Basi di Dati e Conoscenza

Progetto A.A. 2019/2020

PROGETTO DI UNA BASE DI DATI PER UN'AGENZIA DI VIAGGIO

0253606

Giovanni Pica

Indice

1. Descrizione del Minimondo

Un'agenzia di viaggio organizza viaggi organizzati della durata da un giorno ad una settimana in località italiane ed europee. I viaggi utilizzano autobus privati.
La compagnia intende realizzare un sistema informativo per la gestione dei viaggi e delle prenotazioni. La segreteria dell'agenzia crea nuovi itinerari di viaggio. Un itinerario è caratterizzato da una data di partenza, da una data di rientro, da un insieme di pernottamenti in località potenzialmente differenti e di durata differente e da un costo. Per ciascun pernottamento è di interesse tenere traccia di quale sia l'albergo scelto per ospitare i partecipanti. La scelta avviene all'interno di un insieme di alberghi mantenuto nel sistema informativo. Ciascun albergo è associato ad una capienza massima di persone, una città, un indirizzo, un insieme di recapiti (email, telefono, fax) e un referente. Quando un piano di viaggio viene creato, questo è definito dalle tappe che verranno coperte. Gli utenti del sistema possono prenotarsi al viaggio specificando il numero di persone per cui effettuano la prenotazione. Al termine della prenotazione viene fornito un codice che può essere utilizzato per disdire la prenotazione, fino a 20 giorni prima della partenza. Quando le prenotazioni sono concluse (ovverosia, 20 giorni prima della partenza), la segreteria è a conoscenza di quante persone parteciperanno al viaggio. Assegnano quindi al viaggio una guida, tra le guide che lavorano per l'agenzia. Assegnano inoltre un insieme di autobus per consentire il trasporto di tutte le persone. Un autobus può infatti avere una capienza differente. Ciascun autobus, inoltre, ha un costo giornaliero di utilizzo, che copre anche il costo dell'autista. Una volta definito il numero di autobus necessari, la segreteria, per ciascuna città toccata dal viaggio, associa un albergo che abbia una disponibilità massima di posti superiore alle persone che effettuano il viaggio. Ciascun hotel ha un costo per notte per persona che è conservato nel sistema. Per tutti i viaggi terminati, la segreteria può generare un report che mostri le informazioni sui partecipanti e il guadagno derivante da tale viaggio.

2. Analisi dei Requisiti

Identificazione dei termini ambigui e correzioni possibili

Linea	Termine	Nuovo termine	Motivo correzione
2	località	tappe	sinonimo di "tappe"
5	da un insieme di pernottamenti in località potenzialmente differenti e di durata differente e da un costo.	da un costo e da un insieme di pernottamenti potenzialmente differenti e di durata differente	si esplicita il riferimento all'itinerario
8	persone	partecipanti	sinonimo di “partecipanti”
11	piano di viaggio	itinerario	sinonimo di “itinerario”
12	persone	partecipanti	sinonimo di “partecipanti”
12	di quante persone parteciperanno	del numero di partecipanti	espressione più specifica
17	assegnano	la segreteria assegna	si esplicita il riferimento alla segreteria
18	persone	partecipanti	sinonimo di “persone”
21	disponibilità massima di posti superiore alle persone che effettuano il viaggio	capienza di posti maggiore al numero di partecipanti	espressione più specifica
22	persona	partecipante	sinonimo di “partecipante”
22	hotel	albergo	sinonimo di “albergo”
24	informazioni sui partecipanti	informazioni sul numero di partecipanti	si esplicita il riferimento al numero di partecipanti

Specifica disambiguata

Un'agenzia di viaggio organizza viaggi organizzati della durata da un giorno ad una settimana in tappe italiane ed europee. I viaggi utilizzano autobus privati. La compagnia intende realizzare un sistema informativo per la gestione dei viaggi e delle prenotazioni. La segreteria dell'agenzia crea nuovi itinerari di viaggio. Un itinerario è caratterizzato da una data di partenza, da una data di rientro, da un costo e da un insieme di pernottamenti in località potenzialmente differenti e di durata differente. Per ciascun pernottamento è di interesse tenere traccia di quale sia l'albergo scelto per

ospitare i partecipanti. La scelta avviene all'interno di un insieme di alberghi mantenuto nel sistema informativo. Ciascun albergo è associato ad una capienza massima di partecipanti, una città, un indirizzo, un insieme di recapiti (email, telefono, fax) e un referente. Quando un itinerario viene creato, questo è definito dalle tappe che verranno coperte. Gli utenti del sistema possono prenotarsi al viaggio specificando il numero di partecipanti per cui effettuano la prenotazione. Al termine della prenotazione viene fornito un codice che può essere utilizzato per disdire la prenotazione, fino a 20 giorni prima della partenza. Quando le prenotazioni sono concluse (ovverosia, 20 giorni prima della partenza), la segreteria è a conoscenza del numero di partecipanti al viaggio. La segreteria assegna quindi al viaggio una guida, tra le guide che lavorano per l'agenzia. La segreteria assegna inoltre un insieme di autobus per consentire il trasporto di tutti i partecipanti. Un autobus può infatti avere una capienza differente. Ciascun autobus,inoltre, ha un costo giornaliero di utilizzo, che copre anche il costo dell'autista. Una volta definito il numero di autobus necessari, la segreteria, per ciascuna città toccata dal viaggio,associa un albergo che abbia una capienza di posti maggiore al numero di partecipanti. Ciascun albergo ha un costo per notte per partecipante che è conservato nel sistema. Per tutti i viaggi terminati, la segreteria può generare un report che mostri le informazioni sul numero dei partecipanti e il guadagno derivante da tale viaggio.

Glossario dei Termini

Termine	Descrizione	Sinonimi	Collegamenti
Itinerario	Viaggio offerto.	Piano di viaggio	Prenotazione, Albergo, Guida, Autobus, Tappa
Albergo	Albergo dell'itinerario.	Hotel	Itinerario, Città, Tappa
Città	Città dove i partecipanti si fermano.		Albergo
Città Europea	Città in europa.		Tappa Europea
Città Italiana	Città situata in italia.		Tappa Italiana
Tappa	Fermata in una città.	Località	Albergo, Itinerario
Tappa Italiana	Fermata in una città situata in Italia.		Città Italiana
Tappa Europea	Feermata in una città in europa.		Città Europea
Prenotazione	Prenotazione al viaggio.		Itinerario
Autobus	Mezzo per il quale i partecipanti si muovono.		Itinerario
Guida	Guida dell'agenzia assegnata al viaggio.		Itinerario

Raggruppamento dei requisiti in insiemi omogenei

Frasi relative a Itinerario
Un itinerario è caratterizzato da una data di partenza, da una data di rientro, da un costo e da un insieme di pernottamenti in località potenzialmente differenti e di durata differente. Quando un itinerario viene creato, questo è definito dalle tappe che verranno coperte.

Frasi relative a Albergo
Ciascun albergo è associato ad una capienza massima di partecipanti, una città, un indirizzo, un insieme di recapiti (email, telefono, fax) e un referente. Una volta definito il numero di autobus necessari, la segreteria, per ciascuna città toccata dal viaggio,associa un albergo che abbia una capienza di posti maggiore al numero di partecipanti. Ciascun albergo ha un costo per notte per partecipante che è conservato nel sistema.

Frasi relative a Città
Una volta definito il numero di autobus necessari, la segreteria, per ciascuna città toccata dal viaggio,associa un albergo che abbia una capienza di posti maggiore al numero di partecipanti.

Frasi relative a Tappa
Quando un itinerario viene creato, questo è definito dalle tappe che verranno coperte.

Frasi relative a Tappa Italiana ed Europea
Un'agenzia di viaggio organizza viaggi organizzati della durata da un giorno ad una settimana in tappe italiane ed europee.

Frasi relative a Prenotazione
Al termine della prenotazione viene fornito un codice che può essere utilizzato per disdire la prenotazione, fino a 20 giorni prima della partenza. Quando le prenotazioni sono concluse (ovverosia, 20 giorni prima della partenza), la segreteria è a conoscenza del numero di partecipanti al viaggio.

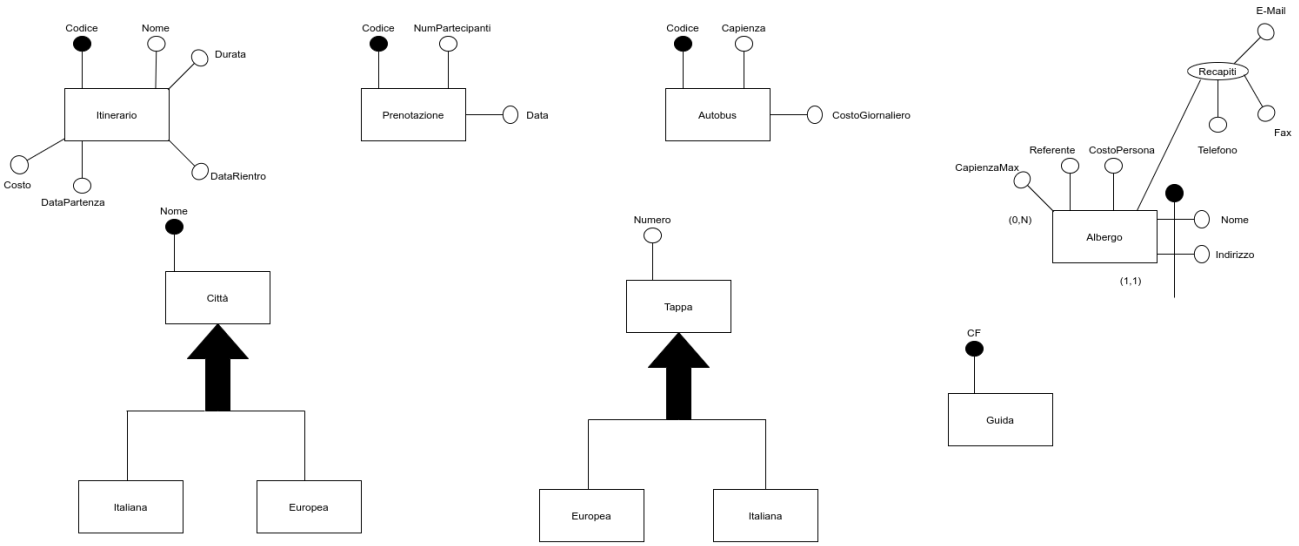
Frasi relative a Autobus
La segreteria assegna inoltre un insieme di autobus per consentire il trasporto di tutti i partecipanti. Un autobus può infatti avere una capienza differente. Ciascun autobus,inoltre, ha un costo giornaliero di utilizzo, che copre anche il costo dell'autista.

Frasi relative a Guida
La segreteria assegna quindi al viaggio una guida, tra le guide che lavorano per l'agenzia.

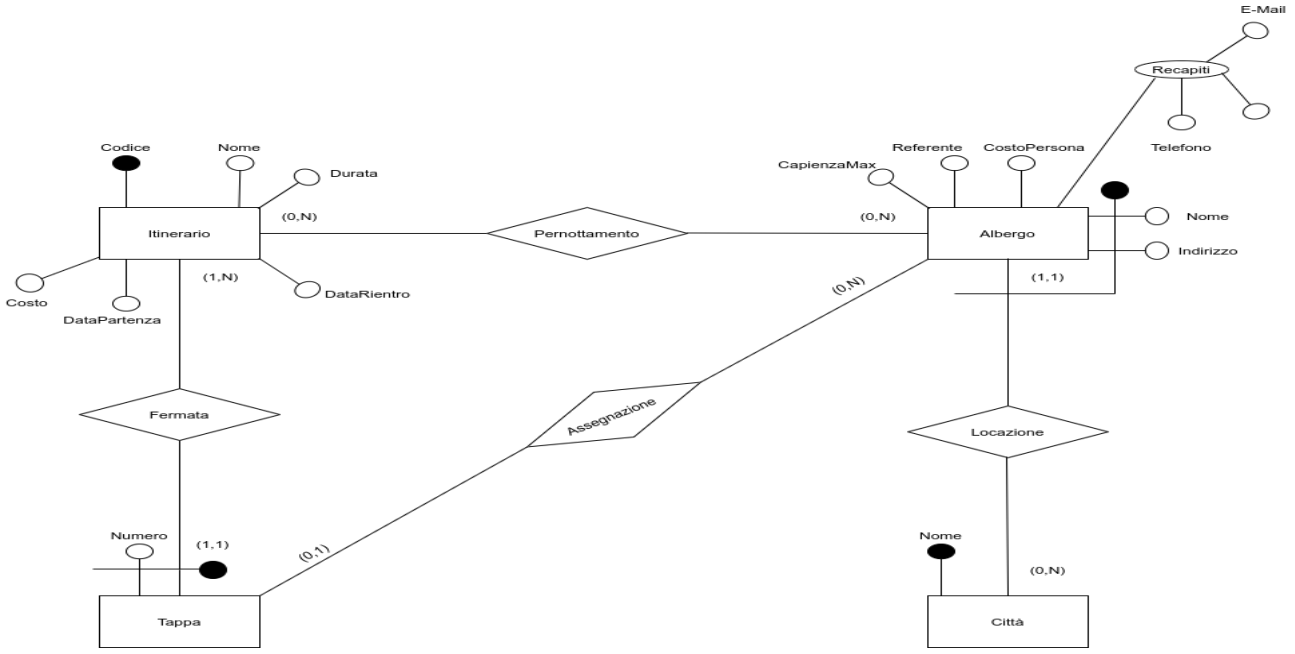
3. Progettazione concettuale

Costruzione dello schema E-R

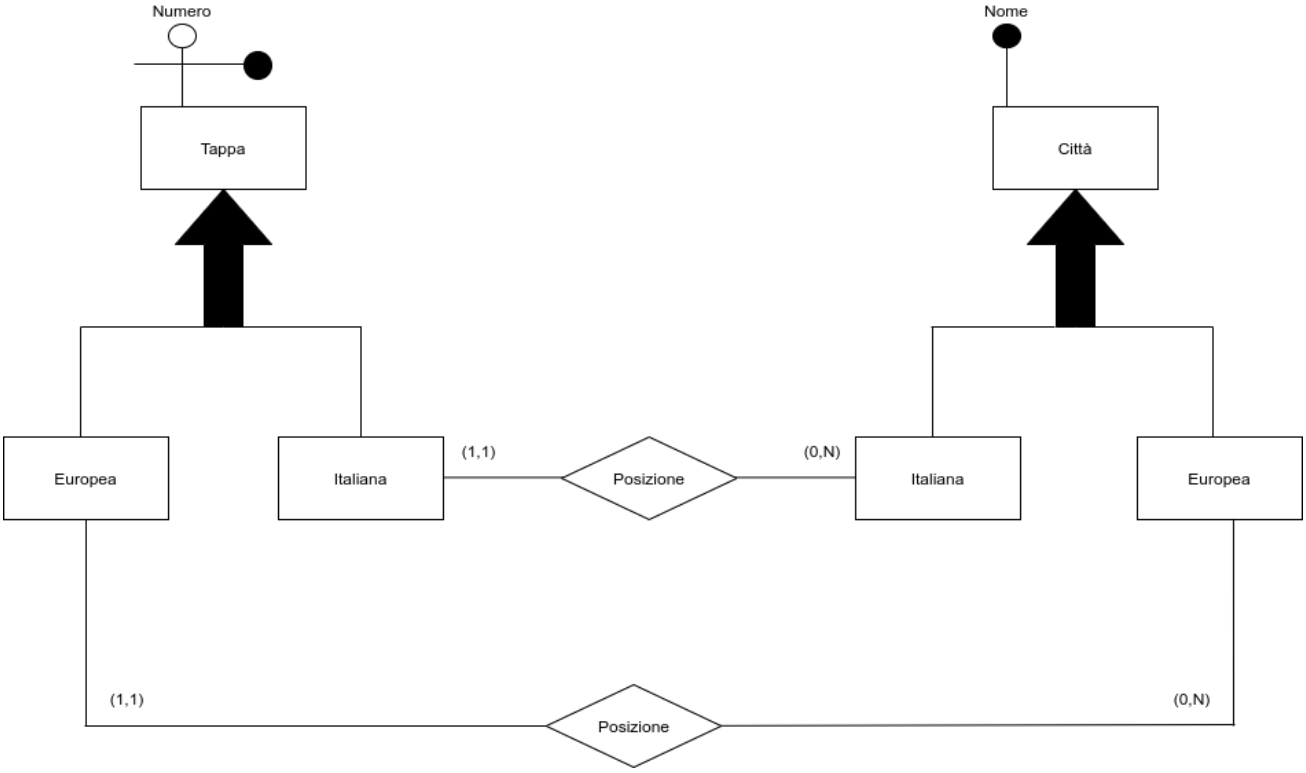
La strategia che è stata utilizzata è la botttom-up, con Tappa e Città che sono state generalizzate in Europea e Italiana, in quanto il viaggio può toccare uno dei due tipi di località. Inoltre Albergo avrà identificatore esterno con Città, mentre Tappa con Itinerario.



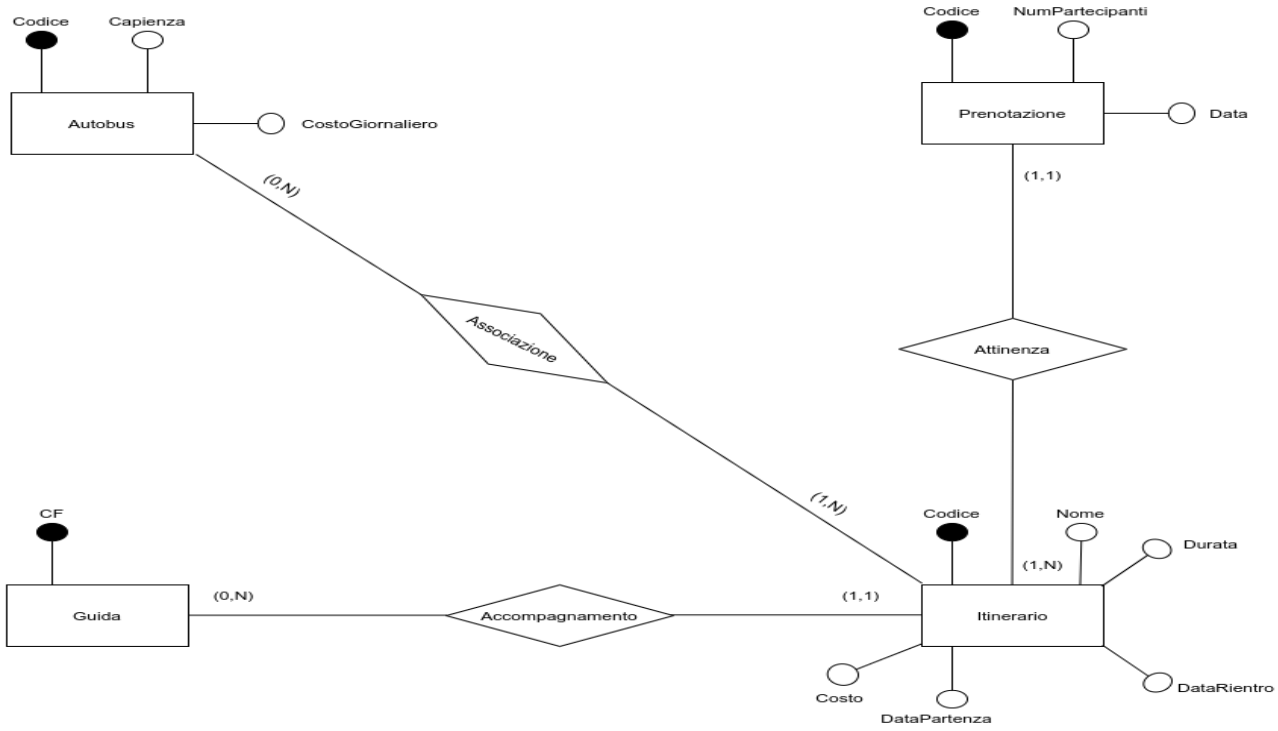
Si sono collegati i concetti di Itinerario con Albergo, che a sua voltà è collegato con identificatore esterno con Città, inoltre si è collegato Tappa con Albergo e con Itinerario con anche qui identificatore esterno.



Si sono collegati i concetti di Tappa Italiana con Città Italiana e Tappa Europea con Città Europea, per disambiguare il fatto in cui si abbia un collegamento tra una tappa europea ed una città europea, infatti la specifica vuole che si separino questi due concetti di Europea e Italiana.

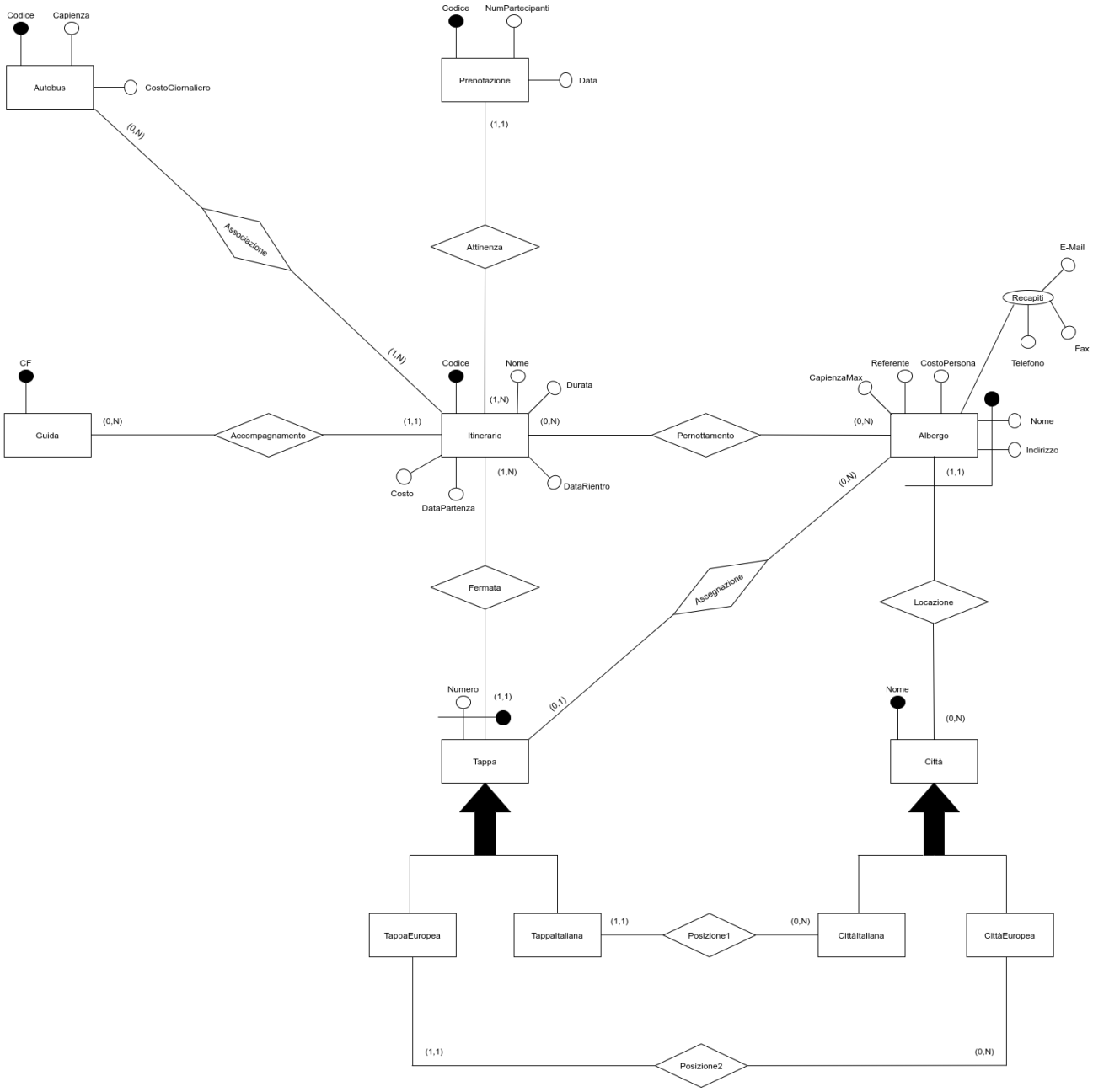


Si inseriscono gli ultimi concetti mancanti:



Integrazione finale

Per evitare disambiguità sono state specificate TAPPAEUROPEA/ITALIANA CITTÀEUROPEA/ITALIANA e POSIZIONE1 POSIZIONE2.



Regole aziendali

RD1) La cancellazione della prenotazione **SI OTTIENE** con un codice fornito al termine della prenotazione stessa.

RV1) Il numero di giorni prima della partenza per disdire la prenotazione **DEVE ESSERE** inferiore a 20.

RV2) Un albergo **DEVE** avere la capienza di posti maggiore al numero di partecipanti

RV3) La durata di un piano di viaggio **DEVE ESSERE** inferiore a 7 giorni.

Dizionario dei dati

Entità	Descrizione	Attributi	Identificatori
Itinerario	L'itinerario del viaggio che si andrà a svolgere.	Codice, Costo,Nome, Durata, DataPartenza, DataRientro	Codice
Albergo	L'albergo che ospiterà i partecipanti.	Nome, Indirizzo, CostoPersona, CapienzaMax, Referente, Recapiti	Nome + Indirizzo + Città (id. esterno)
Città	La città che verrà toccata dal viaggio.	Nome	Nome
CittàEuropea	Città toccata dal viaggio situata in Europa.	eredita da Città	eredita da Città
CittàItaliana	Città toccata dal viaggio situata in Italia.	eredita da Città	eredita da Città
Tappa	La specifica fermata.	Numero	Numero + Itinerario (id. esterno)
TappaEuropea	Fermata in Europa.	eredita da Tappa	eredita da Tappa
TappaItaliana	Fermata in Italia.	eredita da Tappa	eredita da Tappa
Prenotazione	L'effettiva prenotazione al viaggio.	Codice, NumPartecipanti	Codice
Autobus	L'autobus utilizzato per muoversi.	Codice, Capienza, CostoGiornaliero	Codice
Guida	La guida messa a disposizione dall'agenzia.	CF	CF

4. Progettazione logica

Volume dei dati

Concetto nello schema	Tipo[1]	Volume atteso
Itinerario	E	100
Albergo	E	50
Città	E	100
CittàEuropea	E	50
CittàItaliana	E	50
Tappa	E	200
TappaEuropea	E	100
TappaItaliana	E	100
Prenotazione	E	5000
Autobus	E	150
Guida	E	30
Attinenza	R	5000
Associazione	R	300
Accompagnamento	R	100
Pernottamento	R	300
Fermata	R	200
Assegnazione	R	200
Locazione	R	50
Posizione1	R	100
Posizione2	R	100

Tavola delle operazioni

Cod.	Descrizione	Frequenza attesa
OP1	Inserisci nuovo itinerario	5/giorno
OP2	Assegna albergo ad un itinerario.	5/giorno
OP3	Inserisci un nuovo albergo indicando tutti i suoi dati.	2/giorno
OP4	Inserisci una nuova prenotazione indicando tutti i suoi dati e a quale itinerario è assegnata.	50/giorno
OP5	Cancellare prenotazione.	20/giorno
OP6	Inserisci una nuova guida indicando tutti i suoi dati.	1/giorno
OP7	Assegna autobus ad un itinerario.	10/giorno
OP8	Inserisci un nuovo autobus indicando tutti i suoi dati.	5/giorno
OP9	Inserisci una nuova tappa europea specificando tutti i suoi dati.	5/giorno

1 Indicare con E le entità, con R le relazioni

OP10	Inserisci una nuova tappa italiana specificando tutti i suoi dati.	5/giorno
OP11	Assegna tappa ad un albergo.	5/giorno
OP12	Inserisci una nuova città europea specificando tutti i suoi dati.	2/giorno
OP13	Inserisci una nuova città italiana specificando tutti i suoi dati.	2/giorno
OP14	Stampa tutti gli itinerari offerti dall'agenzia.	20/giorno
OP15	Stampa tutti gli itinerari che sono stati assegnati ad una determinata guida.	2/giorno
OP16	Stampa le tappe che verranno coperte dal viaggio.	15/giorno
OP17	Generare un report con informazioni sui partecipanti e il guadagno derivante dal viaggio.	10/mese

Costo delle operazioni

OP1) Abbiamo un accesso in scrittura in ITINERARIO, uno in scrittura in ACCOMPAGNAMENTO ed uno in lettura in GUIDA. Dunque avremo:

(2+2+1)*5 = 25 accessi giornalieri

OP2) Abbiamo un accesso in lettura in ALBERGO, uno in scrittura in PERNOTTAMENTO ed uno in scrittura in ITINERARIO. Dunque avremo:

(1+2+2)*5 = 25 accessi giornalieri

OP3) Abbiamo un accesso in scrittura in ALBERGO, uno in scrittura in LOCAZIONE ed uno in lettura in CITTÀ.Dunque avremo:

(2+2+1)*2 = 10 accessi giornalieri

OP4) Abbiamo un accesso in scrittura in PRENOTAZIONE, uno in scrittura in ATTINENZA, uno in scrittura in ITINERARIO. Dunque avremo:

(2+2+2)*50 = 300 accessi giornalieri

OP5) Abbiamo un accesso in scrittura in ITINERARIO, uno in scrittura in ATTINENZA ed uno in scrittura in PRENOTAZIONE. Dunque avremo:

(2+2+2)*50 = 300 accessi giornalieri

OP6) Abbiamo un accesso in scrittura in GUIDA. Dunque avremo:

2*1 = 2 accessi giornalieri

OP7) Abbiamo un accesso in lettura in AUTOBUS, uno in scrittura in ASSOCIAZIONE, uno in scrittura in ITINERARIO. Dunque avremo:

(1+2+2)*10 = 50 accessi giornalieri

OP8) Abbiamo un accesso in scrittura in AUTOBUS. Dunque avremo:

2*5 = 10 accessi giornalieri

OP9) Abbiamo un accesso in scrittura in TAPPA ed un altro in scrittura nell'entita figlia TAPPAEUROPEA. Dunque:

(2+2)*5 = 20 accessi giornalieri

OP10) Abbiamo un accesso in scrittura in TAPPA ed un altro in scrittura nell'entità figlia TAPPAITALIANA. Dunque:

(2+2)*5 = 20 accessi giornalieri

OP11) Abbiamo un accesso in lettura in TAPPA, uno in scrittura in ASSEGNAZIONE ed uno in scrittura in ALBERGO. Dunque avremo:

(1+2+2)*5 = 25 accessi giornalieri

OP12) Abbiamo un accesso in scrittura in CITTÀ ed un altro nell'entità figlia CITTÀEUROPEA. Dunque avremo:

(2+2)*2 = 8 accessi giornalieri

OP13) Abbiamo un accesso in scrittura in CITTÀ ed un altro nell'entità figlia CITTÀITALIANA. Dunque avremo:

(2+2)*2 = 8 accessi giornalieri

OP14) Abbiamo 100 accessi in lettura in ITINERARIO. Dunque avremo:

100*20 = 2000 accessi giornalieri

OP15) Abbiamo un accesso in lettura in GUIDA, uno in lettura in ACCOMPAGNAMENTO ed uno in lettura in ITINERARIO. Dunque avremo:

(1+1+1)*2 = 6 accessi giornalieri

OP16) Abbiamo un accesso in lettura in ITINERARIO, M accessi in lettura in FERMATA ed M accessi in lettura in TAPPA, dove M è il numero medio di tappe di un itinerario. Supponendo M = 2 avremo:

(1+2+2)*15 = 75 accessi giornalieri

OP17) Abbiamo un accesso in lettura in ITINERARIO, uno in lettura in ATTINENZA, uno in lettura in PRENOTAZIONE, uno in lettura in ASSOCIAZIONE, uno in lettura in AUTOBUS, uno in lettura in PERNOTTAMENTO, uno in lettura in ALBERGO. Dunque avremo:

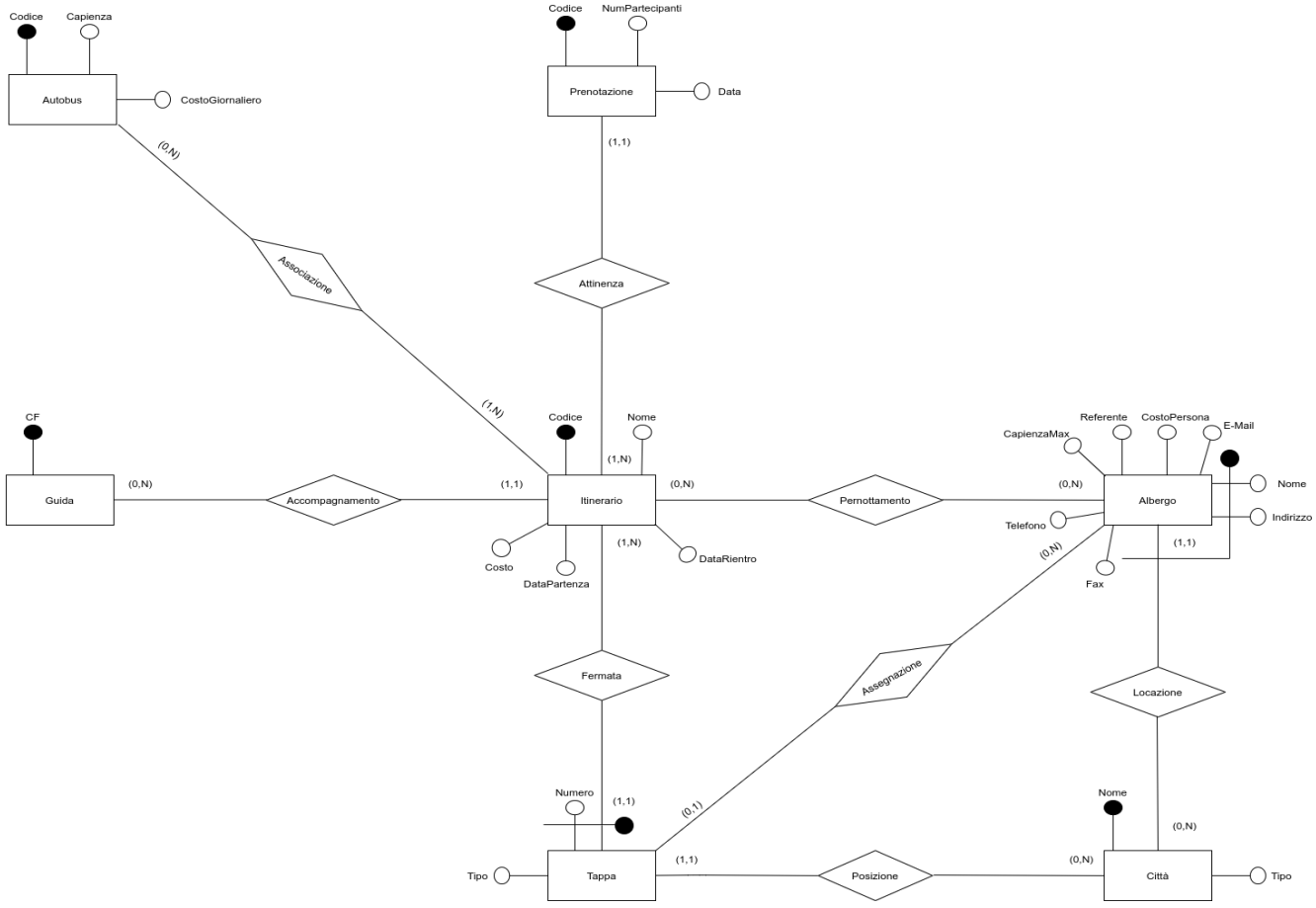
(1+1+1+1+1+1+1)*10 = 70 accessi mensili

Ristrutturazione dello schema E-R

1) Analisi Delle Ridondanze: L'unico elemento ridondante nello schema è Durata in ITINERARIO, in quanto può essere ricavato tramite una differenza tra DataRientro e DataPartenza. Dunque si è tolto in fase di ristrutturazione tale attributo.

2) Eliminazione delle generalizzazioni: Abbiamo due generalizzazioni, una in CITTÀ ed un'altra in TAPPA, si è deciso di eliminarle entrambe ed aggiungere un attributo Tipo che serve a capire se quella TAPPA/CITTÀ è Europea o Italiana. Inoltre TAPPA ora avra una relazione uno a molti (con molti opzionale) con CITTÀ. Facendo ciò diminuiscono le operazioni 9,10,12,13 che avranno un consumo giornaliero ridotto del 50%. Per fare in modo che una tappa italiana fosse collegata ad una città italiana (o tappa europea con città europea) è stato implementato un trigger che controlla questo attributo Tipo che andremo a vedere nella progettazione fisica.

3) Scelta degli Identificatori Principali: In Albergo si hanno due soluzioni o si sceglie come identificatore la coppia Nome, Indirizzo o Nome, Indirizzo e l' id. esterno con CITTÀ, è stata scelta questa seconda opzione in quanto non si rischia un caso estremo in cui si hanno due alberghi con stesso nome stesso indirizzo ma in città diverse, può infatti capitare che due nazioni abbiano degli stessi indirizzi. Inoltre TAPPA avrà come chiave la coppia Numero, CodItinerario.



Trasformazione di attributi e identificatori

Guardando l'E-R si nota che ci sono 3 attributi ripetuti sotto il nome di Codice delle entità AUTOBUS, PRENOTAZIONE, ITINERARIO. Per evitare ciò sono stati richiamati questi attributi rispettivamente con CodAutobus, CodPrenotazione, CodItinerario, quando avremo dei vincoli di integrità referenziale. Abbiamo inoltre un attributo composto in ALBERGO che andremo a scomporre. Mentre riguardo agli identificatori esterni, ne abbiamo uno in TAPPA ed uno in ALBERGO che sono quindi CodItinerario e NomeCittà, CittàAlbergo.

Traduzione di entità e associazioni

Entità:

PRENOTAZIONE(Codice, NumPartecipanti, CodItinerario) ;

ITINERARIO(Codice, Costo, Nome, DataRientro, DataPartenza, CFGuida) ;

AUTOBUS(Codice, Capienza, CostoGiornaliero) ;

GUIDA(CF) ;

ALBERGO(Indirizzo, Nome, NomeCittà, CostoPersona, Referente, CapienzaMax, E-Mail, Telefono, Fax) ;

CITTÀ(Nome, Tipo) ;

TAPPA(Numero, CodItinerario, NomeCittà, Tipo) ;

Associazioni:

ASSOCIAZIONE(CodItinerario, CodAutobus) ;

ASSEGNAZIONE(NumeroTappa, CodItinerario, NomeAlbergo, IndirizzoAlbergo, CittàAlbergo);

PERNOTTAMENTO(CodItinerario, IndirizzoAlbergo, NomeAlbergo, CittàAlbergo) ;

Con vincoli di integrità referenziale: CodItinerario di PRENOTAZIONE con Codice di ITINERARIO, CFGuida di ITINERARIO con CF di GUIDA, NomeCittà di ALBERGO con Nome di CITTÀ, CodItinerario di TAPPA con Codice di ITINERARIO, NomeCittà di TAPPA con Nome di CITTÀ, IndirizzoAlbergo, NomeAlbergo di TAPPA con ALBERGO, CodItinerario di ASSOCIAZIONE con Codice di ITINERARIO, CodAutobus di ASSOCIAZIONE con Codice di AUTOBUS, NumeroTappa, CodItinerario di ASSEGNAZIONE con Numero, CodItinerario di TAPPA, NomeAlbergo, IndirizzoAlbergo, CittàAlbergo di ASSEGNAZIONE con Nome,Indirizzo,NomeCittà di ALBERGO, CodItineario di PERNOTTAMENTO con Codice di ITINERARIO, IndirizzoAlbergo, NomeAlbergo, NomeCittà di PERNOTTAMENTO con Indirizzo, Nome, NomeCittà di ALBERGO.

Normalizzazione del modello relazionale

1NF) L'unica tabella che non è in 1NF è quella inerente ad ALBERGO, in quanto presenta un attributo composto Recapiti, ma come già detto nel paragrafo precedente, esso è stato scomposto in più "colonne", quindi non si ha bisogno di nessun processo di normalizzazione.

2NF) Bisogna controllare le dipendenze funzionali:

PRENOTAZIONE: Codice -> NumPartecipanti, CodItineario è in 2NF;

ITINERARIO: Codice -> Durata, Costo, Nome, DataRientro, DataPartenza, CFGuida è in 2NF;

AUTOBUS: Codice -> Capienza, CostoGiornaliero è in 2NF;

GUIDA ha solo CF quindi va bene così;

ALBERGO: Indirizzo, Nome, NomeCittà -> CostoPersona, Referente, CapienzaMax, E-Mail, Telefono, Fax, è in 2NF;

CITTÀ: Nome-> Tipo in 2NF;

TAPPA: Numero, CodItinerario -> NomeCittà, Tipo

Stesso criterio per ASSOCIAZIONE, PERNOTTAMENTO;

ASSEGNAZIONE: NumeroTappa, CodItinerario -> IndirizzoAlbergo, NomeAlbergo, CittàAlbergo

3NF) Controllando le dipendenze funzionali si nota come tutte le tabelle siano gia in 3NF in quanto non si hanno attributi non chiave che dipendono da altri attributi non chiave.

5. Progettazione fisica

Utenti e privilegi

Gli utenti che sono stati utilizzati all'interno dell'applicazione sono 4:

1. agenzia: è l'agenzia di viaggi che si occuperà di inserire nuovi itinerari, alberghi, tappe, autobus e guide e li associerà ai relativi itinerari, oppure di associare tappe ad alberghi sempre se la fermata lo richiede. Inoltre questo utente è colui che genera il report di tutte le informazioni inerenti ad un determinato viaggio, contenente numero di partecipanti e il guadagno ed inoltre si comporta da admin in quanto permette la creazione di un nuovo utente a seconda del ruolo che esso occupa.
2. guida: è la guida associata all'itinerario e questo avrà come privilegio quello di stampare su schermo tutti gli itinerari a cui esso è stato associato;
3. cliente: è il cliente che si prenoterà al viaggio scelto, o cancellerà questa prenotazione, inoltre potrà stampare gli itinerari disponibili e le tappe che un determinato viaggio tocca;
4. login: è colui che si occupa di far accedere all'applicazione gli altri utenti gia citati e dovrà distinguere i vari utenti a seconda del loro ruolo.

Strutture di memorizzazione

Tabella <Albergo>		
Attributo	**Tipo di dato**	**Attributi**[2]
Indirizzo	VARCHAR(45)	PK,NN
Nome	VARCHAR(45)	PK, NN
CostoPersona	INT	NN
CapienzaMax	INT	NN
Referente	VARCHAR(45)	NN
E-Mail	VARCHAR(45)	NN
Telefono	VARCHAR(45)	NN
Fax	VARCHAR(45)	NN
NomeCittà	VARCHAR(45)	PK,NN
Tabella <Associazione>		
Attributo	**Tipo di dato**	**Attributi**
CodItinerario	INT	PK,NN
CodAutobus	INT	PK,NN
Tabella <Assegnazione>		
Attributo	**Tipo di dato**	**Attributi**
CodItinerario	INT	PK,NN
NumeroTappa	INT	PK,NN
NomeAlbergo	VARCHAR(45)	NN
IndirizzoAlbergo	VARCHAR(45)	NN
CittàAlbergo	VARCHAR(45)	NN
Tabella <Autobus>		
Attributo	**Tipo di dato**	**Attributi**
Codice	INT	PK,NN,AI
Capienza	INT	NN
CostoGiornaliero	INT	NN
Tabella <Città>		
Attributo	**Tipo di dato**	**Attributi**
Nome	VARCHAR(45)	PK,NN
Tipo	VARCHAR(45)	NN
Tabella <Guida>		
Attributo	**Tipo di dato**	**Attributi**
CF	VARCHAR(45)	PK,NN
Tabella <Itinerario>		
Attributo	**Tipo di dato**	**Attributi**
Codice	INT	PK,NN,AI
Costo	INT	NN
DataPartenza	DATE	NN

2 PK = primary key, NN = not null, UQ = unique, UN = unsigned, AI = auto increment. È ovviamente possibile specificare più di un attributo per ciascuna colonna.

DataRientro	DATE	NN
CFGuida	VARCHAR(45)	NN
Nome	VARCHAR(100)	NN
Tabella <Pernottamento>		
Attributo	**Tipo di dato**	**Attributi**
CodItinerario	INT	PK,NN
IndirizzoAlbergo	VARCHAR(45)	PK,NN
NomeAlbergo	VARCHAR(45)	PK,NN
NomeCittà	VARCHAR(45)	PK,NN
Tabella <Prenotazione>		
Attributo	**Tipo di dato**	**Attributi**
Codice	INT	PK,NN,AI
NumPartecipanti	INT	NN
CodItineario	INT	NN
Tabella <Tappa>		
Attributo	**Tipo di dato**	**Attributi**
Numero	INT	PK,NN,AI
CodItinerario	INT	PK,NN
NomeCittà	VARCHAR(45)	NN
Tipo	VARCHAR(45)	NN
Tabella <utenti>		
Attributo	**Tipo di dato**	**Attributi**
username	VARCHAR(45)	PK,NN
password	CHAR(32)	NN
ruolo	ENUM('agenzia', 'guida', 'cliente')	NN

Indici

Tabella <Albergo>	
Indice <PRIMARY>	**Tipo[3]: PR**
Colonna 1	<Indirizzo>
Colonna 2	<Nome>
Colonna 3	<NomeCittà>
Tabella <Albergo>	
Indice <fk_Albergo_Città_idx>	**Tipo: IDX**
Colonna 1	<NomeCittà>
Tabella <Associazione>	
Indice <PRIMARY>	**Tipo: PR**
Colonna 1	<CodItinerario>
Colonna 2	<CodAutobus>
Tabella <Associazione>	
Indice <fk_Associazione_Autobus_idx>	**Tipo: IDX**
Colonna 1	<CodAutobus>
Tabella <Associazione>	
Indice <fk_Associazione_Itinerario_idx>	**Tipo: IDX**
Colonna 1	<CodItinerario>

3 IDX = index, UQ = unique, FT = full text, PR = primary.

Tabella <Assegnazione>	
Indice <PRIMARY>	**Tipo: PR**
Colonna 1	<NumeroTappa>
Colonna 2	<CodItinerario>
Tabella <Assegnazione>	
Indice <fk_Assegnazione_Albergo1_idx>	**Tipo: IDX**
Colonna 1	<NomeAlbergo>
Colonna 2	<IndirizzoAlbergo>
Colonna 3	<CittàAlbergo>
Tabella <Autobus>	
Indice <PRIMARY>	**Tipo: PR**
Colonna 1	<CodAutobus>
Tabella <Città>	
Indice <PRIMARY>	**Tipo: PR**
Colonna 1	<NomeCittà>
Tabella <Guida>	
Indice <PRIMARY>	**Tipo: PR**
Colonna 1	<CF>
Tabella <Itinerario>	
Indice <PRIMARY>	**Tipo: PR**
Colonna 1	<Codice>
Tabella <Itinerario>	
Indice <fk_Itinerario_Guida_idx>	**Tipo: IDX**
Colonna 1	<CFGuida>
Tabella <Pernottamento>	
Indice <PRIMARY>	**Tipo: PR**
Colonna 1	<IndirizzoAlbergo>
Colonna 2	<NomeCittà>
Colonna 3	<NomeAlbergo>
Colonna 4	<CodItinerario>
Tabella <Pernottamento>	
Indice <fk_Itinerario_has_Albergo_Albergo1_idx>	**Tipo: IDX**
Colonna 1	<IndirizzoAlbergo>
Colonna 2	<NomeCittà>
Colonna 3	<NomeAlbergo>
Tabella <Pernottamento>	
Indice <fk_Itinerario_has_Albergo_Itinerario1_idx>	**Tipo: IDX**
Colonna 1	<CodItinerario>
Tabella <Prenotazione>	
Indice <PRIMARY>	**Tipo: PR**
Colonna 1	<Codice>
Tabella <Prenotazione>	
Indice <fk_Prenotazione_Itinerario_idx>	**Tipo: IDX**
Colonna 1	<CodItinerario>
Tabella <Tappa>	
Indice <PRIMARY>	**Tipo: PR**

Colonna 1	<Numero>
Colonna 2	<CodItinerario>
Tabella <Tappa>	
Indice <fk_Tappa_Itinerario>	**Tipo: IDX**
Colonna 1	<CodItinerario>
Tabella <Tappa>	
Indice <fk_Tappa_Città>	**Tipo: IDX**
Colonna 1	<NomeCittà>
Tabella <utenti>	
Indice <PRIMARY>	**Tipo: PR**
Colonna 1	<username>

Trigger

Riguardo ad ALBERGO, è stato implementato questo trigger before insert, che controlla se si inserisce un valore 0 o negativo per quanto riguarda il costo per persona (viaggio gratis) o riguardante la capienza massima (che non può essere di certo 0 o minore addirittura).

```
CREATE
DEFINER=`root`@`localhost`
TRIGGER `mydb`.`Albergo_BEFORE_INSERT`
BEFORE INSERT ON `mydb`.`Albergo`
FOR EACH ROW
BEGIN
    if NEW.CostoPersona <= 0 then
           signal sqlstate "45000" set message_text = "The cost of the
Hotel must be greater than 0";
    end if;
    if NEW.CapienzaMax <= 0 then
           signal sqlstate "45001" set message_text = "The capacity of
the hotel must be greater than 0";
     end if;
END
```

Riguardo ad AUTOBUS, per lo stesso motivo di sopra è stato inserito un trigger per controllare il costo giornaliero ed inoltre si è fissato un valore minimo e massimo per la capienza dai 40 ai 60 posti.

```
CREATE
```

```
DEFINER=`root`@`localhost`
TRIGGER `mydb`.`Autobus_BEFORE_INSERT`
BEFORE INSERT ON `mydb`.`Autobus`
FOR EACH ROW
BEGIN
	if NEW.Capienza < 40 then
		signal sqlstate "45000" set message_text = "At least 40 seats";
	end if;
	if NEW.Capienza > 60 then
		signal sqlstate "45001" set message_text = "At most 60 seats";
	end if;
	if NEW.CostoGiornaliero <= 0 then
		signal sqlstate "45002" set message_text = "The cost of the autobus must be greater than 0";
	end if;
END
```

Riguardo a CITTÀ, è stato implementato un trigger che verifica che non si inserisca una qualche stringa diversa da 'Italian' e 'European'.

```
CREATE DEFINER = CURRENT_USER TRIGGER `mydb`.`Città_BEFORE_INSERT` BEFORE INSERT ON `Città` FOR EACH ROW
BEGIN
	if (NEW.`Tipo` <> 'Italian' and NEW.`Tipo` <> 'European') then
		signal sqlstate "45001" set message_text = "You must insert Italian or European";
	end if;
END
```

Riguardo ad ITINERARIO, è stato implementato un trigger sempre per il costo, inoltre esso controlla se si inserisce una data di rientro che è qualche giorno prima di quella di partenza (un controsenso), controlla che

non si inserisca un itinerario che ha la data di partenza che è gia passata (esempio oggi è 1 gennaio, inserisco un itinerario con data di partenza 28 dicembre) ed infine controlla che la durata del viaggio sia di massimo 7 giorni.

```
CREATE
DEFINER=`root`@`localhost`
TRIGGER `mydb`.`Itinerario_BEFORE_INSERT`
BEFORE INSERT ON `mydb`.`Itinerario`
FOR EACH ROW
BEGIN
      if NEW.Costo <= 0 then
            signal sqlstate "45000" set message_text = "The cost of the Itinerary must be greater than 0";
      end if;

      if datediff(NEW.DataRientro, NEW.DataPartenza) < 0 then
            signal sqlstate "45001" set message_text = "The return date may not be earlier than the departure date!";
      end if;

      if datediff(NEW.DataPartenza, now()) < 0 then
            signal sqlstate "45002" set message_text = "It's too late to programming a trip!!";
      end if;

      if datediff(NEW.DataRientro, NEW.DataPartenza) > 7 then
            signal sqlstate "45003" set message_text = "The maximum number of days is 7";
      end if;
END
```

In PERNOTTAMENTO sempre before insert, è stato implementato un trigger che controlla se il numero di partecipanti sia maggiore della capienza, in quanto un ALBERGO non può ospitare un certo numero di partecipanti se non ha la capienza necessaria.

```
CREATE DEFINER = CURRENT_USER TRIGGER
`mydb`.`Pernottamento_BEFORE_INSERT` BEFORE INSERT ON `Pernottamento` FOR
EACH ROW
BEGIN
     declare var_persone int;
     declare var_capienza int;
     select `PartecipantiComplessivi` from `Partecipanti` where
`IdItinerario` = NEW.`CodItinerario` into var_persone;
     select `CapienzaMax` from `Albergo` where `NomeCittà` =
NEW.`CittàAlbergo` and `Indirizzo` = NEW.`IndirizzoAlbergo` and `Nome` =
NEW.`NomeAlbergo` into var_capienza;
     if (var_persone > var_capienza) then
          signal sqlstate "45000" set message_text = "The hotel is too
small!";
      end if;
END
```

In PRENOTAZIONE, sempre before insert, è stato implementato un trigger che controlla se si inserisce un numero di partecipanti inferiore o uguale a 0, inoltre si controlla se non si sta prenotando ad un itinerario ormai passato. Inoltre è stato implementato un trigger before delete che controlla che non si stia cancellando una prenotazione dopo il limite dei 20 giorni di tempo.

```
CREATE
DEFINER=`root`@`localhost`
TRIGGER `mydb`.`Prenotazione_BEFORE_INSERT`
BEFORE INSERT ON `mydb`.`Prenotazione`
FOR EACH ROW
BEGIN
     declare var_data date;
```

```
	select `DataPartenza` from `Itinerario` where `Codice` =
NEW.`CodItinerario` into var_data;
	if var_data <= now() then
		signal sqlstate "45002" set message_text = "It's too late to
register at this itinerary!";
	end if;
	if NEW.NumPartecipanti <= 0 then
		signal sqlstate "45000" set message_text = "The number of
participants must be greater than 0";
	end if;
END
CREATE DEFINER = CURRENT_USER TRIGGER `mydb`.`Prenotazione_BEFORE_DELETE`
BEFORE DELETE ON `Prenotazione` FOR EACH ROW
BEGIN
	declare var_data date;
	select `DataPartenza` from `Itinerario` where `Codice` =
OLD.`CodItinerario` into var_data;
	if (datediff(var_data, now())) < 20 then
		signal sqlstate "45000" set message_text = "Cancelation has
expired!";
	end if;
END
```

Infine in TAPPA, sempre before insert, è stato implementato un trigger che controlla che si inserisca una tappa associata ad una città dello stesso tipo, cioè tappa europea con città europea e tappa italiana con città italiana, in quanto si evince dalla specifica che anche se l'Italia fa parte dell'Europa, si vuole separare un viaggio europeo (quindi fuori dall'Italia) e un viaggio italiano, inoltre controlla che non si inserisca un tipo diverso da 'Italian' o 'European'.

```
CREATE DEFINER = CURRENT_USER TRIGGER `mydb`.`Tappa_BEFORE_INSERT` BEFORE
INSERT ON `Tappa` FOR EACH ROW
BEGIN
	declare var_tipo varchar(45);
```

```
	if (NEW.`Tipo` <> 'Italian' and NEW.`Tipo` <> 'European') then
		signal sqlstate "45001" set message_text = "You must insert Italian or European";
	end if;
	select `Tipo` from `Città` where `Nome` = NEW.`NomeCittà` into var_tipo;
	if (NEW.`Tipo` <> var_tipo) then
		signal sqlstate "45000" set message_text = "The trip and the city must be of the same type (italian or european)";
	end if;
END
```

Eventi

È stato implementato come evento "cleanup", che molto semplicemente fa una pulizia dell'ambiente in un intervallo di tempo di un mese. Dunque si andranno a eliminare tutti gli itinerari che sono "vecchi" di almeno un mese, per gestire anche il caso di cancellazioni di foreign key nella ON DELETE ACTION, tutte queste key sono state impostate in ON DELETE CASCADE, in maniera tale da cancellare tutte le referenze a codesti itinerari.

```
SET GLOBAL event_scheduler = on;
CREATE EVENT IF NOT EXISTS `cleanup` on schedule every 1 month
on completion preserve comment 'Remove old itinerary' do
delete from `Itinerario`
where `DataRientro` <= (now() - interval 1 month);
```

Viste

È stata implementata come vista "Partecipanti", molto utile soprattutto nella procedure del generare un report con informazioni sui partecipanti, in quanto essa non serve nient'altro che a raggruppare più viaggi a seconda dei loro codici, con informazioni sul numero complessivo di partecipanti, che non è nient'altro che una somma del numero di partecipanti per ogni prenotazione a quel viaggio.

```
CREATE ALGORITHM=UNDEFINED DEFINER=`root`@`localhost` SQL SECURITY
DEFINER VIEW `mydb`.`Partecipanti`
(`IdItinerario`,`PartecipantiComplessivi`) AS select
```

```
`mydb`.`Itinerario`.`Codice` AS
`Codice`,sum(`mydb`.`Prenotazione`.`NumPartecipanti`) AS
`NumPartecipanti` from (`mydb`.`Itinerario` join `mydb`.`Prenotazione`
on((`mydb`.`Itinerario`.`Codice` =
`mydb`.`Prenotazione`.`CodItinerario`))) group by
`mydb`.`Itinerario`.`Codice`
```

Stored Procedures e transazioni

Questa procedure serve ad assegnare un albergo ad un itinerario, non è stato messa nessuna transazione in quanto si ha soltanto un inserimento.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE
`assegna_albergo_itinerario`(in var_indirizzo varchar(45), in var_citta
varchar(45), in var_nome varchar(45), in var_itinerario int)
BEGIN
    insert into `Pernottamento` values (var_itinerario, var_indirizzo,
var_nome, var_citta);
END
```

Questa procedure invece assegna un autobus a seconda del nome dell'itinerario, anche qui nessuna transazione in quanto si ha soltanto un inserimento.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE
`assegna_autobus_itinerario`(IN var_codice_autobus int, IN var_codice
int)
BEGIN
     insert into `Associazione` (`CodItinerario`, `CodAutobus`) values
(var_codice, var_codice_autobus);
END
```

Questa procedure permette di assegnare una tappa ad un albergo, anche qui come livello di isolamento è stato scelto READ COMMITTED poichè si chiamerà la assegna_albergo_itinerario in maniera tale da riempire entrambe le tabelle PERNOTTAMENO e ASSEGNAZIONE.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `assegna_tappa_albergo`(IN
var_numero int, IN var_codice int, IN var_indirizzo varchar(45), IN
var_nome varchar(45), IN var_citta varchar(45))
BEGIN
    declare exit handler for sqlexception
    begin
```

```
        rollback;  -- rollback any changes made in the transaction
        resignal;  -- raise again the sql exception to the caller
    end;
    set transaction isolation level read committed;
    start transaction;
    insert into `Assegnazione` values (var_numero, var_codice,
var_indirizzo, var_nome, var_citta);
    commit;
    call assegna_albergo_itinerario(var_indirizzo, var_citta, var_nome,
var_codice);
END
```

Questa procedure si occupa della cancellazione di una prenotazione, anche qui nessuna transazione poichè si ha solo una cancellazione.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `cancella_prenotazione`(IN
var_codice int)
BEGIN
     delete from `Prenotazione` where `Prenotazione`.`Codice` =
var_codice;
END
```

Questa procedure crea un nuovo utente che dovrà indicare oltre a username e password anche il ruolo.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `crea_utente`(IN username
varchar(45), IN pass varchar(45), IN ruolo varchar(45))
BEGIN
     insert into utenti values (username, MD5(pass), ruolo);
END
```

Questa procedure è quella che si occupa del generare un report con informazioni riguardanti i partecipanti e il guadagno derivato da un certo viaggio. Qui è stato utilizzato come livello di isolamento SERIALIZABLE , in particolar modo tutti i lock saranno mantenuti fino alla fine della transazione, permettondole di lavorare sul database come se fosse sola ed anche perchè si hanno più letture nella tabella ITINERARIO.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `genera_report`(IN var_id
int)
BEGIN
    declare var_giorni int;
    declare var_notti int;
    declare exit handler for sqlexception
    begin
        rollback;  -- rollback any changes made in the transaction
```

```
        resignal;  -- raise again the sql exception to the caller
    end;
    set transaction isolation level serializable;
    start transaction;
    select datediff(`DataRientro`, `DataPartenza`) + 1
           from `Itinerario`
        where `Codice` = var_id
        into var_giorni;
    set var_notti = var_giorni -1;
    select `PartecipantiComplessivi` as 'Numero Partecipanti',
(`Itinerario`.`Costo`*`PartecipantiComplessivi` - `CostoGiornaliero`*
var_giorni - `CostoPersona`*var_notti*`PartecipantiComplessivi`) as
'Guadagno'
           from `Itinerario` join `Partecipanti` join `Autobus` join
`Albergo`
         where `Itinerario`.`Codice` = var_id and
`Partecipanti`.`IdItinerario` = var_id;
    commit;
END
```

Questa procedure serve ad aggiungere un nuovo albergo, nessuna transazione anche qui.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `inserisci_albergo`(IN
var_indirizzo varchar(45), IN var_nome varchar(45), IN var_costo_persona
int, IN var_capienza_max int, IN var_referente varchar(45), IN var_email
varchar(45), IN var_tel varchar(45), IN var_fax varchar(45), IN var_citta
varchar(45))
BEGIN
     insert into `Albergo` (`Indirizzo`, `Nome`, `CostoPersona`,
`CapienzaMax`, `Referente`, `E-Mail`, `Telefono`, `Fax`,`NomeCittà`)
values (var_indirizzo, var_nome, var_costo_persona, var_capienza_max,
var_referente, var_email, var_tel, var_fax,var_citta);
END
```

Questa procedure serve ad inserire un nuovo autobus, nessuna transazione.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `inserisci_autobus`(IN
var_capienza int, IN var_costo_giorn int, OUT var_codice int)
BEGIN
     insert into `Autobus` (`Capienza`, `CostoGiornaliero`) values
(var_capienza, var_costo_giorn);
```

```
    set var_codice = last_insert_id();
END
```

Procedure per inserire una nuova città, nessuna transazione.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `inserisci_citta`(IN var_nome
varchar(45), IN var_tipo varchar(45))
BEGIN
    insert into `Città` values (var_nome, var_tipo);
END
```

Procedure che inserisce una nuova Guida, nessuna transazione.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `inserisci_guida`(IN var_CF
varchar(45))
BEGIN
    insert into `Guida` (`CF`) values (var_CF);
END
```

Procedure che inserisce un nuovo itinerario, nessuna transazione.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `inserisci_itinerario`(IN
var_costo int, IN var_dataP DATE, in var_dataR DATE, in var_nome
varchar(100), IN var_guida varchar(45),out var_codice int)
BEGIN
    insert into `Itinerario` (`Costo`, `DataPartenza`,
`DataRientro`,`CFGuida`,`Nome`) values (var_costo, var_dataP, var_dataR,
var_guida,var_nome);
    set var_codice = last_insert_id();
END
```

Procedure che aggiunge una nuova prenotazione associata all'itinerario scelto dal cliente, inoltre verrà generato un codice per disdirla. Nessuna transazione.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE
`inserisci_prenotazione`(IN var_num_partecipanti int, IN
var_itinerario int, out var_codice_prenotazione int)
BEGIN
    insert into `Prenotazione` (`NumPartecipanti`,`CodItinerario`)
values (var_num_partecipanti, var_itinerario);
    set var_codice_prenotazione = last_insert_id();
END
```

Procedure che consiste nell'inserimento di una nuova tappa, specificando la città in cui ci sarà la fermata, il tipo di tappa e l'itinerario di appartenenza. Nessuna transazione.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `inserisci_tappa`(IN
var_codice int, IN var_nome_citta varchar(45), IN var_tipo varchar(45),
OUT var_numero int)
BEGIN
    insert into `Tappa` (`CodItinerario`, `NomeCittà`, `Tipo`) values
(var_codice, var_nome_citta, var_tipo);
    set var_numero = last_insert_id();
END
```

Procedure che differenzia l'utente a seconda di come si è loggato.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `login`(IN var_username
varchar(45), IN var_pass varchar(45), OUT var_role int)
BEGIN
     declare var_user_role ENUM('agenzia', 'guida', 'cliente');
     select `ruolo` from `utenti`
          where `username` = var_username
               and `password` = md5(var_pass)
               into var_user_role;
        -- See the corresponding enum in the client
          if var_user_role = 'agenzia' then
               set var_role = 1;
          elseif var_user_role = 'guida' then
               set var_role = 2;
          elseif var_user_role = 'cliente' then
               set var_role = 3;
          else
               set var_role = 4;
          end if;
END
```

Procedure che stampa su schermo tutti gli itinerari, è stata settata la transazione come READ ONLY, per limitarla solo alla lettura di dati e come livello di isolamento READ COMMITTED.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `stampa_itinerari`()
BEGIN
     set transaction read only;
     set transaction isolation level read committed;
     select `Nome`, `Codice` from `Itinerario`;
    commit;
END
```

Procedure che stampa su schermo gli itinerari associati ad una determinata guida, anche qui transazione in READ ONLY e livello di isolamento in READ COMMITTED.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `stampa_itinerario_guida`(IN
var_CF varchar(45))
BEGIN
      set transaction read only;
      set transaction isolation level read committed;
      select `Nome`
            from `Itinerario`
                  where `CFGuida` = var_CF;
      commit;
END
```

Procedure che stampa le tappe che toccano un determinato viaggio, qui READ ONLY e READ COMMITTED.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `stampa_tappe`(IN
var_itinerario int)
BEGIN
    set transaction read only;
    set transaction isolation level read committed;
    select `NomeCittà` as 'Città Toccate dal Viaggio' from `Tappa` where
`CodItinerario` = var_itinerario;
    commit;
END
```

Appendice: Implementazione

Codice SQL per instanziare il database

Come scritto nella consegna, non sono stati inseriti in questa appendice tutto ciò di cui è stato discusso nel capitolo precedente.

```
-- MySQL Workbench Forward Engineering
SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0;
SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0;
SET @OLD_SQL_MODE=@@SQL_MODE,
SQL_MODE='ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZERO_
DATE,ERROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION';
-- -----------------------------------------------------
-- Schema mydb
-- -----------------------------------------------------
CREATE SCHEMA IF NOT EXISTS `mydb` ;
USE `mydb` ;
-- -----------------------------------------------------
-- Table `mydb`.`Città`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `mydb`.`Città` (
  `Nome` VARCHAR(45) NOT NULL,
  `Tipo` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`Nome`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;
-- -----------------------------------------------------
-- Table `mydb`.`Albergo`
-- -----------------------------------------------------
```

```
CREATE TABLE IF NOT EXISTS `mydb`.`Albergo` (
  `Indirizzo` VARCHAR(45) NOT NULL,
  `Nome` VARCHAR(45) NOT NULL,
  `CostoPersona` INT NOT NULL,
  `CapienzaMax` INT NOT NULL,
  `Referente` VARCHAR(45) NOT NULL,
  `E-Mail` VARCHAR(45) NOT NULL,
  `Telefono` VARCHAR(45) NOT NULL,
  `Fax` VARCHAR(45) NOT NULL,
  `NomeCittà` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`Indirizzo`, `Nome`, `NomeCittà`),
  INDEX `fk_Albergo_Città_idx` (`NomeCittà` ASC) VISIBLE,
  CONSTRAINT `fk_Albergo_Città_idx`
    FOREIGN KEY (`NomeCittà`)
    REFERENCES `mydb`.`Città` (`Nome`)
    ON DELETE RESTRICT
    ON UPDATE RESTRICT)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;
-- -----------------------------------------------------
-- Table `mydb`.`Guida`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `mydb`.`Guida` (
  `CF` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`CF`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
```

```
COLLATE = utf8mb4_0900_ai_ci;
-- -----------------------------------------------------
-- Table `mydb`.`Itinerario`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `mydb`.`Itinerario` (
  `Codice` INT NOT NULL AUTO_INCREMENT,
  `Costo` INT NOT NULL,
  `DataPartenza` DATE NOT NULL,
  `DataRientro` DATE NOT NULL,
  `CFGuida` VARCHAR(45) NOT NULL,
  `Nome` VARCHAR(100) NOT NULL,
  PRIMARY KEY (`Codice`),
  INDEX `fk_Itinerario_Guida_idx` (`CFGuida` ASC) VISIBLE,
  CONSTRAINT `fk_Itinerario_Guida_idx`
    FOREIGN KEY (`CFGuida`)
    REFERENCES `mydb`.`Guida` (`CF`)
    ON DELETE CASCADE
    ON UPDATE RESTRICT)
ENGINE = InnoDB
AUTO_INCREMENT = 20
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;
-- -----------------------------------------------------
-- Table `mydb`.`Tappa`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `mydb`.`Tappa` (
  `Numero` INT NOT NULL AUTO_INCREMENT,
  `CodItinerario` INT NOT NULL,
```

```
  `NomeCittà` VARCHAR(45) NOT NULL,
  `Tipo` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`Numero`, `CodItinerario`),
  INDEX `fk_Tappa_Itinerario_idx` (`CodItinerario` ASC) VISIBLE,
  INDEX `fk_Tappa_Città_idx` (`NomeCittà` ASC) VISIBLE,
  CONSTRAINT `fk_Tappa_Città_idx`
    FOREIGN KEY (`NomeCittà`)
    REFERENCES `mydb`.`Città` (`Nome`)
    ON DELETE RESTRICT
    ON UPDATE RESTRICT,
  CONSTRAINT `fk_Tappa_Itinerario_idx`
    FOREIGN KEY (`CodItinerario`)
    REFERENCES `mydb`.`Itinerario` (`Codice`)
    ON DELETE CASCADE
    ON UPDATE RESTRICT)
ENGINE = InnoDB
AUTO_INCREMENT = 6
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;
-- -----------------------------------------------------
-- Table `mydb`.`Assegnazione`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `mydb`.`Assegnazione` (
  `NumeroTappa` INT NOT NULL,
  `CodItinerario` INT NOT NULL,
  `IndirizzoAlbergo` VARCHAR(45) NOT NULL,
  `NomeAlbergo` VARCHAR(45) NOT NULL,
  `CittàAlbergo` VARCHAR(45) NOT NULL,
```

```
  PRIMARY KEY (`NumeroTappa`, `CodItinerario`),
  INDEX `fk_Assegnazione_Albergo1_idx` (`IndirizzoAlbergo` ASC,
`NomeAlbergo` ASC, `CittàAlbergo` ASC) VISIBLE,
  CONSTRAINT `fk_Assegnazione_Albergo1`
    FOREIGN KEY (`IndirizzoAlbergo` , `NomeAlbergo` , `CittàAlbergo`)
    REFERENCES `mydb`.`Albergo` (`Indirizzo` , `Nome` , `NomeCittà`)
    ON DELETE RESTRICT
    ON UPDATE RESTRICT,
  CONSTRAINT `fk_Assegnazione_Tappa1`
    FOREIGN KEY (`NumeroTappa` , `CodItinerario`)
    REFERENCES `mydb`.`Tappa` (`Numero` , `CodItinerario`)
    ON DELETE CASCADE)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;
-- -----------------------------------------------------
-- Table `mydb`.`Autobus`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `mydb`.`Autobus` (
  `Codice` INT NOT NULL AUTO_INCREMENT,
  `Capienza` INT NOT NULL,
  `CostoGiornaliero` INT NOT NULL,
  PRIMARY KEY (`Codice`))
ENGINE = InnoDB
AUTO_INCREMENT = 5
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;
-- -----------------------------------------------------
-- Table `mydb`.`Associazione`
```

```
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `mydb`.`Associazione` (
  `CodItinerario` INT NOT NULL,
  `CodAutobus` INT NOT NULL,
  PRIMARY KEY (`CodItinerario`, `CodAutobus`),
  INDEX `fk_Associazione_Autobus_idx` (`CodAutobus` ASC) VISIBLE,
  INDEX `fk_Associazione_Itinerario_idx` (`CodItinerario` ASC) VISIBLE,
  CONSTRAINT `fk_Associazione_Autobus_idx`
    FOREIGN KEY (`CodAutobus`)
    REFERENCES `mydb`.`Autobus` (`Codice`)
    ON DELETE RESTRICT
    ON UPDATE RESTRICT,
  CONSTRAINT `fk_Associazione_Itinerario_idx`
    FOREIGN KEY (`CodItinerario`)
    REFERENCES `mydb`.`Itinerario` (`Codice`)
    ON DELETE CASCADE
    ON UPDATE RESTRICT)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;
-- -----------------------------------------------------
-- Table `mydb`.`Pernottamento`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `mydb`.`Pernottamento` (
  `CodItinerario` INT NOT NULL,
  `IndirizzoAlbergo` VARCHAR(45) NOT NULL,
  `NomeAlbergo` VARCHAR(45) NOT NULL,
  `CittàAlbergo` VARCHAR(45) NOT NULL,
```

```
  PRIMARY KEY (`CodItinerario`, `IndirizzoAlbergo`, `NomeAlbergo`,
`CittàAlbergo`),
  INDEX `fk_Itinerario_has_Albergo_Albergo1_idx` (`IndirizzoAlbergo` ASC,
`NomeAlbergo` ASC, `CittàAlbergo` ASC) VISIBLE,
  INDEX `fk_Itinerario_has_Albergo_Itinerario1_idx` (`CodItinerario` ASC)
VISIBLE,
  CONSTRAINT `fk_Itinerario_has_Albergo_Albergo1`
    FOREIGN KEY (`IndirizzoAlbergo` , `NomeAlbergo` , `CittàAlbergo`)
    REFERENCES `mydb`.`Albergo` (`Indirizzo` , `Nome` , `NomeCittà`)
    ON DELETE RESTRICT
    ON UPDATE RESTRICT,
  CONSTRAINT `fk_Itinerario_has_Albergo_Itinerario1`
    FOREIGN KEY (`CodItinerario`)
    REFERENCES `mydb`.`Itinerario` (`Codice`)
    ON DELETE CASCADE)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;
-- -----------------------------------------------------
-- Table `mydb`.`Prenotazione`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `mydb`.`Prenotazione` (
  `Codice` INT NOT NULL AUTO_INCREMENT,
  `NumPartecipanti` INT NOT NULL,
  `CodItinerario` INT NOT NULL,
  PRIMARY KEY (`Codice`),
  INDEX `fk_Prenotazione_Itinerario_idx` (`CodItinerario` ASC) VISIBLE,
  CONSTRAINT `fk_Prenotazione_Itinerario_idx`
    FOREIGN KEY (`CodItinerario`)
```

```
    REFERENCES `mydb`.`Itinerario` (`Codice`)
    ON DELETE CASCADE
    ON UPDATE RESTRICT)
ENGINE = InnoDB
AUTO_INCREMENT = 13
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;
-- -----------------------------------------------------
-- Table `mydb`.`utenti`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `mydb`.`utenti` (
  `username` VARCHAR(45) NOT NULL,
  `password` CHAR(32) NOT NULL,
  `ruolo` ENUM('agenzia', 'guida', 'cliente') NOT NULL,
  PRIMARY KEY (`username`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;
USE `mydb` ;
CREATE USER 'agenzia' IDENTIFIED BY 'password';
GRANT EXECUTE ON procedure `mydb`.`assegna_albergo_itinerario` TO
'agenzia';
GRANT EXECUTE ON procedure `mydb`.`assegna_autobus_itinerario` TO
'agenzia';
GRANT EXECUTE ON procedure `mydb`.`assegna_tappa_albergo` TO 'agenzia';
GRANT EXECUTE ON procedure `mydb`.`crea_utente` TO 'agenzia';
GRANT EXECUTE ON procedure `mydb`.`inserisci_itinerario` TO 'agenzia';
GRANT EXECUTE ON procedure `mydb`.`genera_report` TO 'agenzia';
GRANT EXECUTE ON procedure `mydb`.`inserisci_albergo` TO 'agenzia';
```

```
GRANT EXECUTE ON procedure `mydb`.`inserisci_autobus` TO 'agenzia';
GRANT EXECUTE ON procedure `mydb`.`inserisci_citta` TO 'agenzia';
GRANT EXECUTE ON procedure `mydb`.`inserisci_guida` TO 'agenzia';
GRANT EXECUTE ON procedure `mydb`.`inserisci_tappa` TO 'agenzia';
CREATE USER 'cliente' IDENTIFIED BY 'password';
GRANT EXECUTE ON procedure `mydb`.`cancella_prenotazione` TO 'cliente';
GRANT EXECUTE ON procedure `mydb`.`inserisci_prenotazione` TO 'cliente';
GRANT EXECUTE ON procedure `mydb`.`stampa_itinerari` TO 'cliente';
GRANT EXECUTE ON procedure `mydb`.`stampa_tappe` TO 'cliente';
CREATE USER 'login' IDENTIFIED BY 'password';
GRANT EXECUTE ON procedure `mydb`.`login` TO 'login';
CREATE USER 'guida' IDENTIFIED BY 'password';
GRANT EXECUTE ON procedure `mydb`.`stampa_itinerario_guida` TO 'guida';
SET SQL_MODE=@OLD_SQL_MODE;
SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS;
SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS;
```

Codice del Front-End

File agenzia.c, cioè colui che chiamerà le stored procedure delegate all'agenzia e di far eseguire da terminale il menu associato ad essa.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "defines.h"

struct report{
    int participants;
    int profit;
};
```

```
static size_t parse_report(MYSQL *conn, MYSQL_STMT *stmt, struct report
**ret)
{
    int status;
      size_t row = 0;
      MYSQL_BIND param[2];

      int participants;
    int profit;

      if (mysql_stmt_store_result(stmt)) {
            fprintf(stderr, " mysql_stmt_execute(), 1 failed\n");
            fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
            exit(0);
      }

      *ret = malloc(mysql_stmt_num_rows(stmt) * sizeof(struct report));

      memset(param, 0, sizeof(param));
      param[0].buffer_type = MYSQL_TYPE_LONG;
      param[0].buffer = &participants;
      param[0].buffer_length = sizeof(participants);

      param[1].buffer_type = MYSQL_TYPE_LONG;
      param[1].buffer = &profit;
      param[1].buffer_length = sizeof(profit);


      if(mysql_stmt_bind_result(stmt, param)) {
            finish_with_stmt_error(conn, stmt, "Unable to bind column
parameters\n", true);
      }

      while (true) {
            status = mysql_stmt_fetch(stmt);
```

```
            if (status == 1 || status == MYSQL_NO_DATA)
                  break;

            (*ret)[row].participants = participants;
          (*ret)[row].profit = profit;

            row++;
      }
      return row;
}

static void assignBusIti (MYSQL *conn)
{
      MYSQL_STMT *prepared_stmt;
      MYSQL_BIND param[2];

      //Input for the assignment
      int codBus;
      char codBus_to_convert[46];
      int codIti;
      char codIti_to_convert[46];

      //Get the required information
      printf("\nAutobus Code: ");
      getInput(46, codBus_to_convert, false);
      printf("\nItinerary Code: ");
      getInput(46, codIti_to_convert, false);

      //Apply proper type conversion
      codBus = atoi(codBus_to_convert);
      codIti = atoi(codIti_to_convert);

      //Prepare stored procedure call
      if(!setup_prepared_stmt(&prepared_stmt, "call
assegna_autobus_itinerario(?,?)", conn)){
            finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize
assignment of the Autobus on the Itinerary statement\n", false);
      }
```

```
	//Prepare parameters
	memset(param, 0, sizeof(param));

	param[0].buffer_type = MYSQL_TYPE_LONG;
	param[0].buffer = &codBus;
	param[0].buffer_length = sizeof(codBus);

	param[1].buffer_type = MYSQL_TYPE_LONG;
	param[1].buffer = &codIti;
	param[1].buffer_length = sizeof(codIti);

	if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
		finish_with_stmt_error(conn, prepared_stmt, "Could not bind
parameters for Autobus to Itinerary assignment\n", true);
	}

	// Run procedure
	if (mysql_stmt_execute(prepared_stmt) != 0) {
		print_stmt_error (prepared_stmt, "An error occurred while assign
the Autobus to Itinerary.");
	} else {
		printf("Autobus correctly assigned...\n");
	}

	mysql_stmt_close(prepared_stmt);
}

static void assignTripHotel (MYSQL *conn)
{
	MYSQL_STMT *prepared_stmt;
	MYSQL_BIND param[5];

	//Input for the assignment
	int numero;
	char numero_to_convert[46];
	int codIti;
	char codIti_to_convert[46];
```

```
        char indirizzo[46];
        char name[46];
        char citta[46];

        //Get the required information
        printf("\nTrip number: ");
        getInput(46, numero_to_convert, false);
        printf("\nItinerary code: ");
        getInput(46, codIti_to_convert, false);
        printf("\nHotel address: ");
        getInput(46, indirizzo, false);
        printf("\nHotel name: ");
        getInput(46, name, false);
        printf("\nHotel city: ");
        getInput(46, citta, false);

        //Apply proper type conversion
        numero = atoi(numero_to_convert);
        codIti = atoi(codIti_to_convert);

        //Prepare stored procedure call
        if(!setup_prepared_stmt(&prepared_stmt, "call
    assegna_tappa_albergo(?,?,?,?,?)", conn)){
                finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize
    assignment of the Trip on the Hotel statement\n", false);
        }

        //Prepare parameters
        memset(param, 0, sizeof(param));

        param[0].buffer_type = MYSQL_TYPE_LONG;
        param[0].buffer = &numero;
        param[0].buffer_length = sizeof(numero);

        param[1].buffer_type = MYSQL_TYPE_LONG;
        param[1].buffer = &codIti;
        param[1].buffer_length = sizeof(codIti);
```

```
	param[2].buffer_type = MYSQL_TYPE_VAR_STRING;
	param[2].buffer = indirizzo;
	param[2].buffer_length = strlen(indirizzo);

	param[3].buffer_type = MYSQL_TYPE_VAR_STRING;
	param[3].buffer = name;
	param[3].buffer_length = strlen(name);

	param[4].buffer_type = MYSQL_TYPE_VAR_STRING;
	param[4].buffer = citta;
	param[4].buffer_length = strlen(citta);

	if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
		finish_with_stmt_error(conn, prepared_stmt, "Could not bind
parameters for Trip to Hotel assignment\n", true);
	}

	// Run procedure
	if (mysql_stmt_execute(prepared_stmt) != 0) {
		print_stmt_error (prepared_stmt, "An error occurred while assign
the Trip to Hotel.");
	} else {
		printf("Trip correctly assigned...\n");
	}

	mysql_stmt_close(prepared_stmt);
}

static void createUser (MYSQL *conn)
{
	MYSQL_STMT *prepared_stmt;
	MYSQL_BIND param[3];
	char options[5] = {'1','2', '3'};
	char r;

	// Input for the registration routine
	char username[46];
	char password[46];
```

```
	char ruolo[46];

	// Get the required information
	printf("\nUsername: ");
	getInput(46, username, false);
	printf("password: ");
	getInput(46, password, true);
	printf("Assign a possible role:\n");
	printf("\t1) Cliente\n");
	printf("\t2) Guida\n");
	printf("\t3) Agenzia\n");
	r = multiChoice("Select role", options, 3);

	// Convert role into enum value
	switch(r) {
		case '1':
			strcpy(ruolo, "cliente");
			break;
		case '2':
			strcpy(ruolo, "guida");
			break;
		case '3':
			strcpy(ruolo, "agenzia");
			break;
		default:
			fprintf(stderr, "Invalid condition at %s:%d\n", __FILE__,
__LINE__);
			abort();
	}

	// Prepare stored procedure call
	if(!setup_prepared_stmt(&prepared_stmt, "call crea_utente(?, ?, ?)",
conn)) {
		finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize
user insertion statement\n", false);
	}

	// Prepare parameters
```

```
	memset(param, 0, sizeof(param));

	param[0].buffer_type = MYSQL_TYPE_VAR_STRING;
	param[0].buffer = username;
	param[0].buffer_length = strlen(username);

	param[1].buffer_type = MYSQL_TYPE_VAR_STRING;
	param[1].buffer = password;
	param[1].buffer_length = strlen(password);

	param[2].buffer_type = MYSQL_TYPE_VAR_STRING;
	param[2].buffer = ruolo;
	param[2].buffer_length = strlen(ruolo);

	if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
		finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters for user insertion\n", true);
	}

	// Run procedure
	if (mysql_stmt_execute(prepared_stmt) != 0) {
		print_stmt_error (prepared_stmt, "An error occurred while adding the user.");
	} else {
		printf("User correctly added...\n");
	}

	mysql_stmt_close(prepared_stmt);
}

static void generateReport (MYSQL *conn)
{
	MYSQL_STMT *prepared_stmt;
	MYSQL_BIND param[1];
	int status;
    size_t row = 0;
    int guadagno;
    int participants;
```

```
    bool first = true;
    struct report *rep;

      //Input for the genearation
      int codIti;
      char codIti_to_convert[46];

      //Get the required information
    printf("\nItinerary code: ");
      getInput(46, codIti_to_convert, false);

      //Apply proper conversion
      codIti = atoi(codIti_to_convert);

      //Prepare stored procedure call
      if(!setup_prepared_stmt(&prepared_stmt, "call genera_report(?)", conn))
{
            finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize
the generation of report statement\n", false);
      }

      //Prepare parameters
      memset(param, 0, sizeof(param));

      param[0].buffer_type = MYSQL_TYPE_LONG;
      param[0].buffer = &codIti;
      param[0].buffer_length = sizeof(codIti);

      if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
            finish_with_stmt_error(conn, prepared_stmt, "Could not bind
parameters for the generation of report assignment\n", true);
      }
      // Run procedure
      if (mysql_stmt_execute(prepared_stmt) != 0) {
            print_stmt_error (prepared_stmt, "An error occurred while
generation of report.");
            goto out;
      }
```

```
    // We have multiple result sets here!
      do {
            // Skip OUT variables (although they are not present in the
5   procedure...)
            if(conn->server_status & SERVER_PS_OUT_PARAMS) {
                  goto next;
            }

10          if(first) {
             parse_report(conn, prepared_stmt, &rep);
             first = false;
          }
          else{
             printf("\nNumber of Participants: %d\nProfit of this Itinerary:
    %d\n", rep[row].participants, rep[row].profit);
             row++;
          }

20          // more results? -1 = no, >0 = error, 0 = yes (keep looking)
          next:
            status = mysql_stmt_next_result(prepared_stmt);
            if (status > 0)
                  finish_with_stmt_error(conn, prepared_stmt, "Unexpected
25  condition", true);

      } while (status == 0);

      out:
30    mysql_stmt_close(prepared_stmt);
    }

    static void insertHotel (MYSQL *conn)
    {
35    MYSQL_STMT *prepared_stmt;
      MYSQL_BIND param[9];

      //Input for the assignment
```

```
        char indirizzo[46];
        char nome[46];
        char costo_convert[46];
        char capienza_convert[46];
        int costo_persona;
        int capienza_max;
        char referente[46];
        char email[46];
        char tel[46];
        char fax[46];
        char citta[46];

        //Get the required information
        printf("\nHotel address: ");
        getInput(46, indirizzo, false);
        printf("\nHotel Name: ");
        getInput(46, nome, false);
        printf("\nHotel cost: ");
        getInput(46, costo_convert, false);
        printf("\nHotel capacity: ");
        getInput(46, capienza_convert, false);
        printf("\nHotel referent: ");
        getInput(46, referente, false);
        printf("\nHotel e-mail: ");
        getInput(46, email, false);
        printf("\nHotel telephone: ");
        getInput(46, tel, false);
        printf("\nHotel fax: ");
        getInput(46, fax, false);
        printf("\nHotel City: ");
        getInput(46, citta, false);

        //Aplly proper type convertions
        costo_persona = atoi(costo_convert);
        capienza_max = atoi(capienza_convert);

        //Prepare stored procedure call
```

```
	if(!setup_prepared_stmt(&prepared_stmt, "call
inserisci_albergo(?,?,?,?,?,?,?,?,?)", conn)){
		finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize
insertion of the Hotel statement\n", false);
	}

	//Prepare parameters
	memset(param, 0, sizeof(param));

	param[0].buffer_type = MYSQL_TYPE_VAR_STRING;
	param[0].buffer = indirizzo;
	param[0].buffer_length = strlen(indirizzo);

	param[1].buffer_type = MYSQL_TYPE_VAR_STRING;
	param[1].buffer = nome;
	param[1].buffer_length = strlen(nome);

	param[2].buffer_type = MYSQL_TYPE_LONG;
	param[2].buffer = &costo_persona;
	param[2].buffer_length = sizeof(costo_persona);

	param[3].buffer_type = MYSQL_TYPE_LONG;
	param[3].buffer = &capienza_max;
	param[3].buffer_length = sizeof(capienza_max);

	param[4].buffer_type = MYSQL_TYPE_VAR_STRING;
	param[4].buffer = referente;
	param[4].buffer_length = strlen(referente);

	param[5].buffer_type = MYSQL_TYPE_VAR_STRING;
	param[5].buffer = email;
	param[5].buffer_length = strlen(email);

	param[6].buffer_type = MYSQL_TYPE_VAR_STRING;
	param[6].buffer = tel;
	param[6].buffer_length = strlen(tel);
```

```
	param[7].buffer_type = MYSQL_TYPE_VAR_STRING;
	param[7].buffer = fax;
	param[7].buffer_length = strlen(fax);

	param[8].buffer_type = MYSQL_TYPE_VAR_STRING;
	param[8].buffer = citta;
	param[8].buffer_length = strlen(citta);

	if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
		finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters for Hotel insertion assignment\n", true);
	}

	// Run procedure
	if (mysql_stmt_execute(prepared_stmt) != 0) {
		print_stmt_error (prepared_stmt, "An error occurred while insert the Hotel.");
	} else {
		printf("Hotel correctly inserted...\n");
	}

	mysql_stmt_close(prepared_stmt);
}

static void insertAutobus (MYSQL *conn)
{
	MYSQL_STMT *prepared_stmt;
	MYSQL_BIND param[3];

	// Input for the registration routine
	int capienza;
	char capienza_convert[46];
	int costo_giorn;
	char costo_convert[46];
	int codice;

	// Get the required information
	printf("\nAutobus capacity: ");
```

```
	getInput(46, capienza_convert, false);
	printf("\nAutobus daily cost: ");
	getInput(46, costo_convert, false);

	//Apply proper type convertions
	capienza = atoi(capienza_convert);
	costo_giorn = atoi(costo_convert);

	// Prepare stored procedure call
	if(!setup_prepared_stmt(&prepared_stmt, "call inserisci_autobus(?, ?, ?)", conn)) {
		finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize Autobus insertion statement\n", false);
	}

	// Prepare parameters
	memset(param, 0, sizeof(param));

	param[0].buffer_type = MYSQL_TYPE_LONG;
	param[0].buffer = &capienza;
	param[0].buffer_length = sizeof(capienza);

	param[1].buffer_type = MYSQL_TYPE_LONG;
	param[1].buffer = &costo_giorn;
	param[1].buffer_length = sizeof(costo_giorn);

	param[2].buffer_type = MYSQL_TYPE_LONG; // OUT
	param[2].buffer = &codice;
	param[2].buffer_length = sizeof(codice);

	if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
		finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters for Autobus insertion\n", true);
	}

	// Run procedure
	if (mysql_stmt_execute(prepared_stmt) != 0) {
```

```
            print_stmt_error(prepared_stmt, "An error occurred while adding
the Autobus.");
            goto out;
      }

      // Get back the ID of the newly-added bus
      memset(param, 0, sizeof(param));
      param[0].buffer_type = MYSQL_TYPE_LONG; // OUT
      param[0].buffer = &codice;
      param[0].buffer_length = sizeof(codice);

      if(mysql_stmt_bind_result(prepared_stmt, param)) {
            finish_with_stmt_error(conn, prepared_stmt, "Could not retrieve
output parameter", true);
      }

      // Retrieve output parameter
      if(mysql_stmt_fetch(prepared_stmt)) {
            finish_with_stmt_error(conn, prepared_stmt, "Could not buffer
results", true);
      }

      printf("Autobus correctly added with Code %d...\n", codice);

    out:
      mysql_stmt_close(prepared_stmt);
}

static void insertCity (MYSQL *conn)
{
      MYSQL_STMT *prepared_stmt;
      MYSQL_BIND param[2];

      //Input for the assignment
      char nome[46];
      char tipo[46];

      //Get the required information
```

```
	printf("\nCity Name: ");
	getInput(46, nome, false);
	printf("\nCity type, write Italian if it's Italian otherwise European: ");
	getInput(46, tipo, false);

	//Prepare stored procedure call
	if(!setup_prepared_stmt(&prepared_stmt, "call inserisci_citta(?,?)", conn)){
		finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize insertion of the City statement\n", false);
	}

	//Prepare parameters
	memset(param, 0, sizeof(param));

	param[0].buffer_type = MYSQL_TYPE_VAR_STRING;
	param[0].buffer = nome;
	param[0].buffer_length = strlen(nome);

	param[1].buffer_type = MYSQL_TYPE_VAR_STRING;
	param[1].buffer = tipo;
	param[1].buffer_length = strlen(tipo);

	if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
		finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters for City insertion assignment\n", true);
	}

	// Run procedure
	if (mysql_stmt_execute(prepared_stmt) != 0) {
		print_stmt_error (prepared_stmt, "An error occurred while insert the City.");
	} else {
		printf("City correctly inserted...\n");
	}

	mysql_stmt_close(prepared_stmt);
```

```
}

static void insertGuide (MYSQL *conn)
{
      MYSQL_STMT *prepared_stmt;
      MYSQL_BIND param[1];

      //Input for the assignment
      char cf[46];

      //Get the required information
      printf("\nGuide fiscal code: ");
      getInput(46, cf, false);

      //Prepare stored procedure call
      if(!setup_prepared_stmt(&prepared_stmt, "call inserisci_guida(?)",
conn)){
            finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize
insertion of the Guide statement\n", false);
      }

      //Prepare parameters
      memset(param, 0, sizeof(param));

      param[0].buffer_type = MYSQL_TYPE_VAR_STRING;
      param[0].buffer = cf;
      param[0].buffer_length = strlen(cf);

      if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
            finish_with_stmt_error(conn, prepared_stmt, "Could not bind
parameters for Guide insertion assignment\n", true);
      }

      // Run procedure
      if (mysql_stmt_execute(prepared_stmt) != 0) {
            print_stmt_error (prepared_stmt, "An error occurred while insert
the Guide.");
      } else {
```

```
            printf("Guide correctly inserted...\n");
      }

      mysql_stmt_close(prepared_stmt);
}

static void insertItinerary (MYSQL *conn)
{
      MYSQL_STMT *prepared_stmt;
      MYSQL_BIND param[6];
      printf("\nDate is in form YYYY-MM-DD");

      // Input for the registration routine
      int costo;
      char costo_convert[46];
      char dataPartenza[11];
      MYSQL_TIME dataP;
      MYSQL_TIME dataR;
      char dataRientro[11];
      char nome[101];
      char cfGuida[46];
      int codice;

      // Get the required information
      printf("\nItinerary cost: ");
      getInput(46, costo_convert, false);
      printf("\nItinerary departure date: ");
      getInput(11, dataPartenza, false);
      printf("\nItinerary return date: ");
      getInput(11, dataRientro, false);
      printf("\nItinerary name: ");
      getInput(101, nome, false);
      printf("\nItinerary Guide fiscal code: ");
      getInput(46, cfGuida, false);

      //Apply proper type convertions
      costo = atoi(costo_convert);
      dataP = convertTime (dataPartenza);
```

```
        dataR = convertTime (dataRientro);

        // Prepare stored procedure call
        if(!setup_prepared_stmt(&prepared_stmt, "call
5   inserisci_itinerario(?, ?, ?, ?, ?, ?)", conn)) {
              finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize
    Itinerary insertion statement\n", false);
        }

10      // Prepare parameters
        memset(param, 0, sizeof(param));

        param[0].buffer_type = MYSQL_TYPE_LONG;
        param[0].buffer = &costo;
15      param[0].buffer_length = sizeof(costo);

        param[1].buffer_type = MYSQL_TYPE_DATE;
        param[1].buffer = &dataP;
        param[1].buffer_length = sizeof(dataP);
20
        param[2].buffer_type = MYSQL_TYPE_DATE;
        param[2].buffer = &dataR;
        param[2].buffer_length = sizeof(dataR);

25      param[3].buffer_type = MYSQL_TYPE_VAR_STRING;
        param[3].buffer = nome;
        param[3].buffer_length = strlen(nome);

        param[4].buffer_type = MYSQL_TYPE_VAR_STRING;
30      param[4].buffer = cfGuida;
        param[4].buffer_length = strlen(cfGuida);

        param[5].buffer_type = MYSQL_TYPE_LONG; // OUT
        param[5].buffer = &codice;
35      param[5].buffer_length = sizeof(codice);

        if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
```

```
            finish_with_stmt_error(conn, prepared_stmt, "Could not bind
parameters for Itinerary insertion\n", true);
      }

      // Run procedure
      if (mysql_stmt_execute(prepared_stmt) != 0) {
            print_stmt_error(prepared_stmt, "An error occurred while adding
the Itinerary.");
            goto out;
      }

      // Get back the ID of the newly-added itinerary
      memset(param, 0, sizeof(param));
      param[0].buffer_type = MYSQL_TYPE_LONG; // OUT
      param[0].buffer = &codice;
      param[0].buffer_length = sizeof(codice);

      if(mysql_stmt_bind_result(prepared_stmt, param)) {
            finish_with_stmt_error(conn, prepared_stmt, "Could not retrieve
output parameter", true);
      }

      // Retrieve output parameter
      if(mysql_stmt_fetch(prepared_stmt)) {
            finish_with_stmt_error(conn, prepared_stmt, "Could not buffer
results", true);
      }

      printf("Itinerary correctly added with Code %d...\n", codice);

    out:
      mysql_stmt_close(prepared_stmt);
}

static void insertTrip (MYSQL *conn)
{
      MYSQL_STMT *prepared_stmt;
      MYSQL_BIND param[4];
```

```
        // Input for the registration routine
        int codIti;
        char codIti_to_convert[46];
        char nomeCitta[46];
        char tipo[46];
        int codice;

        // Get the required information
        printf("\nItinerary code: ");
        getInput(100, codIti_to_convert, false);
        printf("\nTrip name of City: ");
        getInput(46, nomeCitta, false);
        printf("\nTrip type, write Italian if it's Italian otherwise European:
");
        getInput(46, tipo, false);

        //Apply proper convertion
        codIti = atoi(codIti_to_convert);

        // Prepare stored procedure call
        if(!setup_prepared_stmt(&prepared_stmt, "call
    inserisci_tappa(?, ?, ?, ?)", conn)) {
              finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize
    Trip insertion statement\n", false);
        }

        // Prepare parameters
        memset(param, 0, sizeof(param));

        param[0].buffer_type = MYSQL_TYPE_LONG;
        param[0].buffer = &codIti;
        param[0].buffer_length = sizeof(codIti);

        param[1].buffer_type = MYSQL_TYPE_VAR_STRING;
        param[1].buffer = nomeCitta;
        param[1].buffer_length = strlen(nomeCitta);
```

```
	param[2].buffer_type = MYSQL_TYPE_VAR_STRING;
	param[2].buffer = tipo;
	param[2].buffer_length = strlen(tipo);

	param[3].buffer_type = MYSQL_TYPE_LONG; // OUT
	param[3].buffer = &codice;
	param[3].buffer_length = sizeof(codice);

	if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
		finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters for Trip insertion\n", true);
	}

	// Run procedure
	if (mysql_stmt_execute(prepared_stmt) != 0) {
		print_stmt_error(prepared_stmt, "An error occurred while adding the Trip.");
		goto out;
	}

	// Get back the ID of the newly-added bus
	memset(param, 0, sizeof(param));
	param[0].buffer_type = MYSQL_TYPE_LONG; // OUT
	param[0].buffer = &codice;
	param[0].buffer_length = sizeof(codice);

	if(mysql_stmt_bind_result(prepared_stmt, param)) {
		finish_with_stmt_error(conn, prepared_stmt, "Could not retrieve output parameter", true);
	}

	// Retrieve output parameter
	if(mysql_stmt_fetch(prepared_stmt)) {
		finish_with_stmt_error(conn, prepared_stmt, "Could not buffer results", true);
	}

	printf("Trip correctly added with Number %d...\n", codice);
```

```
    out:
      mysql_stmt_close(prepared_stmt);
}

void run_as_agenzia (MYSQL *conn)
{
    char options[11] = {'1','2','3','4','5','6','7','8','9','a','q'};
      int op;

      printf("\nSwitching to agency role...");

      if(!parse_config("users/agenzia.json", &conf)) {
            fprintf(stderr, "Unable to load agency configuration\n");
            exit(EXIT_FAILURE);
      }

      if(mysql_change_user(conn, conf.db_username, conf.db_password,
conf.database)) {
            fprintf(stderr, "mysql_change_user() failed\n");
            exit(EXIT_FAILURE);
      }

      while(true){
            printf("\033[2J\033[H");
            printf("*** What should I do for you? ***\n\n");
            printf("1) Assign an Autobus to an Itinerary\n");
            printf("2) Assign a Trip to an Itinerary and to an Hotel\n");
            printf("3) Create a new user\n");
            printf("4) Generate a report of a Itinerary and its profit\n");
            printf("5) Insert a new Hotel\n");
            printf("6) Insert a new Autobus\n");
            printf("7) Insert a new City\n");
            printf("8) Insert a new Guide\n");
            printf("9) Insert a new Itinerary\n");
            printf("a) Insert a new Trip\n");
            printf("q) Quit\n");
```

```
op = multiChoice("Select an option", options, 12);

switch(op) {
      case '1':
            assignBusIti(conn);
            break;

      case '2':
            assignTripHotel(conn);
            break;

      case '3':
            createUser(conn);
            break;

      case '4':
            generateReport(conn);
            break;

      case '5':
            insertHotel(conn);
            break;

      case '6':
            insertAutobus(conn);
            break;

      case '7':
            insertCity(conn);
            break;

      case '8':
            insertGuide(conn);
            break;

      case '9':
            insertItinerary(conn);
```

```
                        break;

                case 'a':
                        insertTrip(conn);
                        break;

                case 'q':
                        return;

                default:
                        fprintf(stderr, "Invalid condition at %s:%d\n",
__FILE__, __LINE__);
                        abort();
            }

            getchar();
        }
}
```

File cliente.c, cioè colui che chiama le stored procedure delegate al cliente e di far eseguire da terminale il menù associato ad esso.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "defines.h"

static void deletePrenotation (MYSQL *conn)
{
      MYSQL_STMT *prepared_stmt;
      MYSQL_BIND param[1];

      //Input for the cancelation
      int codice;
      char cod[46];

      //Get the required information
```

```
        printf("\nPrenotation code: ");
        getInput(46, cod, false);

        //Apply proper type conversion
        codice = atoi(cod);

        //Prepare stored procedure call
        if(!setup_prepared_stmt(&prepared_stmt,                    "call
cancella_prenotazione(?)", conn)){
            finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize
cancelation of the Prenotation statement\n", false);
        }

        //Prepare parameters
        memset(param, 0, sizeof(param));

        param[0].buffer_type = MYSQL_TYPE_LONG;
        param[0].buffer = &codice;
        param[0].buffer_length = sizeof(codice);


        if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
            finish_with_stmt_error(conn,  prepared_stmt,  "Could  not  bind
parameters for Prenotation deletion\n", true);
        }

        // Run procedure
        if (mysql_stmt_execute(prepared_stmt) != 0) {
            print_stmt_error (prepared_stmt, "An error occurred while delete
the Prenotation.");
        }

        // Control if delete changed rows or not
        if (mysql_stmt_affected_rows(prepared_stmt) != 0){
            printf("Prenotation correctly deleted...\n");
        } else {
          printf("You insert wrong codes\n");
      }
```

```
        mysql_stmt_close(prepared_stmt);

    }
5
    static void insertPrenotation (MYSQL *conn)
    {
        MYSQL_STMT *prepared_stmt;
        MYSQL_BIND param[3];
10
          // Input for the registration routine
          int partecipanti;
          char part[46];
          int codIti;
15        char nome[46];
          int codice;

          // Get the required information
          printf("\nPrenotation number of participants : ");
20        getInput(46, part, false);
          printf("\nitinerary code: ");
          getInput(46, nome, false);

          //Apply proper type convertions
25        partecipanti = atoi(part);
          codIti = atoi(nome);

          // Prepare stored procedure call
          if(!setup_prepared_stmt(&prepared_stmt, "call inserisci_prenotazione(?,
30  ?, ?)", conn)) {
                finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize
    Prenotation insertion statement\n", false);
          }

35        // Prepare parameters
          memset(param, 0, sizeof(param));

          param[0].buffer_type = MYSQL_TYPE_LONG;
```

```
	param[0].buffer = &partecipanti;
	param[0].buffer_length = sizeof(&partecipanti);

	param[1].buffer_type = MYSQL_TYPE_LONG;
	param[1].buffer = &codIti;
	param[1].buffer_length = sizeof(codIti);

	param[2].buffer_type = MYSQL_TYPE_LONG; // OUT
	param[2].buffer = &codice;
	param[2].buffer_length = sizeof(codice);

	if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
		finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters for Prenotation insertion\n", true);
	}

	// Run procedure
	if (mysql_stmt_execute(prepared_stmt) != 0) {
		print_stmt_error(prepared_stmt, "An error occurred while adding the Prenotaion.");
		goto out;
	}

	// Get back the ID of the newly-added bus
	memset(param, 0, sizeof(param));
	param[0].buffer_type = MYSQL_TYPE_LONG; // OUT
	param[0].buffer = &codice;
	param[0].buffer_length = sizeof(codice);

	if(mysql_stmt_bind_result(prepared_stmt, param)) {
		finish_with_stmt_error(conn, prepared_stmt, "Could not retrieve output parameter", true);
	}

	// Retrieve output parameter
	if(mysql_stmt_fetch(prepared_stmt)) {
		finish_with_stmt_error(conn, prepared_stmt, "Could not buffer results", true);
```

```
      }

      printf("Prenotation correctly added with Code %d, please save this code if you want to dismiss the prenotation...\n", codice);

    out:
      mysql_stmt_close(prepared_stmt);
}

static void displayItinerary (MYSQL *conn)
{
      MYSQL_STMT *prepared_stmt;
    int status;

      if(!setup_prepared_stmt(&prepared_stmt, "call stampa_itinerari()", conn)){
            finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize itinerary list statement\n", false);
      }

      // Run procedure
      if (mysql_stmt_execute(prepared_stmt) != 0) {
            finish_with_stmt_error(conn, prepared_stmt, "Could not retrieve itinerary list\n", true);
        goto out;
    }

      // We have multiple result sets here!
      do {
            // Skip OUT variables (although they are not present in the procedure...)
            if(conn->server_status & SERVER_PS_OUT_PARAMS) {
                  goto next;
            }

        dump_result_set(conn, prepared_stmt, "List of Itineraries\n");

            // more results? -1 = no, >0 = error, 0 = yes (keep looking)
```

```
          next:
            status = mysql_stmt_next_result(prepared_stmt);
            if (status > 0)
                  finish_with_stmt_error(conn,    prepared_stmt,    "Unexpected
5   condition", true);

        } while (status == 0);

      out:
10      mysql_stmt_close(prepared_stmt);
    }

    static void displayTrips (MYSQL *conn)
    {
15      MYSQL_STMT *prepared_stmt;
        MYSQL_BIND param[1];
      int status;

        //Input for the displaying
20      char nome[46];
        int codIti;

        //Get the required information
        printf("\nitinerary code: ");
25      getInput(46, nome, false);

        codIti = atoi(nome);

        if(!setup_prepared_stmt(&prepared_stmt, "call stampa_tappe(?)", conn)){
30            finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize
    trip list statement\n", false);
        }

        // Prepare parameters
35      memset(param, 0, sizeof(param));

        param[0].buffer_type = MYSQL_TYPE_LONG;
        param[0].buffer = &codIti;
```

```
      param[0].buffer_length = sizeof(codIti);

      if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
            finish_with_stmt_error(conn,  prepared_stmt,  "Could  not  bind
5  parameters for trip list\n", true);
      }

      // Run procedure
      if (mysql_stmt_execute(prepared_stmt) != 0) {
10          finish_with_stmt_error(conn,  prepared_stmt,  "Could  not  retrieve
   trip list\n", true);
        goto out;
    }

15    // We have multiple result sets here!
      do {
            //  Skip  OUT  variables  (although  they  are  not  present  in  the
   procedure...)
            if(conn->server_status & SERVER_PS_OUT_PARAMS) {
20                goto next;
            }

          dump_result_set(conn, prepared_stmt, "List of city touched by this
   itinerary\n");
25
            // more results? -1 = no, >0 = error, 0 = yes (keep looking)
          next:
            status = mysql_stmt_next_result(prepared_stmt);
            if (status > 0)
30                finish_with_stmt_error(conn,   prepared_stmt,   "Unexpected
   condition", true);

      } while (status == 0);

35    out:
      mysql_stmt_close(prepared_stmt);
}
```

```
void run_as_cliente (MYSQL *conn)
{
      char options[5] = {'1','2','3','4','5'};
      char op;

      printf("Switching to client role...\n");

      if(!parse_config("users/cliente.json", &conf)) {
            fprintf(stderr, "Unable to load client configuration\n");
            exit(EXIT_FAILURE);
      }

      if(mysql_change_user(conn,      conf.db_username,      conf.db_password,
conf.database)) {
            fprintf(stderr, "mysql_change_user() failed\n");
            exit(EXIT_FAILURE);
      }

      while(true) {
            printf("\033[2J\033[H");
            printf("*** What should I do for you? ***\n\n");
            printf("1) Delete Prenotation\n");
            printf("2) Insert Prenotation to an itinerary\n");
            printf("3) Show list of itinerary available\n");
            printf("4) Show list of trips of an itinerary\n");
            printf("5) Quit\n");

            op = multiChoice("Select an option", options, 5);

            switch(op) {
                  case '1':
                        deletePrenotation(conn);
                        break;

                  case '2':
                        insertPrenotation(conn);
                        break;
```

```
                case '3':
                        displayItinerary(conn);
                        break;

                case '4':
                        displayTrips(conn);
                        break;

                case '5':
                        return;

                default:
                        fprintf(stderr, "Invalid condition at %s:%d\n", __FILE__, __LINE__);
                        abort();
            }

            getchar();
        }
}
```

File includes.h, che contiene una struct configuration dove ci sono informazioni inerenti all'host, db username, password, il numero di porta e il database stesso ed informazioni sull'username e password dell'utente che fa accesso. Inoltre sono defeinite funzioni che verranno poi utilizzati da altri programmi C.

```
#pragma once

#include <stdbool.h>
#include <mysql.h>

struct configuration {
      char *host;
      char *db_username;
      char *db_password;
      unsigned int port;
      char *database;
```

```
        char username[128];
        char password[128];
};

extern struct configuration conf;

extern int parse_config(char *path, struct configuration *conf);
extern char *getInput(unsigned int lung, char *stringa, bool hide);
extern bool yesOrNo(char *domanda, char yes, char no, bool predef, bool
insensitive);
extern char multiChoice(char *domanda, char choices[], int num);
extern MYSQL_TIME convertTime(char* string);
extern void print_error (MYSQL *conn, char *message);
extern void print_stmt_error (MYSQL_STMT *stmt, char *message);
extern void finish_with_error(MYSQL *conn, char *message);
extern void finish_with_stmt_error(MYSQL *conn, MYSQL_STMT *stmt, char
*message, bool close_stmt);
extern bool setup_prepared_stmt(MYSQL_STMT **stmt, char *statement, MYSQL
*conn);
extern void dump_result_set(MYSQL *conn, MYSQL_STMT *stmt, char *title);
extern void run_as_agenzia(MYSQL *conn);
extern void run_as_guida(MYSQL *conn);
extern void run_as_cliente(MYSQL *conn);
```

File guida.c, cioè colui che chiama le stored procedure delegate alla guida e di far eseguire da terminale il menu associato ad esso.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "defines.h"

static void displayItineraries (MYSQL *conn)
{
        MYSQL_STMT *prepared_stmt;
        MYSQL_BIND param[1];
      int status;
```

```
	//Input for the displaying
	char cf[46];

	//Get the required information
	printf("\nFiscal code of the guide is: ");
	getInput(46, cf, false);

	if(!setup_prepared_stmt(&prepared_stmt, "call
stampa_itinerario_guida(?)", conn)){
		finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize
itineraries list statement\n", false);
	}

	// Prepare parameters
	memset(param, 0, sizeof(param));

	param[0].buffer_type = MYSQL_TYPE_VAR_STRING;
	param[0].buffer = cf;
	param[0].buffer_length = strlen(cf);

	if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
		finish_with_stmt_error(conn, prepared_stmt, "Could not bind
parameters for itineraries list\n", true);
	}

	// Run procedure
	if (mysql_stmt_execute(prepared_stmt) != 0) {
		finish_with_stmt_error(conn, prepared_stmt, "Could not retrieve
itineraries list\n", true);
	  goto out;
	}

	// We have multiple result sets here!
	do {
		// Skip OUT variables (although they are not present in the
procedure...)
		if(conn->server_status & SERVER_PS_OUT_PARAMS) {
			goto next;
		}
```

```
        dump_result_set(conn, prepared_stmt, "List of itineraries at you
associated\n");

            // more results? -1 = no, >0 = error, 0 = yes (keep looking)
          next:
            status = mysql_stmt_next_result(prepared_stmt);
            if (status > 0)
                  finish_with_stmt_error(conn, prepared_stmt, "Unexpected
condition", true);

      } while (status == 0);

    out:
      mysql_stmt_close(prepared_stmt);
}

void run_as_guida (MYSQL *conn)
{
      char options[2] = {'1','2'};
      char op;

      printf("Switching to guide role...\n");

      if(!parse_config("users/guida.json", &conf)) {
            fprintf(stderr, "Unable to load guide configuration\n");
            exit(EXIT_FAILURE);
      }

      if(mysql_change_user(conn, conf.db_username, conf.db_password,
conf.database)) {
            fprintf(stderr, "mysql_change_user() failed\n");
            exit(EXIT_FAILURE);
      }

      while(true) {
            printf("\033[2J\033[H");
            printf("*** What should I do for you? ***\n\n");
            printf("1) Show itineraries at you associated\n");
```

```
            printf("2) Quit\n");

            op = multiChoice("Select an option", options, 2);

            switch(op) {
                  case '1':
                        displayItineraries(conn);
                        break;

                  case '2':
                        return;

                  default:
                        fprintf(stderr, "Invalid condition at %s:%d\n",
__FILE__, __LINE__);
                        abort();
            }

            getchar();
      }
}
```

File inout.c, cioè colui che permette di gestire l'interazione con l'utente a seconda di ciò che inserisce da tastiera, inoltre gestirà i segnali.

```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <termios.h>
#include <sys/ioctl.h>
#include <pthread.h>
#include <signal.h>
#include <stdbool.h>

#include "defines.h"

// Per la gestione dei segnali
```

```
static volatile sig_atomic_t signo;
typedef struct sigaction sigaction_t;
static void handler(int s);

char *getInput(unsigned int lung, char *stringa, bool hide)
{
	char c;
	unsigned int i;

	// Dichiara le variabili necessarie ad un possibile mascheramento dell'input
	sigaction_t sa, savealrm, saveint, savehup, savequit, saveterm;
	sigaction_t savetstp, savettin, savettou;
	struct termios term, oterm;

	if(hide) {
		// Svuota il buffer
		(void) fflush(stdout);

		// Cattura i segnali che altrimenti potrebbero far terminare il programma, lasciando l'utente senza output sulla shell
		sigemptyset(&sa.sa_mask);
		sa.sa_flags = SA_INTERRUPT; // Per non resettare le system call
		sa.sa_handler = handler;
		(void) sigaction(SIGALRM, &sa, &savealrm);
		(void) sigaction(SIGINT, &sa, &saveint);
		(void) sigaction(SIGHUP, &sa, &savehup);
		(void) sigaction(SIGQUIT, &sa, &savequit);
		(void) sigaction(SIGTERM, &sa, &saveterm);
		(void) sigaction(SIGTSTP, &sa, &savetstp);
		(void) sigaction(SIGTTIN, &sa, &savettin);
		(void) sigaction(SIGTTOU, &sa, &savettou);

		// Disattiva l'output su schermo
		if (tcgetattr(fileno(stdin), &oterm) == 0) {
			(void) memcpy(&term, &oterm, sizeof(struct termios));
			term.c_lflag &= ~(ECHO|ECHONL);
			(void) tcsetattr(fileno(stdin), TCSAFLUSH, &term);
		} else {
```

```
                (void) memset(&term, 0, sizeof(struct termios));
                (void) memset(&oterm, 0, sizeof(struct termios));
            }
        }

        // Acquisisce da tastiera al più lung - 1 caratteri
        for(i = 0; i < lung; i++) {
            (void) fread(&c, sizeof(char), 1, stdin);
            if(c == '\n') {
                stringa[i] = '\0';
                break;
            } else
                stringa[i] = c;

            // Gestisce gli asterischi
            if(hide) {
                if(c == '\b') // Backspace
                    (void) write(fileno(stdout), &c, sizeof(char));
                else
                    (void) write(fileno(stdout), "*", sizeof(char));
            }
        }

        // Controlla che il terminatore di stringa sia stato inserito
        if(i == lung - 1)
            stringa[i] = '\0';

        // Se sono stati digitati più caratteri, svuota il buffer della
    tastiera
        if(strlen(stringa) >= lung) {
            // Svuota il buffer della tastiera
            do {
                c = getchar();
            } while (c != '\n');
        }

        if(hide) {
            //L'a capo dopo l'input
            (void) write(fileno(stdout), "\n", 1);
```

```
            // Ripristina le impostazioni precedenti dello schermo
            (void) tcsetattr(fileno(stdin), TCSAFLUSH, &oterm);

            // Ripristina la gestione dei segnali
            (void) sigaction(SIGALRM, &savealrm, NULL);
            (void) sigaction(SIGINT, &saveint, NULL);
            (void) sigaction(SIGHUP, &savehup, NULL);
            (void) sigaction(SIGQUIT, &savequit, NULL);
            (void) sigaction(SIGTERM, &saveterm, NULL);
            (void) sigaction(SIGTSTP, &savetstp, NULL);
            (void) sigaction(SIGTTIN, &savettin, NULL);
            (void) sigaction(SIGTTOU, &savettou, NULL);

            // Se era stato ricevuto un segnale viene rilanciato al processo
stesso
            if(signo)
                  (void) raise(signo);
      }

      return stringa;
}

// Per la gestione dei segnali
static void handler(int s) {
      signo = s;
}


bool yesOrNo(char *domanda, char yes, char no, bool predef, bool insensitive)
{

      // I caratteri 'yes' e 'no' devono essere minuscoli
      yes = tolower(yes);
      no = tolower(no);

      // Decide quale delle due lettere mostrare come predefinite
      char s, n;
      if(predef) {
```

```
            s = toupper(yes);
            n = no;
        } else {
            s = yes;
            n = toupper(no);
        }

        // Richiesta della risposta
        while(true) {
            // Mostra la domanda
            printf("%s [%c/%c]: ", domanda, s, n);

            char c;
            getInput(1, &c, false);

            // Controlla quale risposta è stata data
            if(c == '\0') { // getInput() non può restituire '\n'!
                return predef;
            } else if(c == yes) {
                return true;
            } else if(c == no) {
                return false;
            } else if(c == toupper(yes)) {
                if(predef || insensitive) return true;
            } else if(c == toupper(yes)) {
                if(!predef || insensitive) return false;
            }
        }
    }

    char multiChoice(char *domanda, char choices[], int num)
    {

        // Genera la stringa delle possibilità
        char *possib = malloc(2 * num * sizeof(char));
        int i, j = 0;
        for(i = 0; i < num; i++) {
            possib[j++] = choices[i];
            possib[j++] = '/';
```

```
        }
        possib[j-1] = '\0'; // Per eliminare l'ultima '/'

        // Chiede la risposta
        while(true) {
            // Mostra la domanda
            printf("%s [%s]: ", domanda, possib);

            char c;
            getInput(1, &c, false);

            // Controlla se è un carattere valido
            for(i = 0; i < num; i++) {
                if(c == choices[i])
                    return c;
            }
        }
    }
```

File main.c, cioè colui che interagirà con l'utente, che una volta fatto il login avrà un menu associato al suo ruolo, scegliendo quello che vuole fare.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <mysql.h>

#include "defines.h"

typedef enum {
    AGENCY = 1,
    GUIDE,
    CLIENT,
    FAILED_LOGIN
} role_t;

struct configuration conf;

static MYSQL* conn;
```

```
static role_t attempt_login(MYSQL *conn, char* username, char* password)
{
	MYSQL_STMT *login_procedure;
	MYSQL_BIND param[3];
	int role = 0;

	if(!setup_prepared_stmt(&login_procedure, "call login(?, ?, ?)", conn))
{
		print_stmt_error(login_procedure, "Unable to initialize login
statement\n");
		goto err2;
	}

	// Prepare parameters
	memset(param, 0, sizeof(param));

	param[0].buffer_type = MYSQL_TYPE_VAR_STRING; // IN
	param[0].buffer = username;
	param[0].buffer_length = strlen(username);

	param[1].buffer_type = MYSQL_TYPE_VAR_STRING; // IN
	param[1].buffer = password;
	param[1].buffer_length = strlen(password);

	param[2].buffer_type = MYSQL_TYPE_LONG; // OUT
	param[2].buffer = &role;
	param[2].buffer_length = sizeof(role);

	if (mysql_stmt_bind_param(login_procedure, param) != 0) { // Note_param
		print_stmt_error(login_procedure, "Could not bind parameters for
login");
		goto err;
	}

	// Run procedure
	if (mysql_stmt_execute(login_procedure) != 0) {
		print_stmt_error(login_procedure, "Could not execute login
procedure");
```

```
            goto err;
        }

        // Prepare output parameters
        memset(param, 0, sizeof(param));
        param[0].buffer_type = MYSQL_TYPE_LONG; // OUT
        param[0].buffer = &role;
        param[0].buffer_length = sizeof(role);

        if(mysql_stmt_bind_result(login_procedure, param)) {
            print_stmt_error(login_procedure, "Could not retrieve output
parameter");
            goto err;
        }

        // Retrieve output parameter
        if(mysql_stmt_fetch(login_procedure)) {
            print_stmt_error(login_procedure, "Could not buffer results");
            goto err;
        }

        mysql_stmt_close(login_procedure);
        return role;

    err:
        mysql_stmt_close(login_procedure);
    err2:
        return FAILED_LOGIN;
}

int main () {
        role_t role;

        if(!parse_config("users/login.json", &conf)) {
            fprintf(stderr, "Unable to load login configuration\n");
            exit(EXIT_FAILURE);
        }

        conn = mysql_init (NULL);
```

```
        if (conn == NULL) {
            fprintf (stderr, "mysql_init() failed
    (probably out of memory)\n");
            exit(EXIT_FAILURE);
        }

        if (mysql_real_connect(conn, conf.host, conf.db_username,
    conf.db_password, conf.database, conf.port, NULL,
    CLIENT_MULTI_STATEMENTS | CLIENT_MULTI_RESULTS) == NULL) {
                fprintf (stderr, "mysql_real_connect() failed\n");
                mysql_close (conn);
                exit(EXIT_FAILURE);
        }

        printf("Username: ");
        getInput(128, conf.username, false);
        printf("Password: ");
        getInput(128, conf.password, true);

        role = attempt_login(conn, conf.username, conf.password);

        switch(role) {
            case CLIENT:
                run_as_cliente(conn);
                break;

            case AGENCY:
                run_as_agenzia(conn);
                break;

            case GUIDE:
                run_as_guida(conn);
                break;

            case FAILED_LOGIN:
                fprintf(stderr, "Invalid credentials\n");
                exit(EXIT_FAILURE);
                break;
```

```
            default:
                  fprintf(stderr, "Invalid condition at %s:%d\n", __FILE__,
__LINE__);
                  abort();
      }

      printf("Bye!\n");

      mysql_close (conn);
      return 0;
}
```

File utils.c, cioè colui che contiene tutte le funzioni sql di setup iniziale del prepared statement e di controllo di errori, inoltre è stata aggiunta una funziona convertTime, visto che si era presentato il problema della data inserita dall'utente che appunto non è di tipo MYSQL_TIME.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <mysql.h>
#include <time.h>

#include "defines.h"

MYSQL_TIME convertTime (char* string)
{
      struct tm *time = malloc(sizeof(struct tm));
      MYSQL_TIME sqlTime;
      //Set the delimiter for the string of date
      char delim[] = "-";

      //Strtok for separate the years, month and day
      time->tm_year = atoi(strtok(string, delim));
      time->tm_mon = atoi(strtok(NULL, delim));
      time->tm_mday = atoi(strtok(NULL,delim));

    //Now populate the MYSQL_TIME
    sqlTime.year = time->tm_year;
    sqlTime.month = time->tm_mon;
```

```
    sqlTime.day = time->tm_mday;

    return sqlTime;
}

void print_stmt_error (MYSQL_STMT *stmt, char *message)
{
      fprintf (stderr, "%s\n", message);
      if (stmt != NULL) {
            fprintf (stderr, "Error %u (%s): %s\n",
                  mysql_stmt_errno (stmt),
                  mysql_stmt_sqlstate(stmt),
                  mysql_stmt_error (stmt));
      }
}


void print_error(MYSQL *conn, char *message)
{
      fprintf (stderr, "%s\n", message);
      if (conn != NULL) {
            #if MYSQL_VERSION_ID >= 40101
            fprintf (stderr, "Error %u (%s): %s\n",
            mysql_errno (conn), mysql_sqlstate(conn), mysql_error (conn));
            #else
            fprintf (stderr, "Error %u: %s\n",
            mysql_errno (conn), mysql_error (conn));
            #endif
      }
}

bool setup_prepared_stmt(MYSQL_STMT **stmt, char *statement, MYSQL *conn)
{
      bool update_length = true;

      *stmt = mysql_stmt_init(conn);
      if (*stmt == NULL)
      {
            print_error(conn, "Could not initialize statement handler");
```

```
            return false;
      }

      if (mysql_stmt_prepare (*stmt, statement, strlen(statement)) != 0) {
            print_stmt_error(*stmt, "Could not prepare statement");
            return false;
      }

      mysql_stmt_attr_set(*stmt, STMT_ATTR_UPDATE_MAX_LENGTH,
&update_length);

      return true;
}

void finish_with_error(MYSQL *conn, char *message)
{
      print_error(conn, message);
      mysql_close(conn);
      exit(EXIT_FAILURE);
}

void finish_with_stmt_error(MYSQL *conn, MYSQL_STMT *stmt, char *message,
bool close_stmt)
{
      print_stmt_error(stmt, message);
      if(close_stmt)   mysql_stmt_close(stmt);
      mysql_close(conn);
      exit(EXIT_FAILURE);
}

static void print_dashes(MYSQL_RES *res_set)
{
      MYSQL_FIELD *field;
      unsigned int i, j;

      mysql_field_seek(res_set, 0);
      putchar('+');
      for (i = 0; i < mysql_num_fields(res_set); i++) {
            field = mysql_fetch_field(res_set);
```

```
            for (j = 0; j < field->max_length + 2; j++)
                  putchar('-');
            putchar('+');
      }
      putchar('\n');
}

static void dump_result_set_header(MYSQL_RES *res_set)
{
      MYSQL_FIELD *field;
      unsigned long col_len;
      unsigned int i;

      /* determine column display widths -- requires result set to be */
      /* generated with mysql_store_result(), not mysql_use_result() */

      mysql_field_seek (res_set, 0);

      for (i = 0; i < mysql_num_fields (res_set); i++) {
            field = mysql_fetch_field (res_set);
            col_len = strlen(field->name);

            if (col_len < field->max_length)
                  col_len = field->max_length;
            if (col_len < 4 && !IS_NOT_NULL(field->flags))
                  col_len = 4; /* 4 = length of the word "NULL" */
            field->max_length = col_len; /* reset column info */
      }

      print_dashes(res_set);
      putchar('|');
      mysql_field_seek (res_set, 0);
      for (i = 0; i < mysql_num_fields(res_set); i++) {
            field = mysql_fetch_field(res_set);
            printf(" %-*s |", (int)field->max_length, field->name);
      }
      putchar('\n');

      print_dashes(res_set);
```

```
}

void dump_result_set(MYSQL *conn, MYSQL_STMT *stmt, char *title)
{
      int i;
      int status;
      int num_fields;       /* number of columns in result */
      MYSQL_FIELD *fields;  /* for result set metadata */
      MYSQL_BIND *rs_bind;  /* for output buffers */
      MYSQL_RES *rs_metadata;
      MYSQL_TIME *date;
      size_t attr_size;

      /* Prefetch the whole result set. This in conjunction with
       * STMT_ATTR_UPDATE_MAX_LENGTH set in `setup_prepared_stmt`
       * updates the result set metadata which are fetched in this
       * function, to allow to compute the actual max length of
       * the columns.
       */
      if (mysql_stmt_store_result(stmt)) {
            fprintf(stderr, " mysql_stmt_execute(), 1 failed\n");
            fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
            exit(0);
      }

      /* the column count is > 0 if there is a result set */
      /* 0 if the result is only the final status packet */
      num_fields = mysql_stmt_field_count(stmt);

      if (num_fields > 0) {
            /* there is a result set to fetch */
            printf("%s\n", title);

            if((rs_metadata = mysql_stmt_result_metadata(stmt)) == NULL) {
                  finish_with_stmt_error(conn, stmt, "Unable to retrieve
result metadata\n",                     true);
            }

            dump_result_set_header(rs_metadata);
```

```
            fields = mysql_fetch_fields(rs_metadata);

            rs_bind = (MYSQL_BIND *)malloc(sizeof (MYSQL_BIND) * num_fields);
            if (!rs_bind) {
                  finish_with_stmt_error(conn, stmt, "Cannot allocate output
buffers\n", true);
            }
            memset(rs_bind, 0, sizeof (MYSQL_BIND) * num_fields);

            /* set up and bind result set output buffers */
            for (i = 0; i < num_fields; ++i) {

                  // Properly size the parameter buffer
                  switch(fields[i].type) {
                        case MYSQL_TYPE_DATE:
                        case MYSQL_TYPE_TIMESTAMP:
                        case MYSQL_TYPE_DATETIME:
                        case MYSQL_TYPE_TIME:
                              attr_size = sizeof(MYSQL_TIME);
                              break;
                        case MYSQL_TYPE_FLOAT:
                              attr_size = sizeof(float);
                              break;
                        case MYSQL_TYPE_DOUBLE:
                              attr_size = sizeof(double);
                              break;
                        case MYSQL_TYPE_TINY:
                              attr_size = sizeof(signed char);
                              break;
                        case MYSQL_TYPE_SHORT:
                        case MYSQL_TYPE_YEAR:
                              attr_size = sizeof(short int);
                              break;
                        case MYSQL_TYPE_LONG:
                        case MYSQL_TYPE_INT24:
                              attr_size = sizeof(int);
                              break;
                        case MYSQL_TYPE_LONGLONG:
```

```
                        attr_size = sizeof(int);
                        break;
                    default:
                        attr_size = fields[i].max_length;
                        break;
                }

                // Setup the binding for the current parameter
                rs_bind[i].buffer_type = fields[i].type;
                rs_bind[i].buffer = malloc(attr_size + 1);
                rs_bind[i].buffer_length = attr_size + 1;

                if(rs_bind[i].buffer == NULL) {
                    finish_with_stmt_error(conn, stmt, "Cannot allocate
output buffers\n", true);
                }
            }

            if(mysql_stmt_bind_result(stmt, rs_bind)) {
                finish_with_stmt_error(conn, stmt, "Unable to bind output
parameters\n", true);
            }

            /* fetch and display result set rows */
            while (true) {
                status = mysql_stmt_fetch(stmt);

                if (status == 1 || status == MYSQL_NO_DATA)
                    break;

                putchar('|');

                for (i = 0; i < num_fields; i++) {

                    if (rs_bind[i].is_null_value) {
                        printf (" %-*s |", (int)fields[i].max_length,
"NULL");
                        continue;
                    }
```

```
                    switch (rs_bind[i].buffer_type) {

                        case MYSQL_TYPE_VAR_STRING:
                        case MYSQL_TYPE_DATETIME:
                            printf(" %-*s |",
(int)fields[i].max_length,(char*)rs_bind[i].buffer);
                            break;

                        case MYSQL_TYPE_DATE:
                        case MYSQL_TYPE_TIMESTAMP:
                            date = (MYSQL_TIME *)rs_bind[i].buffer;
                            printf(" %d-%02d-%02d |", date->year, date-
>month,
                                date->day);
                            break;

                        case MYSQL_TYPE_STRING:
                            printf(" %-*s |",(int)fields[i].max_length,
(char*)rs_bind[i].buffer);
                            break;

                        case MYSQL_TYPE_FLOAT:
                        case MYSQL_TYPE_DOUBLE:
                            printf(" %.02f |",
*(float*)rs_bind[i].buffer);
                            break;

                        case MYSQL_TYPE_LONG:
                        case MYSQL_TYPE_SHORT:
                        case MYSQL_TYPE_TINY:
                            printf(" %-*d |",
(int)fields[i].max_length, *(int*)rs_bind[i].buffer);
                            break;

                        case MYSQL_TYPE_NEWDECIMAL:
                            printf(" %-*.02lf |",
(int)fields[i].max_length, *(float*)rs_bind[i].buffer);
                            break;
```

```
                                default:
                                    printf("ERROR: Unhandled type (%d)\n",
                                          rs_bind[i].buffer_type);
                                    abort();
                            }
                        }
                        putchar('\n');
                        print_dashes(rs_metadata);
                }

                mysql_free_result(rs_metadata); /* free metadata */

                /* free output buffers */
                for (i = 0; i < num_fields; i++) {
                        free(rs_bind[i].buffer);
                }
                free(rs_bind);
        }
}
```

File parse.c, che è un parser del file json, infatti per distinguere i vari utenti sono stati utilizzati 4 file json, guida, utente,login, agenzia.json.

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "defines.h"

#define BUFF_SIZE 4096

// The final config struct will point into this
static char config[BUFF_SIZE];

/**
 * JSON type identifier. Basic types are:
 *    o Object
```

```
 *    o Array
 *    o String
 *    o Other primitive: number, boolean (true/false) or null
 */
typedef enum {
      JSMN_UNDEFINED = 0,
      JSMN_OBJECT = 1,
      JSMN_ARRAY = 2,
      JSMN_STRING = 3,
      JSMN_PRIMITIVE = 4
} jsmntype_t;

enum jsmnerr {
      /* Not enough tokens were provided */
      JSMN_ERROR_NOMEM = -1,
      /* Invalid character inside JSON string */
      JSMN_ERROR_INVAL = -2,
      /* The string is not a full JSON packet, more bytes expected */
      JSMN_ERROR_PART = -3
};

/**
 * JSON token description.
 * type          type (object, array, string etc.)
 * start   start position in JSON data string
 * end           end position in JSON data string
 */
typedef struct {
      jsmntype_t type;
      int start;
      int end;
      int size;
#ifdef JSMN_PARENT_LINKS
      int parent;
#endif
} jsmntok_t;

/**
 * JSON parser. Contains an array of token blocks available. Also stores
```

```
 * the string being parsed now and current position in that string
 */
typedef struct {
      unsigned int pos; /* offset in the JSON string */
      unsigned int toknext; /* next token to allocate */
      int toksuper; /* superior token node, e.g parent object or array */
} jsmn_parser;

/**
 * Allocates a fresh unused token from the token pool.
 */
static jsmntok_t *jsmn_alloc_token(jsmn_parser *parser, jsmntok_t *tokens,
size_t num_tokens) {
      jsmntok_t *tok;
      if (parser->toknext >= num_tokens) {
            return NULL;
      }
      tok = &tokens[parser->toknext++];
      tok->start = tok->end = -1;
      tok->size = 0;
#ifdef JSMN_PARENT_LINKS
      tok->parent = -1;
#endif
      return tok;
}

/**
 * Fills token type and boundaries.
 */
static void jsmn_fill_token(jsmntok_t *token, jsmntype_t type,
                            int start, int end) {
      token->type = type;
      token->start = start;
      token->end = end;
      token->size = 0;
}

/**
 * Fills next available token with JSON primitive.
```

```
 */
static int jsmn_parse_primitive(jsmn_parser *parser, const char *js,
            size_t len, jsmntok_t *tokens, size_t num_tokens) {
      jsmntok_t *token;
      int start;

      start = parser->pos;

      for (; parser->pos < len && js[parser->pos] != '\0'; parser->pos++) {
            switch (js[parser->pos]) {
#ifndef JSMN_STRICT
                  /* In strict mode primitive must be followed by "," or "}"
or "]" */
                  case ':':
#endif
                  case '\t' : case '\r' : case '\n' : case ' ' :
                  case ','  : case ']'  : case '}' :
                        goto found;
            }
            if (js[parser->pos] < 32 || js[parser->pos] >= 127) {
                  parser->pos = start;
                  return JSMN_ERROR_INVAL;
            }
      }
#ifdef JSMN_STRICT
      /* In strict mode primitive must be followed by a comma/object/array */
      parser->pos = start;
      return JSMN_ERROR_PART;
#endif

found:
      if (tokens == NULL) {
            parser->pos--;
            return 0;
      }
      token = jsmn_alloc_token(parser, tokens, num_tokens);
      if (token == NULL) {
            parser->pos = start;
            return JSMN_ERROR_NOMEM;
```

```
		}
		jsmn_fill_token(token, JSMN_PRIMITIVE, start, parser->pos);
#ifdef JSMN_PARENT_LINKS
		token->parent = parser->toksuper;
#endif
		parser->pos--;
		return 0;
}

/**
 * Fills next token with JSON string.
 */
static int jsmn_parse_string(jsmn_parser *parser, const char *js,
		size_t len, jsmntok_t *tokens, size_t num_tokens) {
	jsmntok_t *token;

	int start = parser->pos;

	parser->pos++;

	/* Skip starting quote */
	for (; parser->pos < len && js[parser->pos] != '\0'; parser->pos++) {
		char c = js[parser->pos];

		/* Quote: end of string */
		if (c == '\"') {
			if (tokens == NULL) {
				return 0;
			}
			token = jsmn_alloc_token(parser, tokens, num_tokens);
			if (token == NULL) {
				parser->pos = start;
				return JSMN_ERROR_NOMEM;
			}
			jsmn_fill_token(token, JSMN_STRING, start+1, parser->pos);
#ifdef JSMN_PARENT_LINKS
			token->parent = parser->toksuper;
#endif
			return 0;
```

```
            }

            /* Backslash: Quoted symbol expected */
            if (c == '\\' && parser->pos + 1 < len) {
                  int i;
                  parser->pos++;
                  switch (js[parser->pos]) {
                        /* Allowed escaped symbols */
                        case '\"': case '/' : case '\\' : case 'b' :
                        case 'f' : case 'r' : case 'n'  : case 't' :
                              break;
                        /* Allows escaped symbol \uXXXX */
                        case 'u':
                              parser->pos++;
                              for(i = 0; i < 4 && parser->pos < len &&
    js[parser->pos] != '\0'; i++) {
                                    /* If it isn't a hex character we have an
    error */
                                    if(!((js[parser->pos] >= 48 && js[parser-
    >pos] <= 57) || /* 0-9 */
                                                (js[parser->pos] >= 65 &&
    js[parser->pos] <= 70) || /* A-F */
                                                (js[parser->pos] >= 97 &&
    js[parser->pos] <= 102))) { /* a-f */
                                          parser->pos = start;
                                          return JSMN_ERROR_INVAL;
                                    }
                                    parser->pos++;
                              }
                              parser->pos--;
                              break;
                        /* Unexpected symbol */
                        default:
                              parser->pos = start;
                              return JSMN_ERROR_INVAL;
                  }
            }
      }
      parser->pos = start;
```

```
	return JSMN_ERROR_PART;
}

/**
 * Parse JSON string and fill tokens.
 */
static int jsmn_parse(jsmn_parser *parser, const char *js, size_t len,
jsmntok_t *tokens, unsigned int num_tokens) {
	int r;
	int i;
	jsmntok_t *token;
	int count = parser->toknext;

	for (; parser->pos < len && js[parser->pos] != '\0'; parser->pos++) {
		char c;
		jsmntype_t type;

		c = js[parser->pos];
		switch (c) {
			case '{': case '[':
				count++;
				if (tokens == NULL) {
					break;
				}
				token = jsmn_alloc_token(parser, tokens, num_tokens);
				if (token == NULL)
					return JSMN_ERROR_NOMEM;
				if (parser->toksuper != -1) {
					tokens[parser->toksuper].size++;
#ifdef JSMN_PARENT_LINKS
					token->parent = parser->toksuper;
#endif
				}
				token->type = (c == '{' ? JSMN_OBJECT : JSMN_ARRAY);
				token->start = parser->pos;
				parser->toksuper = parser->toknext - 1;
				break;
			case '}': case ']':
				if (tokens == NULL)
```

```
                        break;
                    type = (c == '}' ? JSMN_OBJECT : JSMN_ARRAY);
#ifdef JSMN_PARENT_LINKS
                    if (parser->toknext < 1) {
                        return JSMN_ERROR_INVAL;
                    }
                    token = &tokens[parser->toknext - 1];
                    for (;;) {
                        if (token->start != -1 && token->end == -1) {
                            if (token->type != type) {
                                return JSMN_ERROR_INVAL;
                            }
                            token->end = parser->pos + 1;
                            parser->toksuper = token->parent;
                            break;
                        }
                        if (token->parent == -1) {
                            if(token->type != type || parser->toksuper
== -1) {
                                return JSMN_ERROR_INVAL;
                            }
                            break;
                        }
                        token = &tokens[token->parent];
                    }
#else
                    for (i = parser->toknext - 1; i >= 0; i--) {
                        token = &tokens[i];
                        if (token->start != -1 && token->end == -1) {
                            if (token->type != type) {
                                return JSMN_ERROR_INVAL;
                            }
                            parser->toksuper = -1;
                            token->end = parser->pos + 1;
                            break;
                        }
                    }
                    /* Error if unmatched closing bracket */
                    if (i == -1) return JSMN_ERROR_INVAL;
```

```
                        for (; i >= 0; i--) {
                              token = &tokens[i];
                              if (token->start != -1 && token->end == -1) {
                                    parser->toksuper = i;
                                    break;
                              }
                        }
#endif
                        break;
                  case '\"':
                        r = jsmn_parse_string(parser, js, len, tokens,
num_tokens);
                        if (r < 0) return r;
                        count++;
                        if (parser->toksuper != -1 && tokens != NULL)
                              tokens[parser->toksuper].size++;
                        break;
                  case '\t' : case '\r' : case '\n' : case ' ':
                        break;
                  case ':':
                        parser->toksuper = parser->toknext - 1;
                        break;
                  case ',':
                        if (tokens != NULL && parser->toksuper != -1 &&
                                    tokens[parser->toksuper].type != JSMN_ARRAY
&&
                                    tokens[parser->toksuper].type !=
JSMN_OBJECT) {
#ifdef JSMN_PARENT_LINKS
                              parser->toksuper = tokens[parser-
>toksuper].parent;
#else
                              for (i = parser->toknext - 1; i >= 0; i--) {
                                    if (tokens[i].type == JSMN_ARRAY ||
tokens[i].type == JSMN_OBJECT) {
                                          if (tokens[i].start != -1 &&
tokens[i].end == -1) {
                                                parser->toksuper = i;
                                                break;
```

```
                                        }
                                    }
                                }
#endif
                        }
                        break;
#ifdef JSMN_STRICT
                /* In strict mode primitives are: numbers and booleans */
                case '-': case '0': case '1' : case '2': case '3' : case
'4':
                case '5': case '6': case '7' : case '8': case '9':
                case 't': case 'f': case 'n' :
                        /* And they must not be keys of the object */
                        if (tokens != NULL && parser->toksuper != -1) {
                                jsmntok_t *t = &tokens[parser->toksuper];
                                if (t->type == JSMN_OBJECT ||
                                        (t->type == JSMN_STRING && t->size !=
0)) {
                                    return JSMN_ERROR_INVAL;
                                }
                        }
#else
                /* In non-strict mode every unquoted value is a primitive */
                default:
#endif
                        r = jsmn_parse_primitive(parser, js, len, tokens,
num_tokens);
                        if (r < 0) return r;
                        count++;
                        if (parser->toksuper != -1 && tokens != NULL)
                                tokens[parser->toksuper].size++;
                        break;

#ifdef JSMN_STRICT
                /* Unexpected char in strict mode */
                default:
                        return JSMN_ERROR_INVAL;
#endif
        }
```

```
        }

        if (tokens != NULL) {
                for (i = parser->toknext - 1; i >= 0; i--) {
                        /* Unmatched opened object or array */
                        if (tokens[i].start != -1 && tokens[i].end == -1) {
                                return JSMN_ERROR_PART;
                        }
                }
        }

        return count;
}

/**
 * Creates a new parser based over a given  buffer with an array of tokens
 * available.
 */
static void jsmn_init(jsmn_parser *parser) {
        parser->pos = 0;
        parser->toknext = 0;
        parser->toksuper = -1;
}

static int jsoneq(const char *json, jsmntok_t *tok, const char *s)
{
        if (tok->type == JSMN_STRING
            && (int) strlen(s) == tok->end - tok->start
            && strncmp(json + tok->start, s, tok->end - tok->start) == 0) {
                return 0;
        }
        return -1;
}

static size_t load_file(char *filename)
{
        FILE *f = fopen(filename, "rb");
        if(f == NULL) {
                fprintf(stderr, "Unable to open file %s\n", filename);
```

```
            exit(1);
        }

        fseek(f, 0, SEEK_END);
        size_t fsize = ftell(f);
        fseek(f, 0, SEEK_SET);  //same as rewind(f);

        if(fsize >= BUFF_SIZE) {
            fprintf(stderr, "Configuration file too large\n");
            abort();
        }

        fread(config, fsize, 1, f);
        fclose(f);

        config[fsize] = 0;
        return fsize;
    }

    int parse_config(char *path, struct configuration *conf)
    {
        int i;
        int r;
        jsmn_parser p;
        jsmntok_t t[128]; /* We expect no more than 128 tokens */

        load_file(path);

        jsmn_init(&p);
        r = jsmn_parse(&p, config, strlen(config), t, sizeof(t)/sizeof(t[0]));
        if (r < 0) {
            printf("Failed to parse JSON: %d\n", r);
            return 0;
        }

        /* Assume the top-level element is an object */
        if (r < 1 || t[0].type != JSMN_OBJECT) {
            printf("Object expected\n");
            return 0;
```

```
        }

        /* Loop over all keys of the root object */
        for (i = 1; i < r; i++) {
            if (jsoneq(config, &t[i], "host") == 0) {
                /* We may use strndup() to fetch string value */
                conf->host = strndup(config + t[i+1].start,
t[i+1].end-t[i+1].start);
                i++;
            } else if (jsoneq(config, &t[i], "username") == 0) {
                conf->db_username = strndup(config + t[i+1].start,
t[i+1].end-t[i+1].start);
                i++;
            } else if (jsoneq(config, &t[i], "password") == 0) {
                conf->db_password = strndup(config + t[i+1].start,
t[i+1].end-t[i+1].start);
                i++;
            } else if (jsoneq(config, &t[i], "port") == 0) {
                conf->port = strtol(config + t[i+1].start, NULL, 10);
                i++;
            } else if (jsoneq(config, &t[i], "database") == 0) {
                conf->database = strndup(config + t[i+1].start,
t[i+1].end-t[i+1].start);
                i++;
            } else {
                printf("Unexpected key: %.*s\n", t[i].end-t[i].start,
config + t[i].start);
            }
        }
        return 1;
}
```

I file json accennati precedentemente:

agenzia.json:

```
{
        "host": "localhost",
        "username": "agenzia",
        "password": "password",
        "port": 3306,
```

```
        "database": "mydb"
}
```

cliente.json:

```
{
        "host": "localhost",
        "username": "cliente",
        "password": "password",
        "port": 3306,
        "database": "mydb"
}
```

guida.json:

```
{
        "host": "localhost",
        "username": "guida",
        "password": "password",
        "port": 3306,
        "database": "mydb"
}
```

login.json:

```
{
        "host": "localhost",
        "username": "login",
        "password": "password",
        "port": 3306,
        "database": "mydb"
}
```

Infine il Makefile utilizzato per compilare e per far eseguire il client.

```
CC = gcc
INCLUDES = -I/usr/include/mysql
LIBS = -L/usr/local/lib/mysql -lmysqlclient
all: client
main.o: main.c
```

```
	$(CC) -c $(INCLUDES) main.c
agenzia.o:
	$(CC) -c $(INCLUDES) agenzia.c
cliente.o:
	$(CC) -c $(INCLUDES) cliente.c
guida.o:
	$(CC) -c $(INCLUDES) guida.c
inout.o:
	$(CC) -c $(INCLUDES) inout.c
parse.o:
	$(CC) -c $(INCLUDES) parse.c
utils.o: utils.c
	$(CC) -c $(INCLUDES) utils.c
client: main.o agenzia.o cliente.o guida.o inout.o parse.o utils.o
	$(CC) -o client main.o agenzia.o cliente.o guida.o inout.o parse.o
utils.o $(LIBS)
clean:
	rm -f client main.o agenzia.o cliente.o guida.o inout.o parse.o utils.o
	echo clean done
```