE	7 BYTE
FOTO	BLOB	4 BYTE
TARGA AUTO	CHAR (7)	7 BYTE
MODELLO AUTO	VARCHAR (80)	80 BYTE

TOT: 563 BYTE \* 2000 = 1,12 MB + 15% = 1,30 MB

PASSEGGERI		
USERNAME	VARCHAR2 (20)	20 BYTE
PASSWORD	VARCHAR2 (20)	20 BYTE
EMAIL	VARCHAR2 (50)	50 BYTE
NOME	VARCHAR2 (100)	100 BYTE
COGNOME	VARCHAR2 (100)	100 BYTE
CITTA'	VARCHAR2 (50)	50 BYTE
VIA	VARCHAR2 (100)	100 BYTE
CIVICO	NUMBER (5)	5 BYTE
TELEFONO	CHAR (10)	10 BYTE
DOCUMENTO	CHAR (9)	9 BYTE
FOTO	BLOB	4 BYTE

TOT: 468 BYTE \* 8000 = 3,74 MB + 15% = 4,30 MB

FEEDBACK AUTISTI		
CODICE	NUMBER (6)	5 BYTE
AUTISTA	VARCHAR2 (20)	20 BYTE
PASSEGGERO	VARCHAR2 (20)	20 BYTE
CODICE VIAGGIO	NUMBER (6)	5 BYTE
VOTO	NUMBER (1)	3 BYTE
COMMENTO	CLOB	4 BYTE

TOT : 57 BYTE \* 500.000 = 28,5 MB + 10% = 31 MB

FEEDBACK PASSEGGERI		
CODICE	NUMBER (6)	5 BYTE
AUTISTA	VARCHAR2 (20)	20 BYTE
PASSEGGERO	VARCHAR2 (20)	20 BYTE
CODICE VIAGGIO	NUMBER (6)	5 BYTE
VOTO	NUMBER (1)	3 BYTE
COMMENTO	CLOB	4 BYTE

TOT : 57 BYTE \* 500.000 = 28,5 MB + 10% = 31 MB

ARCHIVIO VIAGGI		
CODICE	NUMBER (6)	5 BYTE
PARTENZA	CHAR (5)	5 BYTE
ARRIVO	CHAR (5)	5 BYTE
DATA	DATE	7 BYTE
AUTISTA	VARCHAR2 (20)	20 BYTE
CONTRIBUTO ECONOMICO	NUMBER (3)	4 BYTE

TOT: 46 BYTE \* 100.000 = 4,6 MB + 25% = 6 MB

ARCHIVIO PRENOTAZIONI		
CODICE	NUMBER (6)	5 BYTE
CODICE VIAGGIO	NUMBER (6)	5 BYTE
PASSEGGERO	VARCHAR2 (20)	20 BYTE

TOT: 30 BYTE \* 500.000 = 15 MB + 20% = 18 MB

REGISTRAZIONE AUTISTI		
EMAIL	VARCHAR2 (50)	50 BYTE
NOME	VARCHAR2 (100)	100 BYTE
COGNOME	VARCHAR2 (100)	100 BYTE
CITTA'	VARCHAR2 (50)	50 BYTE
VIA	VARCHAR2 (100)	100 BYTE
CIVICO	NUMBER (5)	5 BYTE
TELEFONO	CHAR (10)	10 BYTE
NUMERO PATENTE	CHAR (10)	10 BYTE
SCADENZA PATENTE	DATE	7 BYTE
FOTO	BLOB	4 BYTE
TARGA AUTO	CHAR (7)	7 BYTE
MODELLO AUTO	VARCHAR2 (80)	80 BYTE

TOT: 523 BYTE \* 2000 = 1 MB + 5% = 1,05 MB

REGISTRAZIONE PASSEGGERI		
EMAIL	VARCHAR2 (50)	50 BYTE
NOME	VARCHAR2 (100)	100 BYTE
COGNOME	VARCHAR2 (100)	100 BYTE
CITTA'	VARCHAR2 (50)	50 BYTE
VIA	VARCHAR2 (100)	100 BYTE
CIVICO	NUMBER (5)	5 BYTE
TELEFONO	CHAR (10)	10 BYTE
DOCUMENTO	CHAR (9)	9 BYTE
FOTO	BLOB	4 BYTE

TOT: 428 BYTE \* 8000 = 3,42 MB + 10% = 3,76 MB

Sono stati, poi, dimensionati i tablespace:

- **wawa\_ts** 300 MB, che contiene la maggior parte delle tabelle (la somma degli storage iniziali delle tabelle è di 129 MB)
- **wawa\_foto\_ts** 250 MB, la cui dimensione teorica sarebbe 10 GB (considerando 1 MB a foto), ma si è limitati dalla licenza di Oracle in uso
- **wawa\_feedback\_ts** 250 MB, che contiene i commenti dei feedback (considerando 250 caratteri a commento \* 1.000.000 commenti)
- **wawa\_descrizioni\_ts** 25 MB, che contiene le descrizioni dei viaggi, ovvero tutti quei dettagli che non rientrano in uno specifico attributo (possibilità di portare il bagaglio e animali, eventuali soste, ecc.) (250 caratteri \* 10.000)

```
1 CREATE TABLESPACE wawa_ts DATAFILE 'D:\oracle\app\oracle\oradata\XE\wawa.dbf' SIZE
   300 M;
3 CREATE TABLESPACE wawa_foto_ts DATAFILE 'D:\oracle\app\oracle\oradata\XE\wawa_foto
   .dbf' SIZE 250 M;
5 CREATE TABLESPACE wawa_feedback_ts DATAFILE 'D:\oracle\app\oracle\oradata\XE\
   wawa_feedback.dbf' SIZE 250 M;
7 CREATE TABLESPACE wawa_descrizioni_ts DATAFILE 'D:\oracle\app\oracle\oradata\XE\
   wawa_descrizioni.dbf' SIZE 25 M;
```

### 5.3 Creazione dell'utente DBA, delle tabelle e delle sequenze

Le operazioni sulla base di dati di competenza del *Database Administrator* vengono effettuate tramite l'utente **wawa\_dba**, avente gli appositi privilegi.

```
1 CREATE USER wawa_dba DEFAULT TABLESPACE wawa_ts IDENTIFIED BY 1234;
GRANT DBA, UNLIMITED TABLESPACE TO wawa_dba;
```

Tramite tale utente sono state create, poi, le tabelle precedentemente definite e le sequenze per l'inserimento dei valori delle chiavi primarie:

```
CREATE SEQUENCE contatoreViaggi;
2 CREATE SEQUENCE contatorePrenotazioni;
CREATE SEQUENCE contatoreFeedbackAutisti;
4 CREATE SEQUENCE contatoreFeedbackPasseggeri;

6
CREATE TABLE Citta(
8   cap char(5),
   nome varchar2(50) NOT NULL,
10
   CONSTRAINT PK_CITTA PRIMARY KEY (Cap)
12 )
STORAGE (INITIAL 500 K);
14

16 CREATE TABLE Viaggi(
   codice number(6),
18   partenza char(5) NOT NULL,
```

```

    arrivo char(5) NOT NULL,
20    data date NOT NULL,
    autista varchar2(20) NOT NULL,
22    postiMax number(2) NOT NULL,
    durata char(5) NOT NULL,
24    contributo number(3) NOT NULL,
    descrizione clob,
26    disponibile char(1) DEFAULT 'T',

28    CONSTRAINT PK_VIAGGI PRIMARY KEY (codice),
    CONSTRAINT CC_CONTRIBUTO_VIAGGI CHECK (contributo > 0),
30    CONSTRAINT CC_DURATA_VIAGGI CHECK (durata LIKE '____'),
    CONSTRAINT CC_DISPONIBILE_VIAGGI CHECK (disponibile = 'T' OR disponibile = 'F')
32 )
LOB (descrizione) STORE AS lob_descrizione (TABLESPACE wawa_descrizioni_ts)
34 STORAGE (INITIAL 8 M);

36
CREATE TABLE ArchivioViaggi(
38    codice number(6),
    partenza char(5) NOT NULL,
40    arrivo char(5) NOT NULL,
    data date NOT NULL,
42    autista varchar2(20) NOT NULL,
    contributo number(3) NOT NULL,
44
    CONSTRAINT PK_ARCHIVIOVIAGGI PRIMARY KEY (codice)
46 )
STORAGE (INITIAL 6 M);
48

50 CREATE TABLE Prenotazioni(
    codice number(6),
52    codiceViaggio number(6) NOT NULL,
    passeggero varchar2(20) NOT NULL,
54    data date NOT NULL,
    accettato char(1) DEFAULT 'F',
56
    CONSTRAINT PK_PRENOTAZIONI PRIMARY KEY (codice),
58    CONSTRAINT UC_PRENOTAZIONI UNIQUE (codiceViaggio, passeggero),
    CONSTRAINT CC_ACCETTATO_PRENOTAZIONI CHECK (accettato = 'T' OR accettato = 'F')
60 )
STORAGE (INITIAL 23 M);
62

64 CREATE TABLE ArchivioPrenotazioni(
    codice number(6),
66    codiceViaggio number(6) NOT NULL,
    passeggero varchar2(20) NOT NULL,
68
    CONSTRAINT PK_ARCHIVIOPRENOTAZIONI PRIMARY KEY (codice)
70 )
STORAGE (INITIAL 18 M);
72

74 CREATE TABLE Autisti(
    username varchar2(20),
76    password varchar2(20) NOT NULL,
    email varchar2(50) NOT NULL,
78    nome varchar2(100) NOT NULL,
    cognome varchar2(100) NOT NULL,
80    citta varchar2(50) NOT NULL,
    via varchar2(100) NOT NULL,
82    civico number(5) NOT NULL,
    telefono char(10) NOT NULL,
84    numeroPatente char(10) NOT NULL,

```

```

scadenzaPatente date NOT NULL,
86 foto blob,
targaAuto char(7) NOT NULL,
88 modelloAuto varchar2(80) NOT NULL,

90 CONSTRAINT PK_AUTISTI PRIMARY KEY (username),
CONSTRAINT UC_EMAIL_AUTISTI UNIQUE (email),
92 CONSTRAINT UC_TELEFONO_AUTISTI UNIQUE (telefono),
CONSTRAINT UC_NUMEROPATENTE_AUTISTI UNIQUE (numeroPatente),
94 CONSTRAINT UC_TARGA_AUTISTI UNIQUE (targaAuto)
)
96 LOB (foto) STORE AS lob_foto_autisti (TABLESPACE wawa_foto_ts)
STORAGE (INITIAL 2 M);
98

100 CREATE TABLE Passeggeri(
    username varchar2(20),
102 password varchar2(20) NOT NULL,
    email varchar2(50) NOT NULL,
104 nome varchar2(100) NOT NULL,
    cognome varchar2(100) NOT NULL,
106 citta varchar2(50) NOT NULL,
    via varchar2(100) NOT NULL,
108 civico number(5) NOT NULL,
    telefono char(10) NOT NULL,
110 documento char(9) NOT NULL,
    foto blob,
112
    CONSTRAINT PK_PASSEGGGERI PRIMARY KEY (username),
114 CONSTRAINT UC_EMAIL_PASSEGGGERI UNIQUE (email),
CONSTRAINT UC_TELEFONO_PASSEGGGERI UNIQUE (telefono),
116 CONSTRAINT UC_DOCUMENTO_PASSEGGGERI UNIQUE (documento)
)
118 LOB (foto) STORE AS lob_foto_passeggeri (TABLESPACE wawa_foto_ts)
STORAGE (INITIAL 5 M);
120

CREATE TABLE Feedback_Autisti(
122     codice number(6),
    autista varchar2(20) NOT NULL,
124 passeggero varchar2(20),
    codiceViaggio number(6),
126 voto number(1),
    commento clob,
128
    CONSTRAINT PK_FEEDBACK_AUTISTI PRIMARY KEY (codice),
130 CONSTRAINT UC_FEEDBACK_AUTISTI UNIQUE (passeggero, codiceViaggio),
CONSTRAINT CC_VOTO_FEEDBACK_AUTISTI CHECK (voto >= 0 AND voto <= 5)
132 )
LOB (commento) STORE AS lob_commento_autisti (TABLESPACE wawa_feedback_ts)
134 STORAGE (INITIAL 31 M);

136

CREATE TABLE Feedback_Passeggeri(
138     codice number(6),
    passeggero varchar2(20) NOT NULL,
140 autista varchar2(20),
    codiceViaggio number(6),
142 voto number(1),
    commento clob,
144
    CONSTRAINT PK_FEEDBACK_PASSEGGGERI PRIMARY KEY (codice),
146 CONSTRAINT UC_FEEDBACK_PASSEGGGERI UNIQUE (autista, passeggero, codiceViaggio),
CONSTRAINT CC_VOTO_FEEDBACK_PASSEGGGERI CHECK (voto >= 0 AND voto <= 5)
148 )
LOB (commento) STORE AS lob_commento_passeggeri (TABLESPACE wawa_feedback_ts)
150 STORAGE (INITIAL 31 M);

```



Sono state, infine, create le due tabelle per le registrazioni da applicazione web, rispettivamente per gli autisti e per i passeggeri. È stata considerata come chiave primaria il campo email, coerentemente al fatto che il campo email nelle tabelle Autisti e Passeggeri è vincolato da una UNIQUE CONSTRAINT:

```
2 CREATE TABLE Registrazioni_Autisti(  
    email varchar2(50),  
4     nome varchar2(100) NOT NULL,  
    cognome varchar2(100) NOT NULL,  
6     citta varchar2(50) NOT NULL,  
    via varchar2(100) NOT NULL,  
8     civico number(5) NOT NULL,  
    telefono char(10) NOT NULL,  
10    numeroPatente char(10) NOT NULL,  
    scadenzaPatente date NOT NULL,  
12    foto blob,  
    targaAuto char(7) NOT NULL,  
14    modelloAuto varchar2(80) NOT NULL,  
  
16    CONSTRAINT PK_REGIST_AUTISTI PRIMARY KEY (email),  
    CONSTRAINT UC_TELEFONO_REGIST_AUTISTI UNIQUE (telefono),  
18    CONSTRAINT UC_NUMEROPAT_REGIST_AUTISTI UNIQUE (numeroPatente),  
    CONSTRAINT UC_TARGA_REGIST_AUTISTI UNIQUE (targaAuto)  
20 )  
LOB (foto) STORE AS lob_foto_regist_autisti (TABLESPACE wawa_foto_ts)  
22 STORAGE (INITIAL 1 M);  
  
24  
CREATE TABLE Registrazioni_Passeggeri(  
26    email varchar2(50),  
    nome varchar2(100) NOT NULL,  
28    cognome varchar2(100) NOT NULL,  
    citta varchar2(50) NOT NULL,  
30    via varchar2(100) NOT NULL,  
    civico number(5) NOT NULL,  
32    telefono char(10) NOT NULL,  
    documento char(9) NOT NULL,  
34    foto blob,  
  
36    CONSTRAINT PK_REGIST_PASSEGGERI PRIMARY KEY (email),  
    CONSTRAINT UC_TELEFONO_REGIST_PASSEGGERI UNIQUE (telefono),  
38    CONSTRAINT UC_DOCUMENTO_REGIST_PASSEGGERI UNIQUE (documento)  
    )  
40 LOB (foto) STORE AS lob_foto_regist_passeggeri (TABLESPACE wawa_foto_ts)  
    STORAGE (INITIAL 4 M);
```

## 5.4 Aggiunta delle chiavi esterne e politiche di reazione

Le scelte per le politiche di reazione sono state le seguenti:

- Quando una città viene eliminata vengono eliminati tutti i viaggi ad essa relativi
- Quando un autista viene eliminato si perde memoria anche dei suoi viaggi
- Quando un viaggio viene eliminato vengono eliminate anche le relative prenotazioni
- Quando viene eliminato un passeggero vengono eliminate anche le sue prenotazioni
- Se viene eliminato un utente oggetto di un feedback il feedback viene eliminato, mentre se viene eliminato il viaggio relativo al feedback o l'autore del feedback il relativo campo viene portato a NULL per conservare ugualmente il giudizio, dal momento che esso ha comunque un'utilità statistica indipendentemente da chi l'ha scritto in passato (si è fatto in modo che un utente non possa rilasciare un feedback su un altro utente con cui non ha condiviso un viaggio)

```
1 ALTER TABLE Viaggi ADD CONSTRAINT FK_PARTENZA_VIAGGI FOREIGN KEY (partenza)
  REFERENCES Citta(cap)
  ON DELETE CASCADE;
3
4 ALTER TABLE Viaggi ADD CONSTRAINT FK_ARRIVO_VIAGGI FOREIGN KEY (arrivo) REFERENCES
  Citta(cap)
5 ON DELETE CASCADE;
6
7 ALTER TABLE Viaggi ADD CONSTRAINT FK_AUTISTA_VIAGGI FOREIGN KEY (autista) REFERENCES
  Autisti(username)
8 ON DELETE CASCADE;
9
10 ALTER TABLE ArchivioViaggi ADD CONSTRAINT FK_PARTENZA_ARCHIVIOVIAGGI FOREIGN KEY (
  partenza) REFERENCES Citta(cap)
11 ON DELETE CASCADE;
12
13 ALTER TABLE ArchivioViaggi ADD CONSTRAINT FK_ARRIVO_ARCHIVIOVIAGGI FOREIGN KEY (
  arrivo) REFERENCES Citta(cap)
14 ON DELETE CASCADE;
15
16 ALTER TABLE ArchivioViaggi ADD CONSTRAINT FK_AUTISTA_ARCHIVIOVIAGGI FOREIGN KEY (
  autista) REFERENCES Autisti(username)
17 ON DELETE CASCADE;
18
19 ALTER TABLE Prenotazioni ADD CONSTRAINT FK_CODICEVIAGGIO_PRENOTAZIONI FOREIGN KEY (
  codiceViaggio) REFERENCES Viaggi(codice)
20 ON DELETE CASCADE;
21
22 ALTER TABLE Prenotazioni ADD CONSTRAINT FK_PASSEGGERO_PRENOTAZIONI FOREIGN KEY (
  passeggero) REFERENCES Passeggeri(username)
23 ON DELETE CASCADE;
24
25 ALTER TABLE ArchivioPrenotazioni ADD CONSTRAINT FK_CODICEVIAGGIO_ARCHIVIOIPREN
  FOREIGN KEY (codiceViaggio) REFERENCES ArchivioViaggi(codice)
26 ON DELETE CASCADE;
27
28 ALTER TABLE ArchivioPrenotazioni ADD CONSTRAINT FK_PASSEGGERO_ARCHIVIOIPREN FOREIGN
  KEY (passeggero) REFERENCES Passeggeri(username)
29 ON DELETE CASCADE;
30
31 ALTER TABLE Feedback_Autisti ADD CONSTRAINT FK_AUTISTA_FB_AUT FOREIGN KEY (autista)
  REFERENCES Autisti(username)
32 ON DELETE CASCADE;
33
```

```

ALTER TABLE Feedback_Autisti ADD CONSTRAINT FK_PASSEGGERO_FB_AUT FOREIGN KEY (
    passeggero) REFERENCES Passeggeri(username)
35 ON DELETE SET NULL;

37 ALTER TABLE Feedback_Autisti ADD CONSTRAINT FK_CODICEVIAGGIO_FB_AUT FOREIGN KEY (
    codiceViaggio) REFERENCES ArchivioViaggi(codice)
ON DELETE SET NULL;
39

ALTER TABLE Feedback_Passeggeri ADD CONSTRAINT FK_PASSEGGERO_FB_PASS FOREIGN KEY (
    passeggero) REFERENCES Passeggeri(username)
41 ON DELETE CASCADE;

43 ALTER TABLE Feedback_Passeggeri ADD CONSTRAINT FK_AUTISTA_FB_PASS FOREIGN KEY (
    autista) REFERENCES Autisti(username)
ON DELETE SET NULL;
45

ALTER TABLE Feedback_Passeggeri ADD CONSTRAINT FK_CODICEVIAGGIO_FB_PASS FOREIGN KEY
    (codiceViaggio) REFERENCES ArchivioViaggi(codice)
47 ON DELETE SET NULL;

```

## 5.5 Stored procedures e *job*

Definite le tabelle sono state implementati trigger, procedure e *job* al fine di rendere la base di dati "attiva" e per velocizzare alcune operazioni. Si nota esplicitamente che il COMMIT delle operazioni è lasciato al chiamante delle procedure, il quale potrebbe voler compiere altre operazioni all'interno della stessa transazione.

In primo luogo sono state definite le stored procedures per l'inserimento di feedback e il lancio dell'eccezione nel caso si tenti di inserire un giudizio per una combinazione di autista/-passeggero/viaggio non valida:

```

1  -- Inserimento di un feedback relativo ad un autista
CREATE OR REPLACE PROCEDURE insert_feedback_autista
3  (cod IN Feedback_Autisti.codiceViaggio%TYPE,
    driver IN Feedback_Autisti.autista%TYPE,
5  pass IN Feedback_Autisti.passeggero%TYPE,
    voto IN Feedback_Autisti.voto%TYPE,
7  commento IN Feedback_Autisti.commento%TYPE) AS
BEGIN
9  DECLARE
    codicepren ArchivioPrenotazioni.codice%TYPE;
11   codiceviaggio ArchivioViaggi.codice%TYPE;
BEGIN
13   BEGIN
        SELECT codice INTO codiceviaggio FROM ArchivioViaggi WHERE autista = driver
15         AND codice = cod;
        EXCEPTION WHEN no_data_found THEN raise_application_error (-20001, 'autista
17         non appartenente al viaggio');
    END;

19   BEGIN
21       SELECT codice INTO codicepren FROM ArchivioPrenotazioni WHERE passeggero =
        pass AND codiceViaggio = cod;
23       EXCEPTION WHEN no_data_found THEN raise_application_error (-20001,
        'passeggero non ha effettuato quel viaggio');
25   END;

27   INSERT INTO Feedback_Autisti VALUES (contatoreFeedbackAutisti.NEXTVAL, driver,
        pass, cod, voto, commento);
29 END;
END insert_feedback_autista;

```

```

31 -- Inserimento di un feedback relativo ad un passeggero
33 CREATE OR REPLACE PROCEDURE insert_feedback_passeggero
    (cod IN Feedback_Passeggeri.codiceViaggio%TYPE,
35     driver IN Feedback_Passeggeri.autista%TYPE,
    pass IN Feedback_Passeggeri.passeggero%TYPE,
37     voto IN Feedback_Passeggeri.voto%TYPE,
    commento IN Feedback_Passeggeri.commento%TYPE) AS
39 BEGIN
    DECLARE
41     codicepren ArchivioPrenotazioni.codice%TYPE;
    codiceviaggio ArchivioViaggi.codice%TYPE;
43 BEGIN
    BEGIN
45     SELECT codice INTO codiceviaggio FROM ArchivioViaggi WHERE autista = driver
    AND codice = cod;
    EXCEPTION WHEN no_data_found THEN raise_application_error (-20001, 'autista
47     non appartenente al viaggio');
    END;
49
    BEGIN
51     SELECT codice INTO codicepren FROM ArchivioPrenotazioni WHERE passeggero =
    pass AND codiceViaggio = cod;
53     EXCEPTION WHEN no_data_found THEN raise_application_error (-20002,
    'passeggero non ha effettuato quel viaggio');
55     END;
57
    INSERT INTO Feedback_Passeggeri VALUES (contatoreFeedbackPasseggeri.NEXTVAL,
59     pass, driver, cod, voto, commento);
    END insert_feedback_passeggero;

```

Dopoiché è stata implementata la stored procedure per la prenotazione ad un viaggio:

```

-- Inserimento di una prenotazione
2 CREATE OR REPLACE PROCEDURE prenota_viaggio (cod IN Prenotazioni.codiceViaggio%TYPE,
    pass IN Prenotazioni.passeggero%TYPE) AS
    BEGIN
4     /* Non e' necessario fare alcuna gestione di eccezione: se il codiceViaggio o il
    passeggero non esistono, l'errore viene individuato dalla violazione della
    FOREIGN KEY CONSTRAINT. I duplicati vengono rilevati dalla violazione della
    UNIQUE KEY CONSTRAINT. */
    INSERT INTO Prenotazioni VALUES (contatorePrenotazioni.NEXTVAL, cod, pass,
6     SYSDATE, 'F');
    END prenota_viaggio;

```

Per quanto riguarda la registrazione degli utenti, sono state implementate due procedure per lo spostamento degli utenti dalle tabelle delle registrazioni alle tabelle Autisti e Passeggeri. Entrambe le procedure prendono in ingresso un username e password (da assegnare all'utente) e l'email dell'utente che si intende spostare. In questo modo si potrebbe decidere in futuro di implementare un generatore automatico di username e password, nonché di mandare una notifica via email all'utente.

```

1 -- Sposta un utente autista dalla tabella registrazioni alla tabella autisti
CREATE OR REPLACE PROCEDURE registra_autista (user IN Autisti.username%TYPE,
3     pass IN Autisti.password%TYPE, mail IN Autisti.email%TYPE) AS
    BEGIN
5     DECLARE
    autista Registrazioni_Autisti%ROWTYPE;
7     BEGIN

```

```

SELECT * INTO autista FROM Registrazioni_Autisti WHERE email = mail;
9  INSERT INTO Autisti VALUES (user, pass, mail, autista.nome, autista.cognome,
    autista.citta, autista.via,
11     autista.civico, autista.telefono, autista.numeroPatente,
    autista.scadenzaPatente, autista.foto, autista.targaAuto,
13     autista.modelloAuto);
    DELETE FROM Registrazioni_Autisti WHERE email = mail;
15 END;
END registra_autista;
17
-- Sposta un utente passeggero dalla tabella registrazioni alla tabella passeggeri
19 CREATE OR REPLACE PROCEDURE registra_passeggero (user IN Passeggeri.username%TYPE,
    pass IN Passeggeri.password%TYPE, mail IN Passeggeri.email%TYPE) AS
21 BEGIN
    DECLARE
23     passeggero Registrazioni_Passeggeri%ROWTYPE;
    BEGIN
25     SELECT * INTO passeggero FROM Registrazioni_Passeggeri WHERE email = mail;
        INSERT INTO Passeggeri VALUES (user, pass, mail, passeggero.nome, passeggero
        .cognome, passeggero.citta, passeggero.via,
27     passeggero.civico, passeggero.telefono, passeggero.documento, passeggero
        .foto);
        END;
29 END registra_passeggero;

```

Successivamente sono state definite le stored procedures per l'archiviazione dei viaggi e delle prenotazioni:

```

1  -- Sposta le prenotazioni di viaggi conclusi nell archivio delle prenotazioni
CREATE OR REPLACE PROCEDURE archivia_pren (cod IN Viaggi.codice%TYPE) AS
3  BEGIN
    DECLARE
5      CURSOR pren IS SELECT codice, passeggero, accettato FROM Prenotazioni WHERE
        codiceViaggio = cod;
7      codicepr Prenotazioni.codice%TYPE;
        codpass Prenotazioni.passeggero%TYPE;
9      accettato Prenotazioni.accettato%TYPE;
    BEGIN
11     OPEN pren;
        LOOP
13         FETCH pren INTO codicepr, codpass, accettato;
            EXIT WHEN pren%NOTFOUND;
15         IF accettato = 'T' THEN
            INSERT INTO ArchivioPrenotazioni VALUES (codicepr, cod, codpass);
17         END IF;
            DELETE FROM Prenotazioni WHERE codice = codicepr;
19         END LOOP;
            CLOSE pren;
21     END;
END archivia_pren;
23
25 -- Sposta i viaggi conclusi nell archivio dei viaggi
CREATE OR REPLACE PROCEDURE archivia_viaggi AS
27 BEGIN
    DECLARE
29     cod Viaggi.codice%TYPE;
        part Viaggi.partenza%TYPE;
31     arr Viaggi.arrivo%TYPE;
        dat Viaggi.data%TYPE;
33     aut Viaggi.autista%TYPE;
        cont Viaggi.contributo%TYPE;
35     CURSOR viaggio IS SELECT codice, partenza, arrivo, data, autista, contributo
        FROM Viaggi WHERE data < SYSDATE;

```

```

37 BEGIN
    OPEN viaggio;
39 LOOP
    FETCH viaggio INTO cod, part, arr, dat, aut, cont;
41 EXIT WHEN viaggio%NOTFOUND;
    INSERT INTO ArchivioViaggi VALUES (cod, part, arr, dat, aut, cont);
43 archivia_pren (cod);
    DELETE FROM Viaggi WHERE codice = cod;
45 END LOOP;
    CLOSE viaggio;
47 END;
END archivia_viaggi;

```

Si pone di conseguenza il problema dello spostamento dei viaggi e delle prenotazioni nei loro rispettivi storici. Si nota esplicitamente che tale operazione non può essere svolta manualmente, in quanto, da una parte, non è possibile per un utente inserire feedback finché un viaggio non è stato storicizzato, e dall'altra, risulta inconcepibile che una persona si preoccupi di aggiornare le tabelle ogni *tot* minuti. Oracle mette a disposizione il **DBMS\_Scheduler**, che si occupa di eseguire in automatico un particolare compito (*job*) e di eseguire implicitamente il commit della sua transazione.<sup>2</sup> Possiamo allora implementare tale funzionalità nel seguente modo:

```

-- Archivia automaticamente i viaggi ogni 20 minuti
2 BEGIN
    DBMS_SCHEDULER.CREATE_JOB (
4         job_name          => 'JOB_ARCHIVIA_VIAGGI',
         job_type           => 'STORED_PROCEDURE',
6         job_action        => 'archivia_viaggi',
         start_date         => current_timestamp,
8         repeat_interval   => 'FREQ=MINUTELY; INTERVAL=20;', -- Esegui ogni 20 min
         enabled            => true);
10 END;

```

## 5.6 Triggers e compound triggers

Per quanto riguarda i triggers, ne è stato dapprima definito uno per impostare automaticamente alcune impostazioni sulla sessione (ciò può sembrare superfluo per il DBA, tuttavia si rende nella pratica necessario per il corretto funzionamento delle operazioni dall'applicazione web):

```

CREATE OR REPLACE TRIGGER on_database_logon AFTER LOGON ON DATABASE
2 BEGIN
    -- Si sceglie di utilizzare questo formato per le date
4     EXECUTE IMMEDIATE('ALTER SESSION SET NLS_DATE_FORMAT = "DD-MM-YYYY"');

6     -- Queste due istruzioni rendono il paragone fra stringhe case-insensitive
    EXECUTE IMMEDIATE('ALTER SESSION SET NLS_COMP = LINGUISTIC'); --
8     EXECUTE IMMEDIATE('ALTER SESSION SET NLS_SORT = BINARY_CI');
END;

```

Successivamente sono stati definiti due triggers indispensabili per la piattaforma: uno che riapre il viaggio quando una prenotazione viene eliminata, e un altro che lo chiude quando il numero di prenotazioni accettate è uguale al numero di posti massimi. Per questi trigger si pone il problema della *mutating table*, ovvero il gestore della concorrenza vieta di interrogare la tabella

<sup>2</sup>[https://blog.pythian.com/dbms\\_scheduler-and-implicit-commits/](https://blog.pythian.com/dbms_scheduler-and-implicit-commits/)

target al fine di prevenire eventuali letture sporche della tabella stessa. Per risolvere il problema si potrebbe utilizzare una tabella di appoggio cosiddetta *audit*, tuttavia si vuole mostrare un'ulteriore soluzione che fa uso dei **compound triggers** (implementati nella distribuzione 11g). Un compound trigger può "accendersi" in più istanti, comportandosi contemporaneamente come un trigger *statement-level* e come un trigger *row-level*. Oracle garantisce sempre la consistenza dei dati a livello statement,<sup>3</sup> e pertanto il problema della mutating table non si pone.

I compound trigger possono essere implementati come di seguito mostrato:

```

1  -- Compound trigger sull'eliminazione della prenotazione (risolve mutating table
    durante la CASCADE dei viaggi)
CREATE OR REPLACE TRIGGER on_pren_delete_compound FOR DELETE ON Prenotazioni
COMPOUND TRIGGER
3      idviaggio Prenotazioni.codiceViaggio%TYPE;
    accett Prenotazioni.accettato%TYPE;
5  AFTER EACH ROW IS
BEGIN
7      idviaggio := :old.codiceViaggio;
    accett := :old.accettato;
9  END AFTER EACH ROW;
  AFTER STATEMENT IS
11 BEGIN
    IF accett = 'T' THEN
13        UPDATE Viaggi SET disponibile = 'T' WHERE codice = idviaggio;
    END IF;
15 END AFTER STATEMENT;
END on_pren_delete_compound;
17

19 -- Compound trigger sull'accettazione della prenotazione (risolve mutating table)
CREATE OR REPLACE TRIGGER on_pren_update_compound FOR UPDATE OF accettato ON
Prenotazioni COMPOUND TRIGGER
21      idviaggio Prenotazioni.codiceViaggio%TYPE;
    cont Viaggi.postiMax%TYPE;
23      numposti Viaggi.postiMax%TYPE;
  AFTER EACH ROW IS
25 BEGIN
    idviaggio := :new.codiceViaggio;
27 END AFTER EACH ROW;
  AFTER STATEMENT IS
29 BEGIN
    SELECT COUNT(*) INTO cont FROM Prenotazioni WHERE codiceViaggio = idviaggio AND
    accettato = 'T';
31      SELECT postiMax INTO numposti FROM Viaggi WHERE codice = idviaggio;

33      IF cont = numposti THEN
        UPDATE Viaggi SET disponibile = 'F' WHERE codice = idviaggio;
35      ELSIF cont < numposti THEN
        UPDATE Viaggi SET disponibile = 'T' WHERE codice = idviaggio;
37      END IF;
  END AFTER STATEMENT;
39 END on_pren_update_compound;

```

Successivamente si è implementato un trigger che chiuda il viaggio nel caso in cui il numero di posti massimi venga ridotto al numero di prenotazioni accettate e lancia un'eccezione se il numero di posti massimi è minore del numero di prenotazioni accettate. Anche per questo trigger è necessario utilizzare il compound trigger.

```

1  -- Compound trigger sulla modifica di postiMax di viaggi (risolve mutating table)

```

<sup>3</sup>[https://docs.oracle.com/cd/B19306\\_01/server.102/b14220/consist.htm](https://docs.oracle.com/cd/B19306_01/server.102/b14220/consist.htm)

```

CREATE OR REPLACE TRIGGER on_viaggio_update_compound FOR UPDATE OF postiMax ON
  Viaggi COMPOUND TRIGGER
3   idviaggio Viaggi.codice%TYPE;
   newPosti Viaggi.postiMax%TYPE;
5   errore exception;
   num_pren number(4);
7 BEFORE EACH ROW IS
BEGIN
9   idviaggio := :new.codice;
   newPosti := :new.postiMax;
11 END BEFORE EACH ROW;
   BEFORE STATEMENT IS
13 BEGIN
   SELECT COUNT(*) INTO num_pren FROM Prenotazioni WHERE codiceViaggio = idviaggio
   AND accettato = 'T';
15
   IF num_pren > newPosti THEN
17     RAISE errore;
   ELSIF num_pren = newPosti THEN
19     UPDATE Viaggi SET disponibile = 'F' WHERE codice = idViaggio;
   ELSE
21     UPDATE Viaggi SET disponibile = 'T' WHERE codice = idViaggio;
   END IF;
23
   EXCEPTION WHEN errore THEN raise_application_error(-20003, 'il numero di
   postimax non puo essere minore delle prenotazioni accettate');
25 END BEFORE STATEMENT;
END on_viaggio_update_compound;

```

È stato successivamente implementato un trigger sull'inserimento di un viaggio, affinché il viaggio inserito abbia una data di partenza valida (è infatti impossibile inserire una CHECK CONSTRAINT che compia un paragone sulla SYSDATE):

```

-- Controllo sulla data per l'inserimento di un viaggio
2 CREATE OR REPLACE TRIGGER on_viaggio_insert BEFORE INSERT ON Viaggi FOR EACH ROW
DECLARE
4   toolate exception;
BEGIN
6   IF :new.data < sysdate
       THEN RAISE toolate;
8   END IF;

10  EXCEPTION WHEN toolate THEN raise_application_error(-20005, 'la data non puo
   essere minore di quella attuale');
END;

```

Come è ben noto, Oracle non implementa gli UPDATE a cascata. Si è allora ritenuto necessario implementare due trigger su Autisti e Passeggeri per garantire la consistenza dei dati tra le tabelle referenziate e referenzianti (le altre chiavi sono generate dalle sequenze e pertanto non si prevedono modifiche su di esse).

```

1 CREATE OR REPLACE TRIGGER on_autisti_update AFTER UPDATE ON Autisti FOR EACH ROW
BEGIN
3   UPDATE Viaggi SET autista = :new.username WHERE autista = :old.username;
   UPDATE ArchivioViaggi SET autista = :new.username WHERE autista = :old.username;
5   UPDATE Feedback_Autisti SET autista = :new.username WHERE autista = :old.
   username;
   UPDATE Feedback_Passeggeri SET autista = :new.username WHERE autista = :old.
   username;
7 END;

```



```

9 CREATE OR REPLACE TRIGGER on_passeggeri_update AFTER UPDATE ON Passeggeri FOR EACH
  ROW
  BEGIN
11     UPDATE Prenotazioni SET passeggero = :new.username WHERE passeggero = :old.
        username;
        UPDATE ArchivioPrenotazioni SET passeggero = :new.username WHERE passeggero = :
        old.username;
13     UPDATE Feedback_Passeggeri SET passeggero = :new.username WHERE passeggero = :
        old.username;
        UPDATE Feedback_Autisti SET passeggero = :new.username WHERE passeggero = :old.
        username;
15 END;

```

## 5.7 Definizione delle viste e raccolta delle statistiche

Sono state in seguito create delle viste (non materializzate) sulle interrogazioni frequenti sulla base di dati, sia per scopi statistici ma anche in vista dell'utilizzo da parte dell'applicazione web (si rimanda, a tal fine, alle sezioni sulla definizione delle politiche di sicurezza e sul layer applicativo). Molte query richiedono l'OUTER JOIN al fine di funzionare correttamente.

```

1 -- Ottieni il numero di prenotazioni non ancora accettate per ogni viaggio
  CREATE OR REPLACE VIEW Num_Pren_Sospese AS
3 SELECT V.codice AS codice_viaggio, COUNT(P.codice) AS num
  FROM Viaggi V LEFT OUTER JOIN Prenotazioni P ON (V.codice = P.codiceViaggio AND P.
        accettato = 'F') GROUP BY V.codice;
5
7 -- Ottieni il numero di prenotazioni accettate per ogni viaggio
  CREATE OR REPLACE VIEW Num_Pren_Accettate AS
9 SELECT V.codice AS codice_viaggio, COUNT(P.codice) AS num
  FROM Viaggi V LEFT OUTER JOIN Prenotazioni P ON (V.codice = P.codiceViaggio AND P.
        accettato = 'T') GROUP BY V.codice;
11
13 -- Ottieni il numero di posti ancora disponibili per ogni viaggio
  CREATE OR REPLACE VIEW Num_Posti_Disponibili AS
15 SELECT V.codice AS codice_viaggio, V.postiMax - N.num AS num
  FROM Viaggi V, Num_Pren_Accettate N WHERE V.codice = N.codice_viaggio;
17
19 -- Ottieni le medie dei voti per ciascun autista
  CREATE OR REPLACE VIEW Medie_Feedback_Autisti AS
21 SELECT F.autista, TRUNC(AVG(F.voto), 1) AS media FROM Feedback_Autisti F GROUP BY F.
        autista; -- TRUNC per approssimare le cifre decimali
23
-- Ottieni le medie dei voti per ciascun passeggero
25 CREATE OR REPLACE VIEW Medie_Feedback_Passeggeri AS
  SELECT F.passeggero, TRUNC(AVG(F.voto), 1) AS media FROM Feedback_Passeggeri F GROUP
        BY F.passeggero; -- TRUNC per approssimare le cifre decimali

```

Le seguenti viste vengono utilizzate per mostrare ad un utente tutte le informazioni relative ai viaggi, quelle relative alle prenotazioni, e infine l'elenco dei viaggi e delle prenotazioni.

```

-- Ottieni tutte le informazioni riguardanti tutti i viaggi
2 CREATE OR REPLACE VIEW Schede_Viaggi AS

```

```

SELECT
4     V.codice,
      V.partenza AS cap_partenza,
6     C1.nome AS partenza,
      V.arrivo AS cap_arrivo,
8     C2.nome AS arrivo,
      TO_CHAR(V.data) AS data,
10    V.autista AS username_autista,
      A.nome AS nome_autista,
12    A.cognome AS cognome_autista,
      A.telefono,
14    A.foto,
      M.media AS giudizio_medio,
16    A.modelloAuto AS modello_auto,
      D.num AS posti_disponibili,
18    S.num AS pren_sospese,
      V.durata,
20    V.contributo,
      V.descrizione
22 FROM
      Viaggi V
24    JOIN Citta C1 ON V.partenza = C1.cap
      JOIN Citta C2 ON V.arrivo = C2.cap
26    JOIN Num_Posti_Disponibili D ON V.codice = D.codice_viaggio
      JOIN Num_Pren_Sospese S ON V.codice = S.codice_viaggio
28    JOIN Autisti A ON V.autista = A.username
      LEFT OUTER JOIN Medie_Feedback_Autisti M ON V.autista = M.autista;
30

32 -- Ottieni le informazioni principali riguardanti tutti i viaggi
CREATE OR REPLACE VIEW Elenco_Viaggi AS
34 SELECT
      V.codice,
36     C1.nome AS partenza,
      C2.nome AS arrivo,
38     V.data AS data,
      A.nome,
40     A.cognome,
      M.media AS giudizio_medio,
42     V.contributo,
      V.disponibile
44 FROM
      Viaggi V
46    JOIN Citta C1 ON V.partenza = C1.cap
      JOIN Citta C2 ON V.arrivo = C2.cap
48    JOIN Autisti A ON V.autista = A.username
      LEFT OUTER JOIN Medie_Feedback_Autisti M ON A.username = M.autista;
50

52 -- Ottieni tutte le informazioni riguardanti tutte le prenotazioni
CREATE OR REPLACE VIEW Schede_Prenotazioni AS
54 SELECT
      PR.codice AS codice_prenotazione,
56     V.partenza AS cap_partenza,
      C1.nome AS partenza,
58     V.arrivo AS cap_arrivo,
      C2.nome AS arrivo,
60     TO_CHAR(V.data) AS data_viaggio,
      TO_CHAR(PR.data) AS data_prenotazione,
62     PR.passeggero AS username_passeggero,
      P.nome AS nome_passeggero,
64     P.cognome AS cognome_passeggero,
      P.telefono,
66     P.foto,
      M.media AS giudizio_medio
68 FROM

```

```

70     JOIN Citta C1 ON V.partenza = C1.cap
71     JOIN Citta C2 ON V.arrivo = C2.cap
72     JOIN Prenotazioni PR ON V.codice = PR.codiceViaggio
73     JOIN Passeggeri P ON PR.passeggero = P.username
74     LEFT OUTER JOIN Medie_Feedback_Passeggeri M ON V.autista = M.passeggero;
75
76 -- Ottieni le informazioni principali riguardanti tutte le prenotazioni
77
78 CREATE OR REPLACE VIEW Elenco_Prenotazioni AS
79 SELECT
80     PR.codice AS codice_prenotazione,
81     PR.codiceViaggio AS codice_viaggio,
82     V.autista AS username_autista,
83     C1.nome AS partenza,
84     C2.nome AS arrivo,
85     TO_CHAR(V.data) AS data_viaggio,
86     PR.passeggero AS username_passeggero,
87     P.nome AS nome_passeggero,
88     P.cognome AS cognome_passeggero,
89     M.media AS giudizio_medio,
90     PR.accettato
91 FROM
92     Viaggi V
93     JOIN Citta C1 ON V.partenza = C1.cap
94     JOIN Citta C2 ON V.arrivo = C2.cap
95     JOIN Prenotazioni PR ON V.codice = PR.codiceViaggio
96     JOIN Passeggeri P ON PR.passeggero = P.username
97     LEFT OUTER JOIN Medie_Feedback_Passeggeri M ON P.username = M.passeggero;

```

Le seguenti query calcolano il numero di partenze ed arrivi per ciascuna città (una per i soli viaggi conclusi, una per i soli viaggi futuri, e una per entrambi):

```

1 -- Ottieni il numero di arrivi conclusi per ogni città
2 CREATE OR REPLACE VIEW Num_Arrivi_Archivio AS
3 SELECT C.cap, COUNT(V.codice) AS num FROM Citta C LEFT JOIN ArchivioViaggi V ON (C.
4     cap = V.arrivo)
5 GROUP BY C.cap;
6
7 -- Ottieni il numero di arrivi futuri per ogni città
8 CREATE OR REPLACE VIEW Num_Arrivi_Futuri AS
9 SELECT C.cap, COUNT(V.codice) AS num FROM Citta C LEFT JOIN Viaggi V ON (C.cap = V.
10     arrivo)
11 GROUP BY C.cap;
12
13 -- Ottieni il numero totale di arrivi per ogni città
14 CREATE OR REPLACE VIEW Num_Arrivi_Totali AS
15 SELECT V1.cap, V1.num + V2.num AS num FROM Num_Arrivi_Archivio V1, Num_Arrivi_Futuri
16     V2 WHERE
17     V1.cap = V2.cap;
18
19 -- Ottieni il numero di partenze concluse per ogni città
20 CREATE OR REPLACE VIEW Num_Partenze_Archivio AS
21 SELECT C.cap, COUNT(V.codice) AS num FROM Citta C LEFT JOIN ArchivioViaggi V ON (C.
22     cap = V.partenza)
23 GROUP BY C.cap;
24
25 -- Ottieni il numero di partenze future per ogni città
26 CREATE OR REPLACE VIEW Num_Partenze_Futuri AS
27 SELECT C.cap, COUNT(V.codice) AS num FROM Citta C LEFT JOIN Viaggi V ON (C.cap = V.
28     partenza)
29 GROUP BY C.cap;
30
31 -- Ottieni il numero totale di partenze per ogni città
32 CREATE OR REPLACE VIEW Num_Partenze_Totali AS
33 SELECT V1.cap, V1.num + V2.num AS num FROM Num_Partenze_Archivio V1, Num_Partenze_Futuri
34     V2 WHERE
35     V1.cap = V2.cap;

```

```

27 CREATE OR REPLACE VIEW Num_Partenze_Totali AS
    SELECT V1.cap, V1.num + V2.num AS num FROM Num_Partenze_Archivio V1,
        Num_Partenze_Futuri V2 WHERE
29 V1.cap = V2.cap;

```

Mentre le seguenti query calcolano l'utente con la media migliore (una per gli autisti e una per i passeggeri):

```

1 -- Ottieni l'autista con la media migliore
  CREATE OR REPLACE VIEW Media_Migliore_Autisti AS
3 SELECT autista FROM Medie_Feedback_Autisti WHERE media IN (SELECT MAX(M.media)
  FROM Medie_Feedback_Autisti M);
5
  -- Ottieni il passeggero con la media migliore
7 CREATE OR REPLACE VIEW Media_Migliore_Passeggeri AS
  SELECT passeggero FROM Medie_Feedback_Passeggeri WHERE media IN (SELECT MAX(M.media)
9 FROM Medie_Feedback_Passeggeri M);

```

Infine sono state pensate alcune query per la raccolta di semplici dati statistici sull'utilizzo del servizio (in futuro si potrebbe pensare di implementare un vero e proprio *data warehouse*):

```

1 -- Ottieni il numero di autisti registrati
  CREATE OR REPLACE VIEW Num_Autisti AS
3 SELECT COUNT(*) AS num_autisti FROM Autisti;

5 -- Ottieni il numero di passeggeri registrati
  CREATE OR REPLACE VIEW Num_Passeggeri AS
7 SELECT COUNT(*) AS num_passeggeri FROM Passeggeri;

9 -- Ottieni il numero di partecipazioni ai viaggi conclusi
  CREATE OR REPLACE VIEW Num_Partecipazioni AS
11 SELECT COUNT(*) AS num_partecipazioni FROM ArchivioPrenotazioni;

13 -- Ottieni il numero di viaggi conclusi
  CREATE OR REPLACE VIEW Num_Viaggi_Fatti AS
15 SELECT COUNT(*) AS num_viaggi_fatti FROM ArchivioViaggi;

17 -- Ottieni la spesa media per i viaggi conclusi
  CREATE OR REPLACE VIEW Media_Spese_Archivio AS
19 SELECT TRUNC(AVG(contributo), 2) AS media_spesa FROM ArchivioViaggi;

21 -- Ottieni la spesa media per i viaggi futuri
  CREATE OR REPLACE VIEW Media_Spese_Futuri AS
23 SELECT TRUNC(AVG(contributo), 2) AS media_spesa FROM Viaggi;

25 -- Ottieni il numero di viaggi per passeggero
  CREATE OR REPLACE VIEW Viaggi_Per_Passeggero AS
27 SELECT passeggero, COUNT(*) AS num_viaggi FROM ArchivioPrenotazioni GROUP BY
    passeggero;

29 -- Ottieni, per ogni quantita' di viaggi, il numero di viaggiatori
  CREATE OR REPLACE VIEW Num_Viaggiatori_Per_Num_Viaggi AS
31 SELECT num_viaggi, COUNT(*) AS num_persone FROM Viaggi_Per_Passeggero GROUP BY
    num_viaggi;

```

## 5.8 Definizione delle politiche di sicurezza

Si pone ora la questione della gestione della sicurezza della base di dati, soprattutto in vista dell'applicazione web. Un primo approccio al problema era stato stato quello di definire l'utente

**wawa\_dba** affinché le operazioni non venissero eseguite direttamente con il ruolo di *system*. D'altra parte è ora evidente l'applicazione web non può e non deve possedere gli stessi privilegi dell'utente DBA.

La creazione del ruolo di *webapp* consente all'applicazione web di interfacciarsi in maniera sicura e controllata alla base di dati, concedendo ad essa di visualizzare e modificare solo le opportune tabelle. Al fine di aumentare la sicurezza (ma anche di aumentare il disaccoppiamento tra il layer applicativo e la struttura interna delle tabelle), sono state predisposte delle viste su alcune interrogazioni frequenti. In questo modo, l'applicazione web non accede direttamente alle tabelle, ma richiama semplicemente le viste su cui possiede gli opportuni privilegi (notiamo esplicitamente che i permessi possono essere concessi solo dopo aver creato le tabelle e le viste). Infine è stato definito l'utente **wawa\_webapp**, avente tali privilegi:

```
1 CREATE ROLE webapp;
  GRANT CONNECT TO webapp;
3
4 GRANT SELECT ON Autisti TO webapp;
5 GRANT SELECT ON Passeggeri TO webapp;
  GRANT SELECT ON Prenotazioni TO webapp;
7 GRANT SELECT ON Elenco_Viaggi TO webapp;          -- Vista
  GRANT SELECT ON Schede_Viaggi TO webapp;          -- Vista
9 GRANT SELECT ON Elenco_Prenotazioni TO webapp;     -- Vista
  GRANT SELECT ON Schede_Prenotazioni TO webapp;     -- Vista
11 GRANT SELECT ON contatoreViaggi TO webapp;
  GRANT SELECT ON contatorePrenotazioni TO webapp;
13
14 GRANT UPDATE ON Autisti TO webapp;
15 GRANT UPDATE ON Passeggeri TO webapp;
  GRANT INSERT, UPDATE, DELETE ON Prenotazioni TO webapp;
17 GRANT INSERT, UPDATE, DELETE ON Viaggi TO webapp;
  GRANT INSERT ON Registrazioni_Autisti TO webapp;
19 GRANT INSERT ON Registrazioni_Passeggeri TO webapp;
21
22 CREATE USER wawa_webapp DEFAULT TABLESPACE wawa_ts IDENTIFIED BY 1234;
  GRANT webapp TO wawa_webapp;
```

## 5.9 Definizione degli indici

Dal momento che Oracle crea di default indici definiti sulle chiavi primarie si è ritenuto necessario aggiungere indici secondari definiti sui campi di seguito elencati, che abbiamo selezionato in base alla frequenza di utilizzo delle query:

- **Citta.nome**, poiché la ricerca di un viaggio si basa soprattutto sui luoghi di partenza e di arrivo, i quali vengono generalmente cercati per nome e non per CAP
- **Feedback\_Autisti.autista**, poiché la frequente ricerca dei feedback relativi ad un autista sfrutta la ricerca sul campo autista dei feedback
- **Feedback\_Passeggeri.passeggero**, come sopra
- **Prenotazioni.codiceViaggio**, utile quando si ricercano le prenotazioni relative ad un certo viaggio

Riportiamo le istruzioni SQL necessarie alla creazione di tali indici.

```
1 CREATE INDEX INDICE_CITTA_NOME ON Citta (nome);
2 CREATE INDEX INDICE_FEEDBACKA_AUT ON Feedback_Autisti (autista);
  CREATE INDEX INDICE_FEEDBACKP_PASS ON Feedback_Passeggeri (passeggero);
```

## 5.10 Gestione della concorrenza

Per evitare anomalie di ogni tipo si è scelto come di consueto il protocollo **2PL stretto** per le transazioni in lettura/scrittura. Si è successivamente scelto il livello di isolamento **REPEATABLE READ** per le transazioni in lettura, che garantisce che se un dato è letto due volte, il risultato sarà sempre lo stesso. In questo modo si evitano letture inconsistenti, che seppur tollerate talvolta dai servizi di e-commerce, non si ritengono adatte a questo tipo di piattaforma. Si garantisce così ai clienti una migliore esperienza di utilizzo del servizio, dovendo tuttavia imporre all'utente di completare la sua transazione in un tempo limite (una transazione potrebbe altrimenti bloccare a lungo la schedule del DBMS). Si è scelto come tempo massimo 8 minuti.

## 5.11 Controllo dell'affidabilità

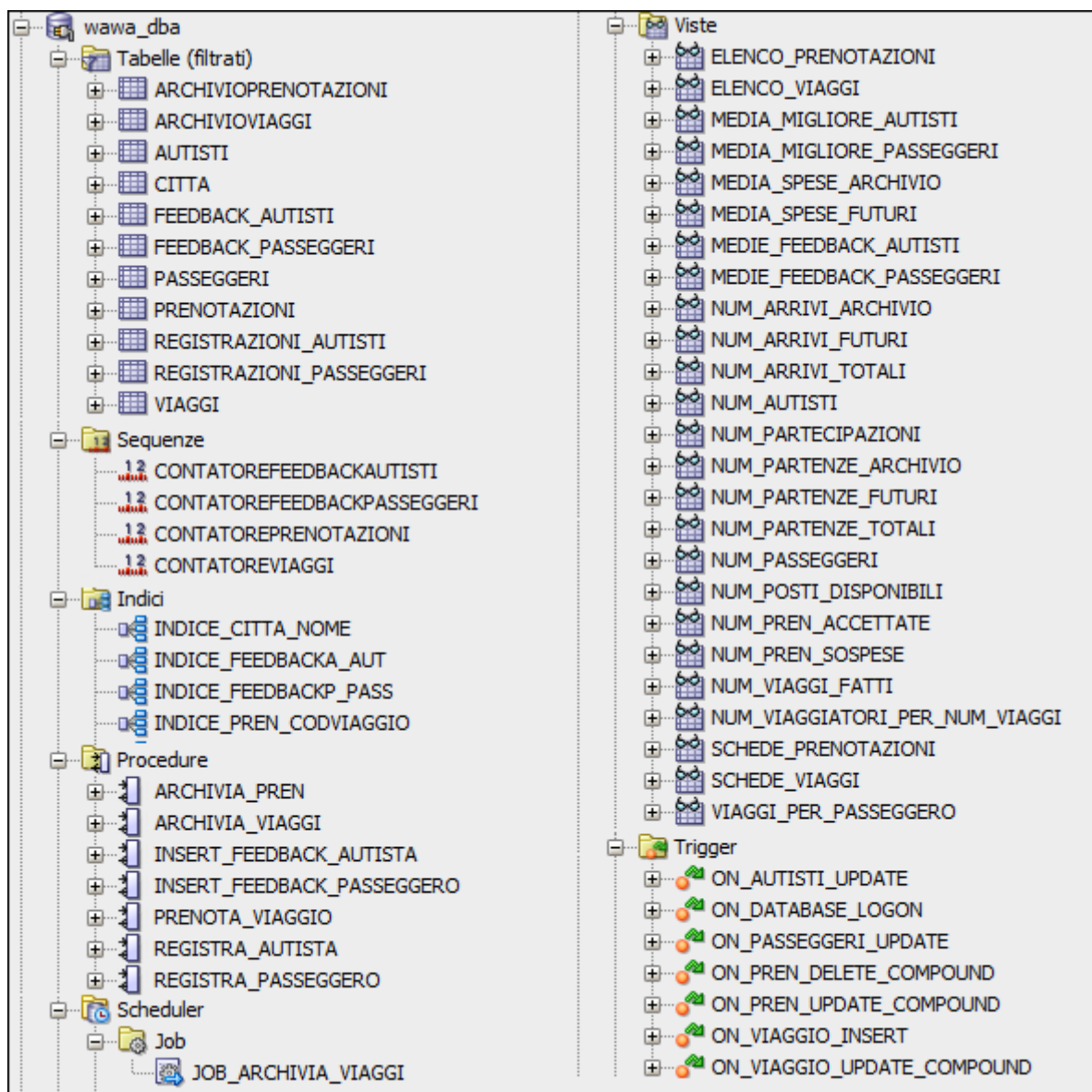
Si è scelto come metodo di storage il **RAID 1**, che mantiene una copia esatta di tutti i dati su almeno due dischi. I vantaggi sono la sua elevata affidabilità (che aumenta linearmente con il numero di copie) e la sua velocità in lettura (che si è ritenuta più importante rispetto alla velocità in scrittura). Si deve, tuttavia, far fronte alla bassa scalabilità del sistema, presupponendo quindi che siano state effettuate corrette analisi a livello direzionale. Si nota esplicitamente che la capacità di scrittura viene ridotta a quella del disco più lento.

Avendo scelto Oracle come DBMS, sono implicite alcune scelte per quanto riguarda il controllo dell'affidabilità del sistema:

- La scrittura sulla base di dati è di tipo differito, avendo Oracle dei *redo file* ma non degli *undo file*
- In caso di guasti soft si utilizza la procedura di **ripresa a caldo** che permette di ripristinare il corretto funzionamento del database a partire dall'ultimo checkpoint
- In caso di guasti hard si procede alla **ripresa a freddo**, utilizzando le copie di backup che Oracle ha effettuato (in questo caso sul disco copia) in modo da ritornare, grazie alla lettura del log, all'ultimo checkpoint ed effettuare quindi una ripresa a caldo

## 5.12 Schema finale

Al termine della progettazione fisica, lo *schema* del DBA è il seguente:<sup>4</sup>



<sup>4</sup>Nella figura sono stati ignorati gli oggetti che Oracle crea di default, ad esempio gli indici sulle chiavi primarie.

## 6 Progettazione dell'applicazione

### 6.1 Introduzione

Le scelte progettuali che hanno riguardato l'applicazione web sono state fatte allo scopo di illustrare l'approccio SQL/CLI e il funzionamento dello standard ODBC di comunicazione con il database. Il team di sviluppo ha concordato, per l'applicazione, l'utilizzo del linguaggio JavaScript (ver. 1.7), supportato dalle opportune librerie. Di seguito si mostrano solo gli aspetti essenziali dell'applicazione, e pertanto si rimanda al codice sorgente per una completa visione della sua implementazione.

Storicamente JavaScript nasce per lo scripting a livello client, ovvero si integrano gli script nelle pagine web HTML e li si eseguono successivamente dal web browser dell'utente. La libreria **Node.js**<sup>5</sup> permette a JavaScript di essere impiegato anche per lo scripting lato server, tramite un'architettura event-driven e capace di I/O asincrono.

D'altra parte, Oracle mette a disposizione **node-oracledb**, un add-on open-source per Node.js che permette il collegamento con il database tramite lo standard di comunicazione ODBC. L'installazione e la configurazione del driver sulla macchina esulano dagli obiettivi del progetto, e pertanto si rimanda alla documentazione ufficiale.<sup>6</sup>

La creazione del server sulla macchina locale si effettua come segue:

```
const serverconfig = {
2   host : 'localhost',
   port : 1337
4 };

6 var server = app.listen(serverconfig.port, serverconfig.host,
   console.log('server online @ http://%s:%s', serverconfig.host, serverconfig.port
   ));
```

Per quanto riguarda il caricamento del driver e la connessione al database:

```
1 // Caricamento del driver oracledb
var oracledb = require(__dirname + '/thirdparty/node_modules/oracledb');
3
// Configurazione dell'user che accede al database Oracle
5 const dbconfig = {
   user : 'wawa_webapp',
7   password : '1234',
   connectionString : 'localhost/XE'
9 };

11 // Esempio di connessione al database
function testConnection()
13 {
   oracledb.getConnection(dbconfig)
15   .then(function (conn) { console.log('accesso al database oracle come %s
      riuscito', oracle_user); return conn.close(); }) // Chiude la connessione
17   .catch(function (err) { console.error(err.message); }); // Sul fallimento del
      tentativo di connessione al database, segnala l'errore
}
```

---

<sup>5</sup><https://nodejs.org/>

<sup>6</sup><https://github.com/oracle/node-oracledb>



Poichè le operazioni sul database sono operazioni asincrone, non è possibile utilizzare gli usuali costrutti di flusso per eseguire le istruzioni nel giusto ordine. L'add-on di Oracle permette, allora, di utilizzare alternativamente funzioni *callback* oppure oggetti *Promise* (.then segnala il completamento di una operazione asincrona mentre .catch ne segnala il fallimento). Si è scelto di impiegare la seconda opzione.

## 6.2 Comunicazione tra client e database

Il client comunica con il server mediante richieste (**request**) di GET (es: carica una pagina) e di POST (per i dati più sensibili, es: esegui il login). Una volta ottenuti i dati richiesti, ad esempio da database, vengono comunicati al client con una risposta (**response**) in formato di stringa o di testo in formato JSON.<sup>7</sup> Lo scambio delle informazioni con il database avviene, in generale, tramite i seguenti passi:

1. Il client manda una request all'applicazione
2. L'applicazione si collega al database tramite driver Oracle
3. Il driver esegue uno statement e salva l'eventuale risultato in un array
4. L'applicazione converte l'array in formato JSON e lo spedisce tramite response al client
5. Il driver chiude la connessione al database

Il driver si occupa, inoltre, di risolvere gli opportuni tipi dei dati e di comunicare eventuali errori. La connessione al database va immediatamente chiusa, da un lato, per non congestionare il listener di Oracle e, dall'altro, per ridurre il rischio che un utente esterno possa introdursi con cattive intenzioni nel database tramite la connessione lasciata aperta.

## 6.3 Accesso come utente passeggero o autista

Abbiamo visto come le specifiche del progetto hanno portato alla creazione di due tipologie di utenti del servizio: gli autisti e i passeggeri. Ci si aspetta, allora, che dal lato applicativo provengano tre informazioni: l'username (poichè identifica univocamente l'utente), la password (affinchè si possa verificare la validità dell'accesso) e infine la tipologia di utente (autista/passeggero, affinchè si possa interrogare la giusta tabella). Supponendo, dunque, che l'applicazione possieda queste tre informazioni, l'accesso come utente si può eseguire nella maniera seguente:

```
app.post('/login', urlEncodedParser, function (req, res) {
2   var login_type = req.body.login_type.toLowerCase();
   var target_table;
4
   if (login_type == 'passeggero')
6     target_table = 'Passeggeri';
   else if (login_type == 'autista')
8     target_table = 'Autisti';
10
   var sql = "SELECT * FROM wawa_dba." + target_table + " WHERE username = '" + req
   .body.login_username + "' AND password = '" + req.body.login_password + "'";
12
   oracledb.getConnection(dbconfig)
   .then(function (conn)
14     {
       conn.execute(sql) // Esecuzione di una query tramite driver
16     .then(function (result)
       {
```

<sup>7</sup><https://www.json.org/json-it.html>

```

18         if (result.rows.length > 0) // Se la query ha prodotto risultati, allora
    ...
        {
20             res.cookie('username', req.body.login_username, { maxAge : 7 * 24 *
                60 * 60 * 1000 })
22             .cookie('type', login_type, { maxAge : 7 * 24 * 60 * 60 * 1000 })
                .redirect('/');
24         }
        else
26         {
            console.error('login errato come %s', login_type);
28             res.end('credenziali di accesso non valide');
        }

30         return conn.close();
32     })
    .catch(function (sql_error)
34     {
        console.error(sql_error);
36         res.end(sql_error);
    });
38 })
    .catch(function (conn_error)
40     {
        console.error(conn_error.message);
42         res.end(conn_error.message);
    });
44 });

```

Si osserva come l'accesso come un utente viene comunicato (e mantenuto) tramite cookie, poichè si vuole implementare un'applicazione web. I cookie tengono traccia dell'username con il quale si è effettuato l'accesso e della tipologia di utente (se autista o passeggero). Nella pratica entrambi i cookie andrebbero codificati, altrimenti chiunque potrebbe aggirare la procedura di accesso semplicemente modificando i propri cookie sul browser.

## 6.4 Ricerca dei viaggi disponibili

Un utente passeggero deve poter specificare la città di partenza, la città di destinazione e la data di partenza (ed eventualmente può voler specificare solo alcune di queste informazioni). Infine l'utente può decidere di visualizzare l'elenco dei viaggi secondo uno specifico ordine (più o meno recente, più o meno economico, o in base al giudizio). La ricerca deve tener conto delle diverse combinazioni per cui l'utente specifica tali informazioni, e deve, inoltre, tener conto dei soli viaggi ancora disponibili e per cui l'utente non si è ancora prenotato.

Il modo migliore per ottenere quanto detto è di spezzare, in un primo momento, il formato di una interrogazione SQL nelle sue varie clausole (SELECT, FROM, WHERE, ecc.), per poi aggiungere man mano i dettagli che l'utente specifica. L'algoritmo per la costruzione della query è di seguito esposto:

```

var partenza = req.body.search_partenza.trim();
2 var arrivo = req.body.search_destinazione.trim();
var data = req.body.search_data.trim();
4
var nested = "SELECT P.codiceViaggio FROM wawa_dba.Prenotazioni P WHERE P.passeggero
    = '" + req.body.username + "'";
6 var select_clause = "SELECT V.codice, V.partenza, V.arrivo, TO_CHAR(V.data), V.nome,
    V.cognome, V.giudizio_medio, V.contributo";
var from_clause = " FROM wawa_dba.Elenco_Viaggi V"; // Vista
8 var where_clause = " WHERE V.disponibile = 'T' AND V.codice NOT IN(" + nested + ")";
var order_clause = "";
10

```

```

12 if (partenza != '')
    where_clause = where_clause + " AND V.partenza = '" + partenza + "'";
14
16 if (arrivo != '')
    where_clause = where_clause + " AND V.arrivo = '" + arrivo + "'";
18
18 if (data != '')
    where_clause = where_clause + " AND V.data = '" + data + "'";
20
22 if (req.body.search_order == 'data-asc')
    order_clause = " ORDER BY V.data";
24 else if (req.body.search_order == 'data-desc')
    order_clause = " ORDER BY V.data DESC";
26 else if (req.body.search_order == 'rating-desc')
    order_clause = " ORDER BY V.giudizio_medio DESC, V.data";
28 else if (req.body.search_order == 'rating-asc')
    order_clause = " ORDER BY V.giudizio_medio ASC, V.data";
30 else if (req.body.search_order == 'contr-desc')
    order_clause = " ORDER BY V.contributo DESC, V.data";
32 else if (req.body.search_order == 'contr-asc')
    order_clause = " ORDER BY V.contributo ASC, V.data";
34
34 var sql = select_clause + from_clause + where_clause + order_clause;
36 // ... Resto di codice in JavaScript per l'esecuzione dello statement

```

E si procede quindi con l'esecuzione dello statement SQL secondo le modalità esposte precedentemente.

## 6.5 Creazione di un viaggio

La creazione di un viaggio avviene tramite uno statement di INSERT. Affinchè le modifiche apportate alla base di dati siano permanenti, è necessario che il driver esegua il commit sulla base di dati (e gestisca, eventualmente, il fallimento del commit stesso). La creazione di un nuovo viaggio e il commit delle modifiche si ottiene come segue:

```

app.post('/crea_viaggio', urlEncodedParser, function (req, res) {
2   var sql = "INSERT INTO wawa_dba.Viaggi(codice, partenza, arrivo, data, autista,
    postiMax, durata, contributo) VALUES (wawa_dba.contatoreViaggi.NEXTVAL, '" + req
    .body.crea_partenza + "', '" + req.body.crea_destinazione + "', TO_DATE('" + req
    .body.crea_data + "', 'YYYY-MM-DD'), '" + req.body.username + "', '" + req.body.crea_postimax +
    "', '" + req.body.crea_durata + "', '" + req.body.crea_contributo + "')";
4
    oracledb.getConnection(dbconfig)
    .then(function (conn)
6    {
        conn.execute(sql)
8        .then(function (result)
        {
10           conn.commit()
            .then(function ()
12            {
                res.send('DB_COMMIT_SUCCESS');
14                return conn.close();
            })
16        .catch(function (commit_error)
        {
18            console.error(commit_error.message);
            res.send(commit_error);
20        });
    });
})

```

```

22     .catch(function (sql_error)
23     {
24         console.error(sql_error.message);
25         res.send(sql_error);
26     });
27 })
28 .catch(function (conn_error)
29 {
30     console.error(conn_error.message);
31     res.send(conn_error.message);
32 });
33 });

```

## 6.6 Utilizzo delle viste

Nella progettazione fisica ci si è preoccupati di creare delle viste al fine di semplificare, tra le altre cose, le operazioni sulla base di dati. È infatti possibile, ora, utilizzare tali viste per eseguire facilmente delle query dal livello applicazione, in quanto è la vista a preoccuparsi (nella maggior parte dei casi) di selezionare i giusti attributi e di fare gli opportuni join. La distribuzione Oracle 11g XE, oltre alle consuete ottimizzazioni delle query, è in grado di eliminare dalla chiamata ad una vista gli join che non concorrono a formare il risultato finale (*Join Elimination*).<sup>8</sup> Alcune query possono allora essere eseguite come segue:

```

1 // Ottieni tutti i viaggi creati da uno specifico autista
2 var sql = "SELECT codice, partenza, arrivo, data FROM wawa_dba.Schede_Viaggi WHERE
3     username_autista = '" + req.body.username + "' ORDER BY data";
4
5 // Ottieni tutte le prenotazioni effettuate da uno specifico passeggero
6 var sql = "SELECT * FROM wawa_dba.Elenco_Prenotazioni WHERE username_passeggero = '"
7     + req.body.username + "' ORDER BY data_viaggio";
8
9 // Ottieni tutte le prenotazioni relative ai viaggi creati da uno specifico autista
10 var sql = "SELECT * FROM wawa_dba.Elenco_Prenotazioni WHERE username_autista = '" +
11     req.body.username + "' AND accettato = 'F'";
12
13 // Ottieni le informazioni riguardanti uno specifico viaggio
14 var sql = "SELECT * FROM wawa_dba.Schede_Viaggi WHERE codice = '" + req.body.
15     codice_viaggio;
16
17 // Ottieni le informazioni relative ad una specifica prenotazione
18 var sql = "SELECT * FROM wawa_dba.Schede_Prenotazioni WHERE codice_prenotazione = '"
19     + req.body.codice_pren + "'";

```

## 6.7 Altre operazioni sulle base di dati

Di seguito sono elencati degli statement di esempio per eseguire altre operazioni sulla base di dati. Si traslascia, per motivi di leggibilità, il codice JavaScript completo (del tutto analogo a quello già precedentemente mostrato).

```

1 // Prenota un viaggio
2 var sql = "INSERT INTO wawa_dba.Prenotazioni VALUES (wawa_dba.
3     contatorePrenotazioni.NEXTVAL, "
4     + req.body.codice_viaggio + ", '" + req.body.username + "', SYSDATE, 'F')";

```

<sup>8</sup><https://antognini.ch/2010/01/join-elimination/>

```

6 // Accetta una prenotazione
var sql = "UPDATE wawa_dba.Prenotazioni SET accettato = 'T' WHERE codice = '" +
req.body.codice_pren + "'";

8

10 // Elimina un viaggio
var sql = "DELETE FROM wawa_dba.Viaggi WHERE codice = '" + req.body.
codice_viaggio + "'";

12

14 // Cancella una prenotazione
var sql = "DELETE FROM wawa_dba.Prenotazioni WHERE codice = '" + req.body.
codice_pren + "'";

16

18 // Ottieni le informazioni sul profilo PERSONALE per un autista
var sql = "SELECT nome, cognome, email, citta, via, civico, telefono,
numeroPatente, TO_CHAR(scadenzaPatente), targaAuto, modelloAuto \
FROM wawa_dba.Autisti WHERE username = '" + req.body.username + "'";

20

22 // Ottieni le informazioni sul profilo PERSONALE per un passeggero
var sql = "SELECT nome, cognome, email, citta, via, civico, telefono, documento
FROM wawa_dba.Passeggeri WHERE username = '" + req.body.username + "'";

24

26 // Modifica il profilo di un autista
var sql = "UPDATE wawa_dba.Autisti SET nome = '" + req.body.user_nome + "',
cognome = '" + req.body.user_cognome + "', email = '" + req.body.user_email + "
', telefono = '" + req.body.user_telefono + "', citta = '" + req.body.user_citta
+ "', via = '" + req.body.user_via + "', civico = '" + req.body.user_civico + "
', numeroPatente = '" + req.body.user_patente + "', scadenzaPatente = TO_DATE('"
+ req.body.user_scadenza + "'), targaAuto = '" + req.body.user_targa + "',
modelloAuto = '" + req.body.user_modello + "' WHERE username = '" + req.cookies.
username + "'";

28

30 // Modifica il profilo di un passeggero
var sql = "UPDATE wawa_dba.Passeggeri SET nome = '" + req.body.user_nome + "',
cognome = '" + req.body.user_cognome + "', email = '" + req.body.user_email + "
', telefono = '" + req.body.user_telefono + "', citta = '" + req.body.user_citta
+ "', via = '" + req.body.user_via + "', civico = '" + req.body.user_civico + "
', documento = '" + req.body.user_documento + "' WHERE username = '" + req.
cookies.username + "'";

```

## 7 Livello presentazione

### 7.1 Dal livello applicazione al livello presentazione

È stato, infine, progettato un esempio di sito web in HTML/CSS e JavaScript. Le operazioni eseguibili dal sito web, risultato della progettazione dell'applicazione, sono, pertanto, le seguenti:

- Registrazione di nuovi utenti
- Accesso al servizio come autista o come passeggero (tramite cookie)
- Ricerca, visualizzazione e prenotazione di viaggi disponibili
- Visualizzazione e accettazione delle prenotazioni
- Modifica del proprio profilo utente
- Visualizzazione delle informazioni relative ad un autista (con voto)
- Visualizzazione delle informazioni relative ad un passeggero (con voto)
- Visualizzazione e cancellazione dei propri viaggi creati
- Visualizzazione e cancellazione delle proprie prenotazioni effettuate

### 7.2 Visualizzazione del voto medio

A livello di database il giudizio medio relativo ad un utente è espresso sotto forma di numero, ottenuto tramite un opportuno *average* (AVG) di tutti i voti relativi a quell'utente. A livello presentazione, tuttavia, può risultare più espressivo mostrare il voto con delle "stelline" (da una a cinque) come di seguito mostrato:

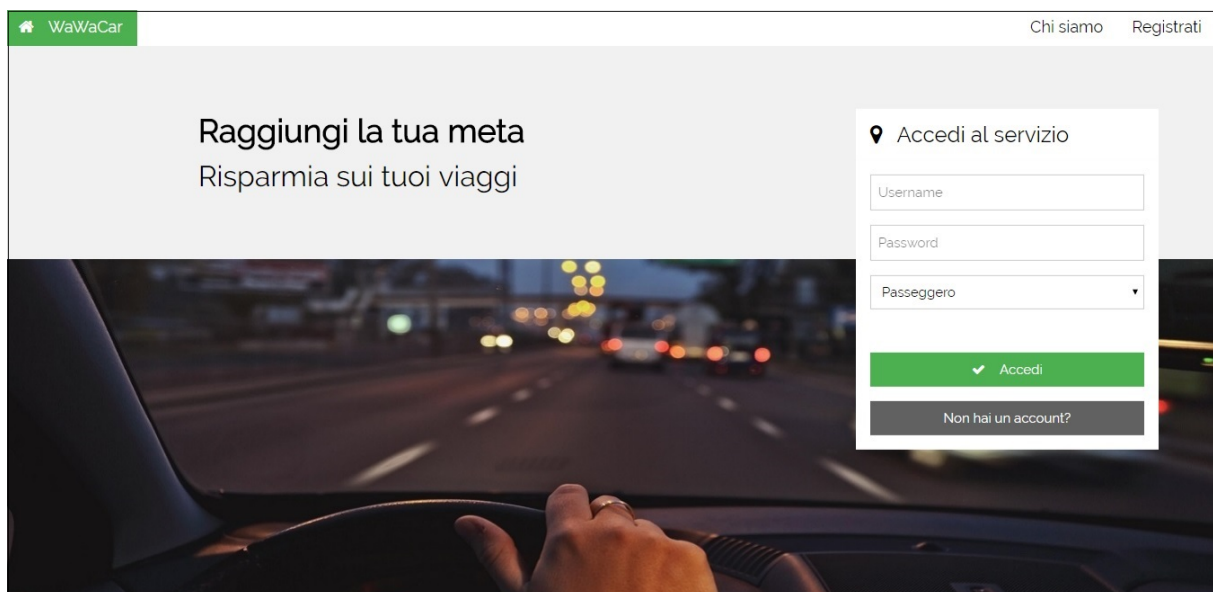
Monica Bellucci	★★★★★
Giacomo Leopardi	★★★★☆
Diletta Leotta	★★★★★

L'algoritmo JavaScript è il seguente:

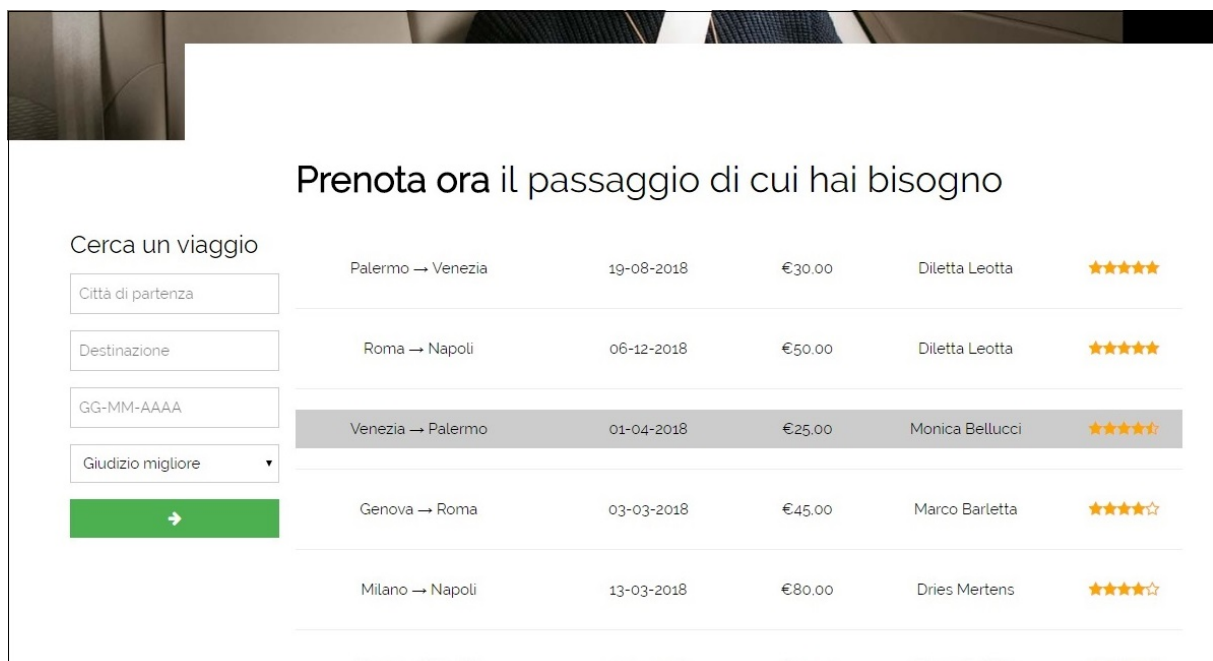
```
1 var rating = '';  
3 for (var j = 1; j <= 5; ++j) // j rappresenta la stellina che si sta considerando  
{  
5   if (j <= media) // Se la stellina deve essere intera  
      rating = rating + '<span class="fa fa-star" style="color:orange"></span>';  
7   else if (j - media < 1) // Se la stellina deve essere a meta'  
      rating = rating + '<span class="fa fa-star-half-o" style="color:orange"></span>';  
9   else  
      rating = rating + '<span class="fa fa-star-o" style="color:orange"></span>';  
11 }
```

### 7.3 Esempi di pagine web


Sono di seguito esposte alcune delle pagine web per eseguire operazioni sulla base di dati da browser:



Homepage e modulo per l'accesso



Visualizzazione e ricerca dei viaggi



## Venezia → Palermo

Partenza: Venezia (30100)

Destinazione: Palermo (90121)

Giorno: 01-04-2018

Giudizio medio autista: 4.5/5

Autista: Monica Bellucci

Telefono: 3332224444

Auto: Audi R8

Dettagli aggiuntivi: N/A

Posti disponibili: 3

Contributo a persona: €25

Tempi di percorrenza stimati: 02:30

→ Prenota

← Torna indietro

Copyright © 2017 - 2018 WaWaCar

Università degli Studi di Napoli Federico II

Marco Bocchetti - Marco Barletta - Davide Biancardi - Stefano Aramu

Visualizzazione delle informazioni riguardanti il viaggio e l'autista

## Modifica il tuo profilo e gestisci i tuoi viaggi

### Modifica profilo

Nome:

Telefono:

Cognome:

Numero patente:

E-mail:

Scadenza patente:

Città:

Targa auto:

Via:

Modello auto:

Civico:

✓ Modifica

### Viaggi creati

Roma → Napoli	06-12-2018
Palermo → Venezia	19-08-2018

Visualizzazione del proprio profilo utente e dei viaggi creati