SOLUZIONE ESAME DI STATO 2016/2017

Indirizzo: ITIA - INFORMATICA E TELECOMUNICAZIONI ARTICOLAZIONE INFORMATICA

La traccia di esame, pubblicata dal Ministero, si trova al seguente link: esame 2017

La traccia d'esame di quest'anno, ritenuta dagli studenti molto lineare e anche semplice, chiede di realizzare una applicazione web che, in effetti, esiste da diversi anni (*blablacar*, *viaggiareinsieme* e molti altri siti dedicati al *car pooling*).
Cosa dovrà fare l'applicazione, che andremo a realizzare, appare semplice per chi sa cosa sia il car pooling, per chi invece non ne ha mai sentito parlare (alcuni studenti davvero non avevano mai sentito questo termine) sarebbe stato utile inserire un esempio.

Esempio: Gli studenti universitari fuori sede spesso, nelle stesse date per il weekend o le festività, devono affrontare lunghi viaggi di rientro verso casa e cercano di riempire la loro auto per ottimizzare i costi di benzina ed autostrada. Un sito di *car pooling* è l'ideale per chi vuole fare un viaggio a costi contenuti oppure desidera ammortizzare il costo del proprio viaggio in auto.

Quesito 1 - un'analisi della realtà di riferimento, giungendo alla definizione di uno schema concettuale della base di dati che, a suo motivato giudizio, sia idoneo a gestire la realtà presentata;

Leggendo la traccia, come suggerisco sempre agli studenti, individuiamo, evidenziandoli, i soggetti candidati a diventare le entità nel nostro modello concettuale.

Un'azienda *start-up* vuole costruire una piattaforma Web 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.
Gli utenti della piattaforma possono essere di due tipi: utenti-autisti (coloro che offrono un passaggio con la propria macchina) e utenti-passeggeri (coloro che usufruiscono del passaggio).
Gli autisti devono registrarsi sul sito ed inserire i propri dati: generalità, numero e scadenza patente di guida, dati dell'automobile utilizzata, recapito telefonico, email, fotografia.
Per ogni viaggio che intendono condividere, gli autisti devono indicare città di partenza, città di destinazione, data ed ora di partenza, contributo economico richiesto ad ogni passeggero, tempi di percorrenza stimati. È responsabilità dell'autista, mano a mano che accetterà passeggeri per un certo viaggio, dichiarare chiuse le prenotazioni per quel viaggio, utilizzando un'apposita funzione sul portale.
L'utente-passeggero si deve registrare sulla piattaforma, indicando cognome e nome, documento di identità, recapito telefonico ed email. La piattaforma fornisce ai passeggeri la possibilità di indicare città di partenza e di destinazione e data desiderata; presenta quindi un elenco di viaggi (per cui non siano ancora chiuse le prenotazioni), ciascuno con le caratteristiche dell'autista e le modalità del viaggio stesso inserite dall'autista (orario, eventuali soste previste alle stazioni di servizio, possibilità di caricare bagaglio o animali, …).
Il passeggero sceglie quindi il viaggio desiderato con il corrispondente autista, anche esaminando il voto medio e i giudizi dei *feedback* assegnati tramite la piattaforma dai precedenti passeggeri all'autista stesso, e si prenota. Le informazioni sul passeggero vengono inviate per email dalla piattaforma all'autista scelto, il quale può consultare sul portale il voto medio e i giudizi dei *feedback* ricevuti dal passeggero da parte di precedenti autisti e decidere se accettarlo o meno. Il passeggero di conseguenza riceverà una email di accettazione o di rifiuto della prenotazione effettuata, contenente, in caso di accettazione, un promemoria con città di partenza e destinazione, data e orario del viaggio, dati dell'autista e della sua automobile.
A viaggio effettuato, il passeggero può inserire un *feedback* sull'autista, espresso sia in forma di voto numerico che di giudizio discorsivo. A sua volta, l'autista può inserire un *feedback* sul passeggero, espresso sia in forma di voto numerico che di giudizio discorsivo. Sia i voti medi che i singoli giudizi dei *feedback* ricevuti da ciascun autista sono disponibili ai passeggeri; analogamente, sia i voti medi che i singoli giudizi dei *feedback* ricevuti da ciascun passeggero sono disponibili agli autisti.

All'interno della traccia abbiamo selezionato i candidati a diventare le nostre entità nel modello concettuale:

- **utenti-autisti**
- **utenti-passeggeri**
- **viaggi**
- **feedback**

Ricordiamo che un utente-passeggero **prenota** un certo viaggio. Sappiamo che nel nostro database esisterà una tabella **prenotazione**, ma nel modello concettuale, quando si tratta di una azione, che lega due entità, è preferibile indicare "*prenotazione*" come una relazione con degli attributi. La prenotazione non esiste se non legata ad un viaggio e ad un utente, per questo è più corretto indicarla come una relazione.

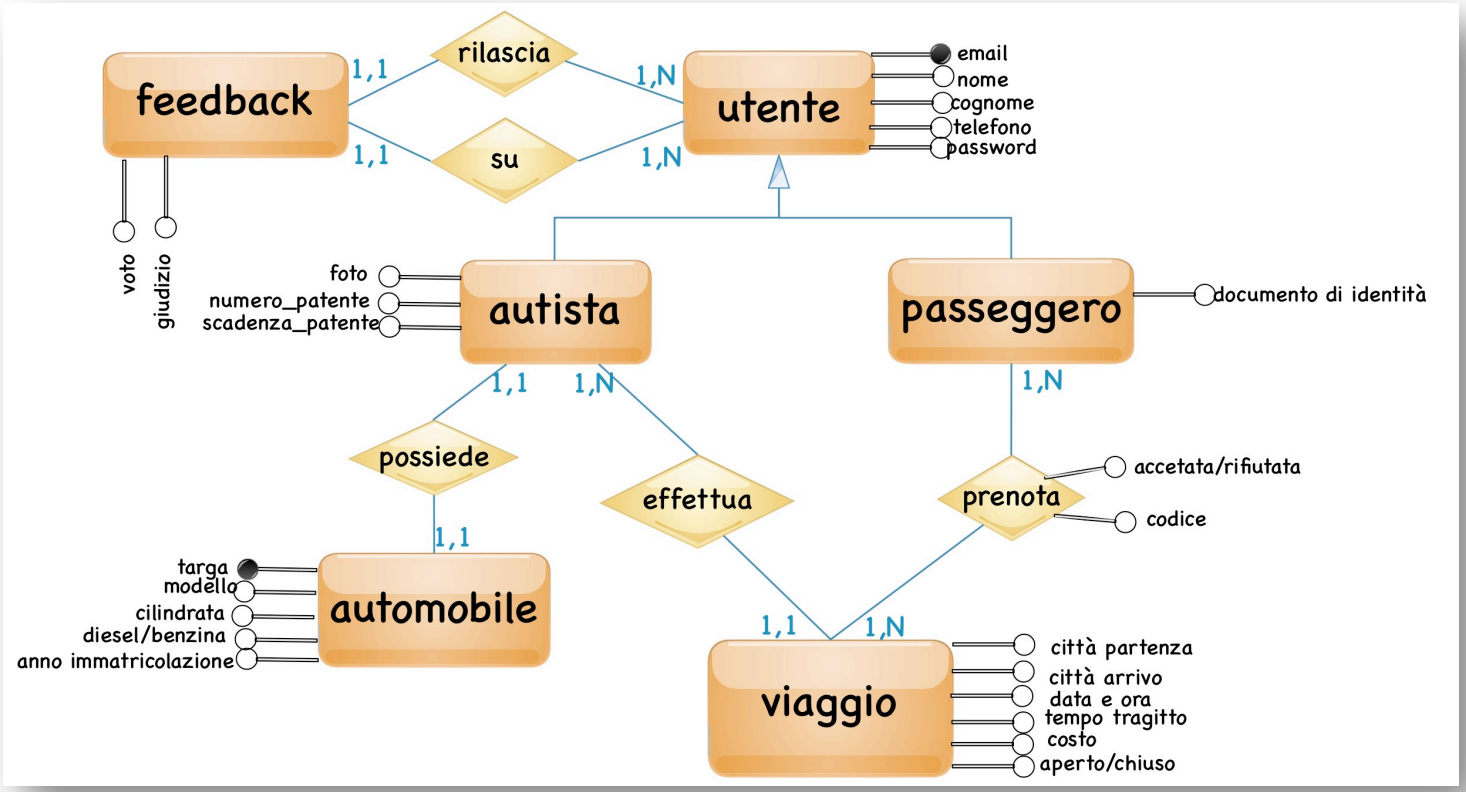
Rileggiamo la traccia selezionando stavolta quelli che appaiono come attributi delle nostre entità:

Un'azienda *start-up* vuole costruire una piattaforma Web 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.
Gli utenti della piattaforma possono essere di due tipi: utenti-autisti (coloro che offrono un passaggio con la propria macchina) e utenti-passeggeri (coloro che usufruiscono del passaggio).
Gli autisti devono registrarsi sul sito ed inserire i propri dati: generalità, numero e scadenza patente di guida, dati dell'automobile utilizzata, recapito telefonico, email, fotografia.
Per ogni viaggio che intendono condividere, gli autisti devono indicare città di partenza, città di destinazione, data ed ora di partenza, contributo economico richiesto ad ogni passeggero, tempi di percorrenza stimati. È responsabilità dell'autista, mano a mano che accetterà passeggeri per un certo viaggio, dichiarare chiuse le prenotazioni per quel viaggio, utilizzando un'apposita funzione sul portale.
L'utente-passeggero si deve registrare sulla piattaforma, indicando cognome e nome, documento di identità, recapito telefonico ed email. La piattaforma fornisce ai passeggeri la possibilità di indicare città di partenza e di destinazione e data desiderata; presenta quindi un elenco di viaggi (per cui non siano ancora chiuse le prenotazioni), ciascuno con le caratteristiche dell'autista e le modalità del viaggio stesso inserite dall'autista (orario, eventuali soste previste alle stazioni di servizio, possibilità di caricare bagaglio o animali, …).
Il passeggero sceglie quindi il viaggio desiderato con il corrispondente autista, anche esaminando il voto medio e i giudizi dei *feedback* assegnati tramite la piattaforma dai precedenti passeggeri all'autista stesso, e si prenota. Le informazioni sul passeggero vengono inviate per email dalla piattaforma all'autista scelto, il quale può consultare sul portale il voto medio e i giudizi dei *feedback* ricevuti dal passeggero da parte di precedenti autisti e decidere se accettarlo o meno. Il passeggero di conseguenza riceverà una email di accettazione o di rifiuto della prenotazione effettuata, contenente, in caso di accettazione, un promemoria con città di partenza e destinazione, data e orario del viaggio, dati dell'autista e della sua automobile.
A viaggio effettuato, il passeggero può inserire un *feedback* sull'autista, espresso sia in forma di voto numerico che di giudizio discorsivo. A sua volta, l'autista può inserire un *feedback* sul passeggero, espresso sia in forma di voto numerico che di giudizio discorsivo. Sia i voti medi che i singoli giudizi dei *feedback* ricevuti da ciascun autista sono disponibili ai passeggeri; analogamente, sia i voti medi che i singoli giudizi dei *feedback* ricevuti da ciascun passeggero sono disponibili agli autisti.

utenti-autisti *attributi*: generalità (nome, cognome, data di nascita, comune di nascita), numero e scadenza patente di guida, dati dell'automobile (modello, cilindrata, diesel/benzina, anno di immatricolazione, targa), recapito telefonico, email, fotografia.
utenti-passeggeri *attributi*: cognome e nome, documento di identità, recapito telefonico ed email
viaggi *attributi*: città di partenza, città d'arrivo, data ed ora di partenza, contributo economico richiesto ad ogni passeggero, tempi, aperto/chiuso (ovvero c'è ancora possibilità di prenotare o meno. Chiaramente riguarda il viaggio e non la singola prenotazione che potrebbe venire singolarmente accettata o meno. Apprezzo l'autore della traccia per aver chiarito che esplicitamente un autista sul sistema chiude il proprio *viaggio* indicando che è al completo. Questo non porta ambiguità su dove inserire questo attributo.)
feedback *attributi*: voto numerico, giudizio discorsivo.
Relazione **prenota** *attributi*: stato (accettata/rifiutata/in attesa), codice prenotazione (chiaramente citato nella sezione riguardante le interrogazioni alla base dati punto 3c).

Osservazioni: ci sono apparentemente alcune differenze tra utenti-autisti e utenti-passeggeri ma la parola stessa dell'entità ci fa optare per disegnare una generalizzazione di una entità generica '**utente**', di come poi verrà tradotta, nel modello logico, ce ne occuperemo nella risoluzione del punto 2.
L'**automobile**, per l'utente-autista, appare un attributo 'abbondante', anche perché davvero riguarda un altro oggetto e non l'utente vero e proprio, per cui, per un corretto disegno ER, deve diventare una entità a parte e comunque una stessa auto potrebbe essere guidata da autisti differenti.

Passiamo a disegnare il modello concettuale:



Entità

L'entità centrale è UTENTE che generalizza l'entità AUTISTA e PASSEGGERO. Per questa entità, a parte gli attributi comuni, diventa naturale assegnare anche una password di accesso al sistema di car pooling anche se nella traccia non se ne fa menzione.

L'AUTISTA possiede una AUTOMOBILE ed effettua un certo VIAGGIO.

I FEEDBACK sono inseriti da autisti per passeggeri e viceversa per cui è diventa naturale inserirli legati all'entità generica UTENTE con doppia relazione, ovvero un utente lascia un feedback su un altro utente, per cui li scrive ma li subisce nello stesso tempo.

Relazioni

Analizziamo le relazioni sulle entità individuate:
UTENTE – FEEDBACK – **Uno a molti** (relazione *rilascia*) - Un utente può scrivere N feedback, un feedback può essere scritto da un solo utente
UTENTE – FEEDBACK – **Uno a molti** (relazione *su*) - Un utente può ricevere N feedback, un feedback può essere riferito da un solo utente.
AUTISTA –VIAGGIO – **Uno a molti** – Un autista *effettua* molti viaggi, un viaggio è portato a termine da un solo autista.
AUTISTA - AUTOMOBILE **– Uno a uno** – Un autista *possiede* una sola macchina, una macchina ha un solo proprietario. Appare inutile e inserire l'eccezione in cui un autista possiede più di una macchina oppure in cui una auto venga utilizzata da più autisti differenti. Atteniamoci alla traccia e possiamo assumere che questa relazione sia serenamente uno a uno.
PASSEGGERO-VIAGGIO – **Molti a molti** – Un passeggero *prenota* molti viaggi, ogni viaggio può avere più prenotazioni.

Quesito 2. Il relativo schema logico

Utilizziamo le regole di trasformazione per creare il modello logico.

Traduzione generalizzazione

Prima regola di trasformazione: Ogni entità diventa una tabella, a meno di relazioni di generalizzazione/specializzazione.
Nel nostro caso, dovremo prendere una decisione se risolvere la nostra generalizzazione con un "*collasso in alto*" o un "*collasso in basso*", ovvero se dovremo avere nella nostra base dati solo una tabella UTENTE, oppure due tabelle AUTISTA e PASSEGERO.
Leggendo superficialmente la traccia, con l'idea di non complicarsi la vita inutilmente, si potrebbe essere tentati ad un semplice, anche se inverosimile, "*collasso in basso*", ovvero avremo nella nostra base dati due entità AUTISTA e PASSEGERO.

L'autore della traccia ha definito degli attributi diversi e alcune interazioni tra le due entità scindendo autista e passeggero proprio per rendere anche un collasso in basso (le due entità divise) una soluzione praticabile e perfettamente aderente alle richieste. Non sarebbe un errore in assoluto anche se un collasso in basso si porta dietro diverse conseguenze.
Si parte dalla considerazione banale che un autista, che per un giorno volesse essere scarrozzato, dovrebbe fare una doppia iscrizione all'applicazione di car pooling, perché solo un passeggero potrebbe richiedere una prenotazione. Ma se osserviamo gli attributi, di fatto un autista non dovrebbe fare una nuova registrazione perché, in quanto autista, ha già inserito gli estremi della sua patente ed è maggiormente identificato e profilato rispetto ad un normale utente-passeggero.
A sua volta un passeggero, che poi volesse utilizzare la propria auto per un viaggio in compagnia, dovrebbe avere la possibilità di proporre un viaggio in cui lui è un autista.
Come tutte le applicazioni web, se un utente volesse inserire un viaggio, dovrebbe

semplicemente riempire una ulteriore form con dati dell'auto, della propria patente e far upload di una foto. Gli basterebbe solo integrare i dati minimi forniti come passeggero, con i dati forniti, per essere un autista.

Sempre per convincerci che non è praticabile il collasso in basso, l'entità FEEDBACK dovrebbe essere sdoppiata per la necessità di memorizzare i feedback che gli autisti rilasciano ai passeggeri e quelli che i passeggeri rilasciano agli autisti. Su questo la traccia è stata molto chiara dividendo nettamente le due entità e non prevedendo il naturale feedback che un passeggero potrebbe anche lasciare sugli altri compagni di viaggio, creando un feedback passeggero per un altro passeggero.

Alla fine, per snellire anche la gestione dei feedback e per essere coerenti al fatto sensato che un autista può comportarsi da passeggero e viceversa, optiamo per un **collasso in alto**, creando una sola una **singola** tabella **UTENTE** e non inseriremo neanche un ruolo.
Sarà l'applicativo che bloccherà un passeggero senza auto collegata e i dati necessari se volesse inserire un viaggio. Un autista, invece, può prenotare viaggi proposti da altri autisti senza restrizioni o vincoli di ruolo.

Traduzione molti a molti

L'unica relazione molti a molti tra passeggeri e viaggi si risolve con la relazione PRENOTA che diventerà la tabella PRENOTAZIONE che lega un passeggero al viaggio e avrà uno stato iniziale 'in attesa' che poi potrà essere 'accettata' o 'rifiutata'.

Traduzione uno a molti

Le relazioni uno a molti si risolvono inserendo le chiavi esterne nelle tabelle a 'molteplicità' molti.
Sempre per i FEEDBACK, appare opportuno legarlo anche al viaggio, altrimenti non si evince che solo dopo aver realmente effettuato un viaggio (o da autista o da passeggero) si può rilasciare un feedback. E volendo, per viaggi diversi con la stessa persona, si potrebbero rilasciare feedback differenti perché magari in un viaggio è andato tutto bene mentre in un successivo viaggio tutto è stato un disastro. E' una finezza applicativa, più che altro, quindi va bene che sia inserita in questa fase e non nel diagramma E/R che ne risulterebbe appesantito.

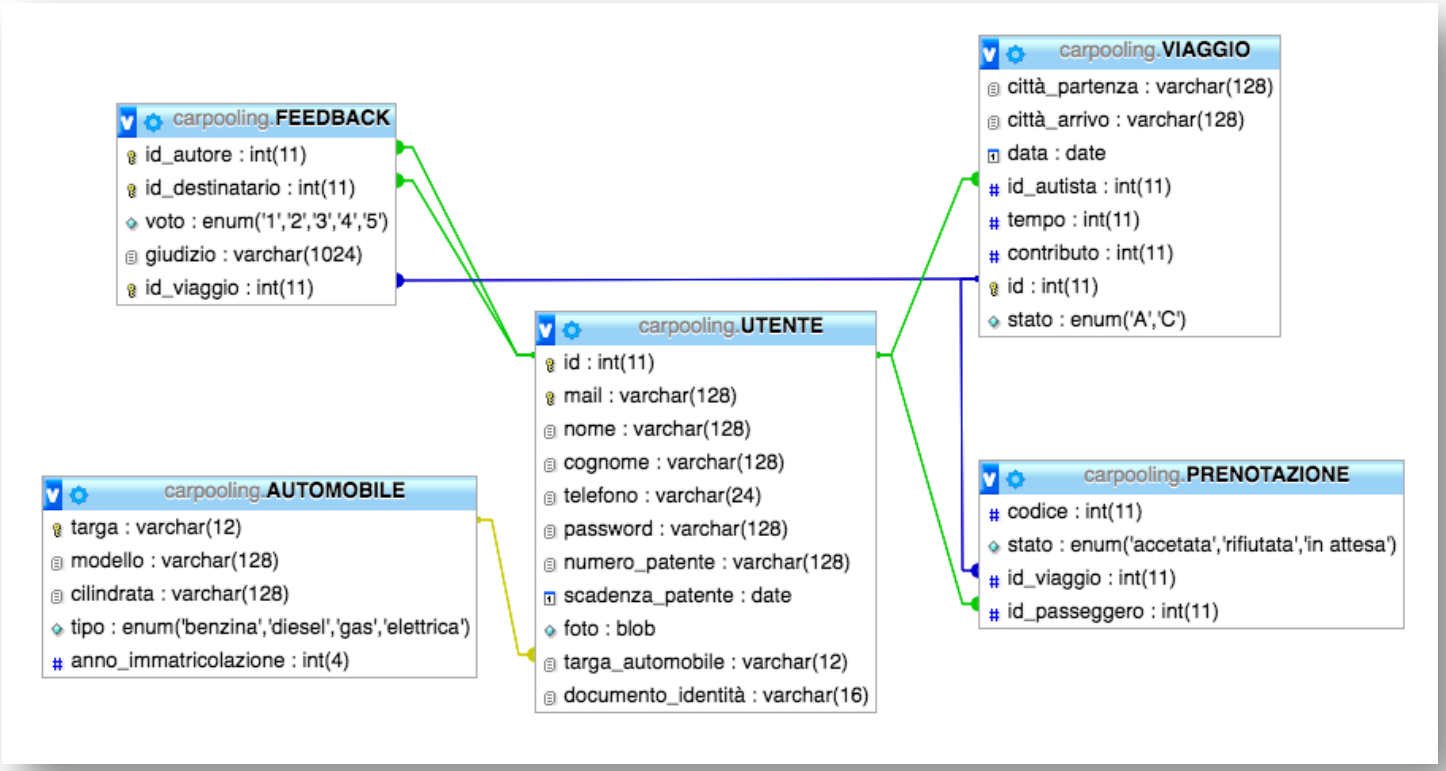
Traduzione uno a uno

Per quanto riguarda UTENTE e AUTOMOBILE decidiamo di portare la chiave esterna dell'automobile sull'UTENTE, perché appare evidente che un utente non è un autista se non indica una automobile. In questo modo il sistema, senza effettuare query su altre tabelle ma controllando solo i dati della patente e se la targa auto è NULL o è valorizzata, decide se quell'utente è abilitato o meno ad inserire un viaggio.
Per un utente passeggero la chiave esterna verso automobile sarà NULL.

Chiavi primarie e tipi di attributi

Inseriamo per comodità e per motivi di performance degli id come chiavi primarie.
Un feedback avrà una chiave primaria tripla (id_autore, id_destinatario e id_viaggio) per evitare che sia possibile per lo stesso viaggio inserire più di un feedback.
La foto dell'utente è un BLOB anche se potrebbe anche essere anche una path ad una foto memorizzata sul Webserver.
Per i tipi di dati che ammettono solo un sottoinsieme di valori abbiamo inserito delle ENUM che per lo *stato* del viaggio (aperto o chiuso), per il tipo di carburante dell'auto e per lo stato della prenotazione.

Ed ecco il nostro modello logico:



Ecco il modello fisico, le CREATE_TABLE del MySQL dove si evince che alcuni campi dell'UTENTE possono ammettere valori NULL (nel caso di utenti solo passeggeri) e le relazioni tra le tabelle:

```
CREATE TABLE IF NOT EXISTS `AUTOMOBILE` (
  `targa` varchar(12) NOT NULL,
  `modello` varchar(128) NOT NULL,
  `cilindrata` varchar(128) NOT NULL,
  `tipo` enum('benzina','diesel','gas','elettrica') NOT NULL,
  `anno_immatricolazione` int(4) NOT NULL,
  PRIMARY KEY (`targa`)
);
```

```
CREATE TABLE IF NOT EXISTS `FEEDBACK` (
  `id_autore` int(11) NOT NULL,
  `id_destinatario` int(11) NOT NULL,
  `voto` enum('1','2','3','4','5') NOT NULL,
  `giudizio` varchar(1024) NOT NULL,
  `id_viaggio` int(11) NOT NULL,
  PRIMARY KEY (`id_autore`,`id_destinatario`,`id_viaggio`),
  KEY `id_destinatario` (`id_destinatario`)
);

CREATE TABLE IF NOT EXISTS `PRENOTAZIONE` (
  `codice` int(11) NOT NULL,
  `stato` enum('accetata','rifiutata','in attesa') NOT NULL,
  `id_viaggio` int(11) NOT NULL,
  `id_passeggero` int(11) NOT NULL
);

CREATE TABLE IF NOT EXISTS `UTENTE` (
  `id` int(11) NOT NULL,
  `mail` varchar(128) NOT NULL,
  `nome` varchar(128) NOT NULL,
  `cognome` varchar(128) NOT NULL,
  `telefono` varchar(24) NOT NULL,
  `password` varchar(128) NOT NULL,
  `numero_patente` varchar(128) DEFAULT NULL,
  `scadenza_patente` date DEFAULT NULL,
  `foto` blob,
  `targa_automobile` int(11) DEFAULT NULL,
  `documento_identità` varchar(16) DEFAULT NULL,
  PRIMARY KEY (`id`)
);

CREATE TABLE IF NOT EXISTS `VIAGGIO` (
  `città_partenza` varchar(128) NOT NULL,
  `città_arrivo` varchar(128) NOT NULL,
  `data` date NOT NULL,
  `id_autista` int(11) NOT NULL,
  `tempo` int(11) NOT NULL,
  `contributo` int(11) NOT NULL,
  `id` int(11) NOT NULL,
  `stato` enum('A','C') NOT NULL,
  PRIMARY KEY (`id`)
);


--
-- Constraints for table `FEEDBACK`
--
ALTER TABLE `FEEDBACK`
  ADD CONSTRAINT `feedback_ibfk_4` FOREIGN KEY (`id_viaggio`) REFERENCES
`VIAGGIO` (`id`),
  ADD CONSTRAINT `feedback_ibfk_1` FOREIGN KEY (`id_autore`) REFERENCES
`AUTISTA` (`id`),
```

```
  ADD CONSTRAINT `feedback_ibfk_3` FOREIGN KEY (`id_destinatario`)
REFERENCES `UTENTE` (`id`);

--
-- Constraints for table `PRENOTAZIONE`
--
ALTER TABLE `PRENOTAZIONE`
  ADD CONSTRAINT `prenotazione_ibfk_2` FOREIGN KEY (`id_passeggero`)
REFERENCES `UTENTE` (`id`),
  ADD CONSTRAINT `prenotazione_ibfk_1` FOREIGN KEY (`id_viaggio`)
REFERENCES `VIAGGIO` (`id`);

--
-- Constraints for table `VIAGGIO`
--
ALTER TABLE `VIAGGIO`
  ADD CONSTRAINT `viaggio_ibfk_1` FOREIGN KEY (`id_autista`) REFERENCES
`UTENTE` (`id`);
```

Quesito 3 - le seguenti interrogazioni espresse in linguaggio SQL:

- a) data una città di partenza, una di arrivo e una data, elencare gli autisti che propongono un viaggio corrispondente con prenotazioni non ancora chiuse, in ordine crescente di orario, riportando i dati dell'auto e il contributo economico richiesto;

```
SELECT nome, cognome, mail, modello, cilindrata, contributo,
DATE_FORMAT(data, '%hh:%mm') FROM VIAGGIO, utente, AUTOMOBILE
WHERE città_partenza='ROMA' and città_arrivo='MILANO'
and data='2017-06-26' and viaggio.id_autista = utente.id
and viaggio.stato = 'A' and automobile.targa =
utente.targa_automobile
ORDER BY data
```

- b) dato il codice di una prenotazione accettata, estrarre i dati necessari per predisporre l'email di promemoria da inviare all'utente passeggero;

Per questa query rileggiamo la traccia, per il promemoria viene richiesto:
"*Il passeggero di conseguenza riceverà una email di accettazione o di rifiuto della prenotazione effettuata, contenente, in caso di accettazione, un promemoria con città di partenza e destinazione, data e orario del viaggio, dati dell'autista e della sua automobile.* "

```
SELECT città_partenza, città_arrivo,
nome as 'Nome Autista', cognome as 'Cognome Autista', mail,
modello, cilindrata, contributo,
DATE_FORMAT(data, '%gg/%MM/%YYYY') as data,
DATE_FORMAT(data, '%hh:%mm') as orario
 FROM VIAGGIO, utente, AUTOMOBILE, prenotazione
WHERE  viaggio.id = prenotazione.id_viaggio
and viaggio.id_autista = utente.id
and automobile.targa = utente.targa_automobile
and codice = '1'
```

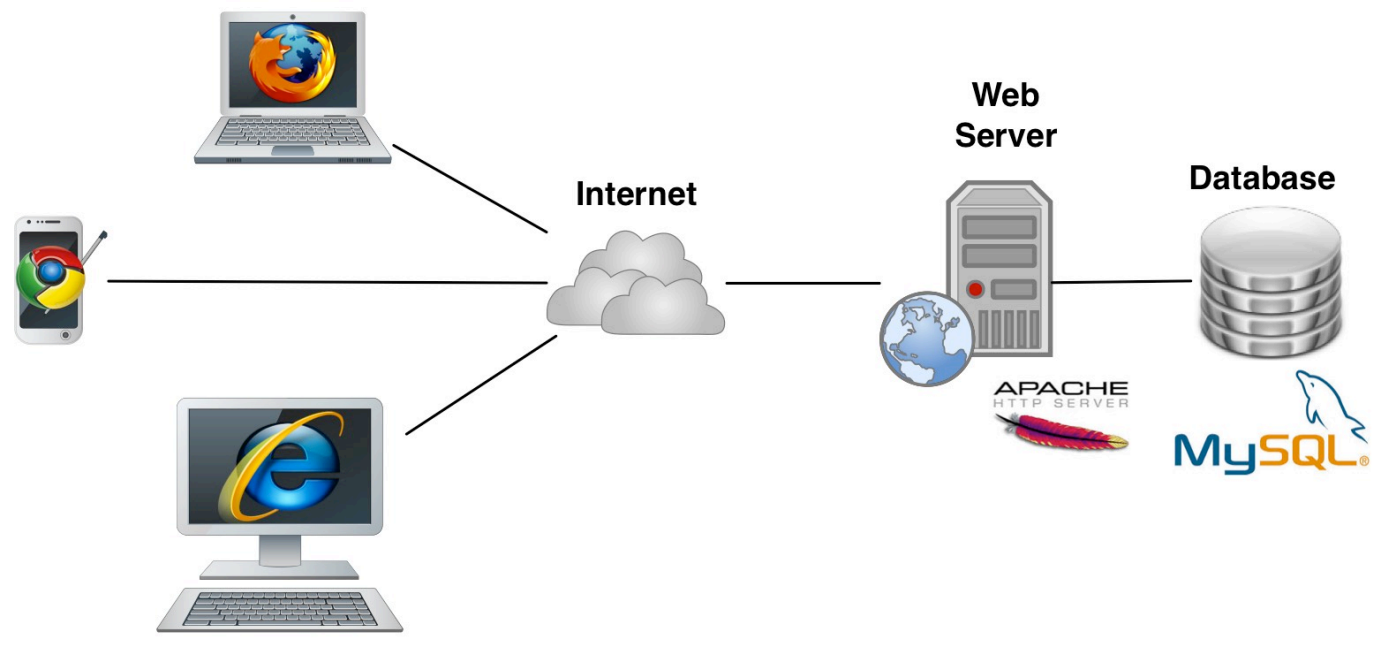
- c) dato un certo viaggio, consentire all'autista di valutare le caratteristiche dei passeggeri visualizzando l'elenco di coloro che lo hanno prenotato, con il voto medio dei feedback ricevuti da ciascun passeggero, presentando solo i passeggeri che hanno voto medio superiore ad un valore indicato dall'autista;

```
SELECT nome, cognome, mail, votomedio FROM (
SELECT nome, cognome, mail, AVG(voto) as votomedio
FROM VIAGGIO, Utente, prenotazione, feedback
WHERE  prenotazione.id_viaggio = '1'
and prenotazione.id_passeggero = feedback.id_destinatario
GROUP BY (feedback.id_destinatario) ) as vototable  where
votomedio > 3
```

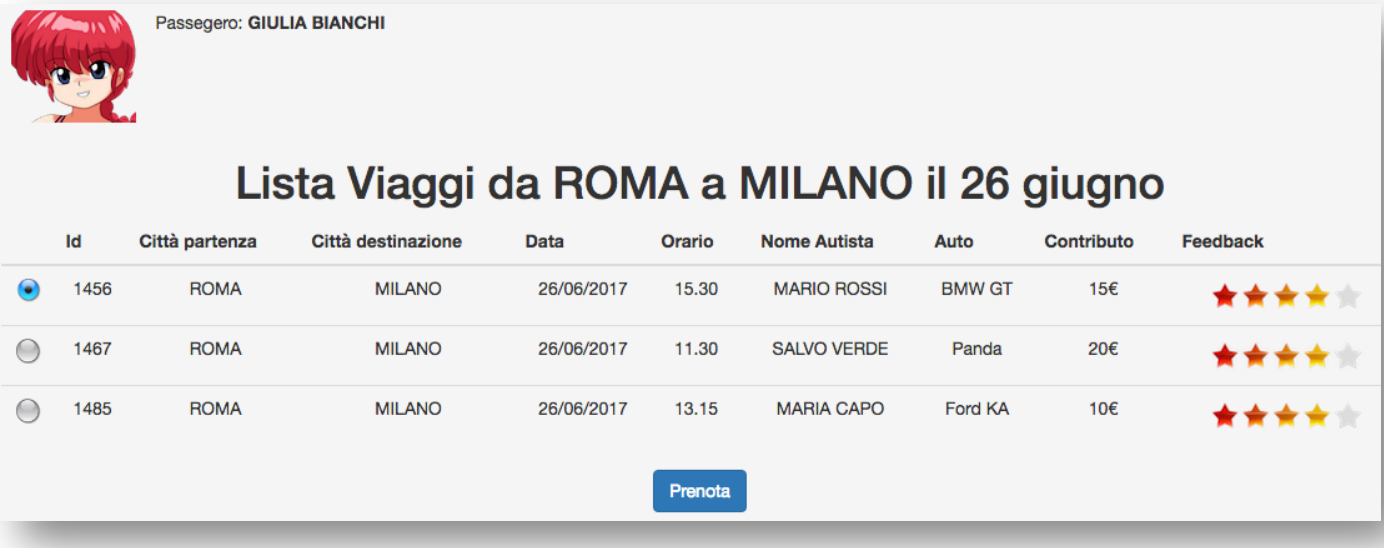
Quesito 4. il progetto di massima della struttura funzionale dell'applicazione Web, realizzando, con appropriati linguaggi a scelta sia lato client che lato server, un segmento significativo dell'applicazione che consente l'interazione con la base di dati.

Per il software si ipotizza di utilizzare una architettura classica:

- MySql come Database
- Apache come Web Server
- Php come linguaggio lato Server
- HTML, CSS, Java Script per implementare le pagine Web.



Inseriamo un piccolo pezzo di codice in cui utente passeggero, che già ha effettuato la login al sistema, visualizza una lista di viaggi e ne può selezionare uno poi effettuando la prenotazione:



Teniamo presente che i dati, per inserire la prenotazione, sono solo l'id dell'utente connesso, che si trova in sessione, e l'id del viaggio che sarà il valore del campo *input type="radio"* nella prima colonna della tabella lista viaggi.

```php
<?php

session_start();

if(isset($_POST['id_viaggio']))
    $id_viaggio =utf8_encode($_POST['id_viaggio']);

$id_passegero =utf8_encode($_SESSION['id_utente']);

$servername = "localhost";
$username = "root";
$password = "root";
$dbname = "carpooling";

// Creare la connessione
$conn = new mysqli($servername, $username, $password, $dbname);
// Check della connessione
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}
$sql = "INSERT INTO prenotazione('stato', 'id_viaggio', 'id_passeggero' )
VALUES ('in attesa','".$id_viaggio."','".id_passeggero."');";

if ($conn->query($sql) === TRUE) {
        echo "<br>Prenotazione inserita correttamente! " ;

} else {
    echo "Error: " . $sql . "<br>" . $conn->error;
}

$conn->close();
?>
```