Basi di Dati

Progetto A.A. 2021/2022

Sistema informativo di un'agenzia di viaggi

0279744

Alessandro Caruso

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 (e-mail, 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
4	Compagnia	Agenzia	Sinonimo per “agenzia” in linea 1
5	Prenotazioni	Prenotazione di questi	Si esplicita il riferimento a “viaggio”
7	Itinerari di viaggio	Viaggio	Disambiguazione termine e sinonimo di “viaggio” in linea 2
7	Itinerario di viaggio	Viaggio	Disambiguazione termine e sinonimo di “viaggio” in linea 2
8-9	Da un insieme di pernottamenti in località potenzialmente differenti e di durata differente e da un costo	da un costo e da un programma di viaggio, caratterizzato da un insieme di tappe. Le tappe sono caratterizzate da una località e da una durata e possono prevedere il pernottamento.	Ristrutturo la frase per esplicitare riferimento di “costo” a “viaggio”. Introduco termine più specifico “programma di viaggio”. Introduco termine più specifico “tappa”, esplicitando il caso in cui una determinata tappa preveda un pernottamento. Inglobo le specifiche espresse in linea 16. Infine, introduco “caratterizzate” per standardizzazione struttura frase.
11	Pernottamento	tappa, se prevede il pernottamento	Sinonimo per “tappa” inoltre esplicito il caso in cui una determinata tappa preveda il pernottamento
12	Scelta	Scelta dell’albergo	Si esplicita il riferimento ad albergo
13-14	Ciascun albergo è associato ad una capienza massima di persone, una città, un indirizzo, un insieme di recapiti (e-mail, telefono, fax) e un referente	Ciascun albergo è caratterizzato da una capienza massima di persone, da una località, da un indirizzo, da un insieme di recapiti (e-mail, telefono, fax), da un referente e da un costo per notte per	Standardizzazione struttura frase, inglobo concetti espressi in linee 31-32. Inoltre sostituisco termine “città” con ”località” in quanto sinonimo con “località” in linea 2

		persona.	
16-17	Quando un piano di viaggio … coperte	---	Non aggiunge informazioni, dovuto a ristrutturazione di linee 8-9.
17	Gli utenti del sistema	Clienti	Termine più specifico, anche la segreteria è un utente del sistema
18	Persone	Partecipanti	Sinonimo per "partecipanti" in linea 12
18-19	fornito un codice	fornito loro un codice	Si esplicita riferimento ai clienti che hanno prenotato
20	Partenza	della data di partenza del viaggio da loro prenotato	Si esplicita riferimento ai clienti che hanno prenotato e uso termine "data di partenza" usato in precedenza per descrivere "viaggio"
22	Prenotazioni	Prenotazioni di un viaggio	Si esplicita riferimento a "viaggio"
22	Partenza	Data di partenza	Sinonimo di "data di partenza" in linea 8
23	Di quante persone parteciperanno	Del numero di partecipanti	Sinonimo di "partecipanti" in linea 12
25	Assegnano inoltre	La segreteria assegna inoltre al viaggio	Esplicito il riferimento di chi compie l'azione e verso "viaggio"
25-26	Un autobus può infatti avere una capienza differente. Ciascun autobus, inoltre, ha un costo giornaliero di utilizzo, che copre anche il costo dell'autista	Un autobus privato è caratterizzato da una capienza massima di persone e da un costo giornaliero di utilizzo, che copre anche il costo dell'autista	Sinonimi di "Autobus privati" in linea 2 e standardizzazione struttura frase.
25	Tutte le persone	Partecipanti	Sinonimo di "partecipanti" in linea 12
26	Autobus	Autobus privati	Sinonimo di "Autobus privati" in linea 2
27	Autobus	Autobus privati	Sinonimo di "Autobus privati" in linea 2
29	Autobus	Autobus privati	Sinonimo di "Autobus privati" in linea 2
29	Città toccata dal viaggio	Tappa del viaggio che prevede il	Sinonimo di "tappa" inoltre richiedo il caso in cui una determinata tappa preveda il pernottamento

		pernottamento	
30	disponibilità massima di posti	Capienza massima di persone	Sinonimo per “capienza massima di persone” in linea 13
31	Persone che effettuano il viaggio	Numero di partecipanti del viaggio	Sinonimo di “partecipanti” in linea 12
31	Hotel	Albergo	Sinonimo di Albergo in linea 11
32	Ciascun albergo ha un costo per notte per persona che è conservato nel sistema	---	Non aggiunge informazioni, dovuto a ristrutturazione di linee 13-14.

Specifica disambiguata

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.

L’agenzia intende realizzare un sistema informativo per la gestione dei viaggi e delle prenotazioni di questi.

La segreteria crea nuovi viaggi. Un viaggio è caratterizzato da una data di partenza, da una data di rientro, da un costo e da un programma di viaggio, caratterizzato da un insieme di tappe. Le tappe sono caratterizzate da una località e da una durata e possono prevedere il pernottamento.

Per ciascuna tappa, se prevede il pernottamento, è di interesse tenere traccia di quale sia l’albergo scelto per ospitare i partecipanti. La scelta dell’albergo avviene all’interno di un insieme di alberghi mantenuti nel sistema informativo. Ciascun albergo è caratterizzato da una capienza massima di persone, da una località, da un indirizzo, da un insieme di recapiti (e-mail, telefono, fax), da un referente e da un costo per notte per persona.

I clienti possono prenotarsi al viaggio specificando il numero di persone per cui effettuano la prenotazione. Al termine della prenotazione viene fornito loro un codice che può essere usato per disdire la prenotazione, fino a 20 giorni prima della data di partenza del viaggio da loro prenotato.

Quando le prenotazioni di un viaggio sono concluse (ovverosia, 20 giorni prima della data di 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 al viaggio un insieme di autobus privati per consentire il trasporto di tutti i partecipanti. Un autobus privato è caratterizzato da una capienza massima di persone e da un costo giornaliero di utilizzo, che copre anche il costo dell’autista.

Una volta definito il numero di autobus privati necessari, la segreteria, per ciascuna tappa del viaggio che prevede il pernottamento, associa un albergo che abbia capienza massima di persone superiore al numero di partecipanti del viaggio.

Per tutti i viaggi terminati, la segreteria può generare un report che mostri le informazioni sui partecipanti e il guadagno derivante da tale viaggio.

Glossario dei Termini

Termine	Descrizione	Collegamenti
Viaggio	Viaggi organizzati, offerti dall'agenzia che utilizza il sistema informativo.	Autobus privato, Tappa, Prenotazione, Partecipante, Guida
Programma di Viaggio	Insieme delle tappe in programma considerando i relativi, opzionali, pernottamenti	
Cliente	Un utente del sistema che intende prenotare un viaggio offerto dal sistema	Viaggio, Prenotazione
Segreteria	Un utente del sistema che possiede i privilegi particolari per l'accesso al sistema, in particolare per l'organizzazione dei viaggi.	Viaggio
Autobus Privato	Mezzo di trasporto per consentire il trasporto dei partecipanti ai viaggi.	Viaggio
Tappa	Luogo di sosta per i partecipanti al viaggio. Può prevedere il pernottamento.	Albergo, Partecipante
Prenotazione	È la prenotazione di un viaggio offerto dall'agenzia da parte di un cliente.	Partecipante, Viaggio
Partecipante	È un cliente che si è correttamente prenotato a un viaggio.	Viaggio
Albergo	È un albergo che consente il pernottamento dei partecipanti di un viaggio in una determinata tappa che prevede il pernottamento.	Viaggio, Tappa
Guida	Guide che lavorano per l'agenzia e che possono essere assegnate ai viaggi offerti dall'agenzia	Viaggio
Report	Resoconto di un viaggio offerto dall'agenzia che si è concluso.	Viaggio

Raggruppamento dei requisiti in insiemi omogenei

Frasi di carattere generale
Un'agenzia di viaggio organizza viaggio della durata da un giorno ad una settimana in località italiane ed europee. L'agenzia intende realizzare una base di dati per la gestione dei suoi viaggi e delle prenotazioni di questi.

Frasi relative ai viaggi
I viaggi utilizzano autobus privati. La segreteria crea nuovi viaggi. Un viaggio è caratterizzato da una data di partenza, da una data di rientro, da un costo e da un programma di viaggio. Quando le prenotazioni di un viaggio sono concluse (ovverosia, 20 giorni prima della data di 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 al viaggio un insieme di autobus privati per consentire il trasporto di tutti i partecipanti. Per tutti i viaggi terminati, la segreteria può generare un report che mostri le informazioni sui partecipanti e il guadagno derivante da tale viaggio.

Frasi relative ai programmi di viaggio
un programma di viaggio, caratterizzato da un insieme di tappe

Frasi relative agli autobus privati
Un autobus privato è caratterizzato da una capienza massima di persone e da un costo giornaliero di utilizzo, che copre anche il costo dell'autista.

Frasi relative ai partecipanti

Frasi relative alle guide

Frasi relative alle tappe
Le tappe sono caratterizzate da una località e da una durata e possono prevedere il pernottamento. Per ciascuna tappa, se prevede il pernottamento, è di interesse tenere traccia di quale sia l'albergo scelto per ospitare i partecipanti. Una volta definito il numero di autobus privati necessari, la segreteria, per ciascuna tappa dell'itinerario di viaggio che prevede il pernottamento, associa un albergo che abbia capienza massima di persone superiore al numero di partecipanti del viaggio.

Frasi relative agli alberghi
La scelta dell'albergo avviene all'interno di un insieme di alberghi mantenuti nel sistema informativo. Ciascun albergo è caratterizzato da una capienza massima di persone, da una località, da un indirizzo, da un insieme di recapiti (e-mail, telefono, fax), da un referente e da un costo per notte per persona.

Frasi relative alle prenotazioni
I clienti possono prenotarsi al viaggio specificando il numero di persone per cui effettuano la prenotazione. Al termine della prenotazione viene fornito loro un codice che può essere usato per disdire la prenotazione, fino a 20 giorni prima della data di partenza del viaggio da loro prenotato. Quando le prenotazioni di un viaggio sono concluse (ovverosia, 20 giorni prima della data di partenza), la segreteria è a conoscenza del numero di partecipanti al viaggio.

3. Progettazione concettuale

Costruzione dello schema E-R

Ho deciso di utilizzare una strategia mista, iniziando come **primo passo** con uno "**schema scheletro**" che include i concetti principali.

Dopo questa prima analisi ho ottenuto:

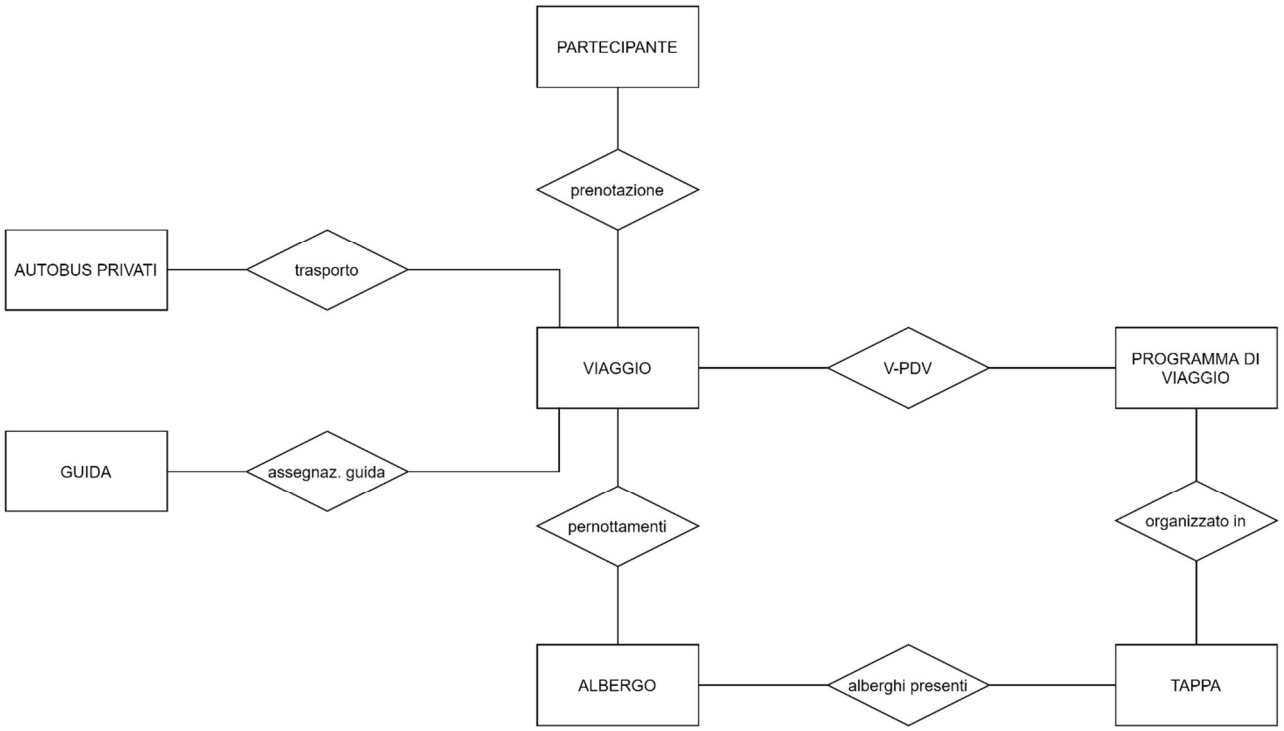


Figura 1

Come **secondo passo** ho deciso di rappresentare con uno schema E-R il minimondo costituito dalle entità **Viaggio-Programma di Viaggio-Tappa-Albergo**.

Inizialmente mi sono concentrato sull'entità "Programma Di Viaggio".

Ho aggiunto tre attributi: il primo "ID Programma" è un codice per ogni programma di viaggio che lo identifica univocamente; il secondo "numero giorni" indica il numero di giornate in cui il programma è organizzato; infine abbiamo l'attributo "Nome Programma". Ho poi introdotto una relazione "organizzato in" tra "Tappa" e "Programma Di Viaggio", con cardinalità: (1, N) per l'entità "Programma Di Viaggio", avendo presupposto che ogni programma di viaggio abbia almeno una tappa e ne può contenere fino ad N; (1, 1) per l'entità "Tappa" in quanto ogni tappa si presuppone appartenga ad uno e un solo viaggio. Quindi, dopo questa analisi ho ottenuto:

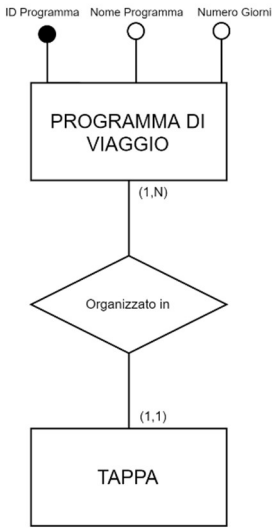


Figura 2

Sono passato poi all'analisi dell'entità "Tappa". Come primo passo ho introdotto una generalizzazione per distinguere i casi in cui una tappa preveda il pernottamento oppure no, aggiungendo l'entità "Tappa Con Pernottamento" con attributo "Trattamento" per specificare il trattamento del pernottamento quali: FB (Full Board), HB (Half Board) , BB (Bed And Breakfast) e OB (Only Bed). Come richiesto esplicitamente dalle specifiche ho introdotto l'attributo "località" all'entità padre. Successivamente, ho introdotto gli attributi "Ora arrivo", "Ora partenza", "Giorno arrivo" e "Giorno partenza" per descrivere in modo più dettagliato il concetto di "durata" espresso dalle specifiche. L'attributo "Giorno arrivo" è un intero positivo che rappresenta il numero ordinario del giorno in cui è prevista la tappa ; ad esempio: 1(primo), 2(secondo), 3(terzo) e così via. "Ora arrivo" indica invece l'ora in cui è previsto l'arrivo nella località a cui fa riferimento la tappa. Sono stati fatti gli stessi ragionamenti per gli attributi "Ora partenza" e "Giorno partenza", con la differenza che si sta facendo riferimento alla partenza. Da questa analisi ho ottenuto:

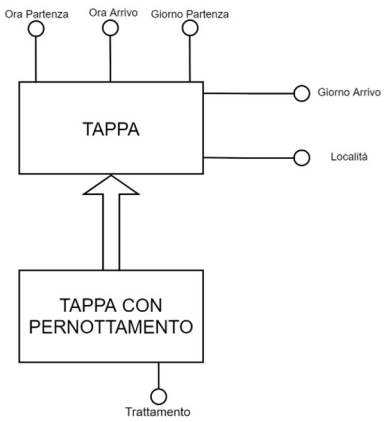


Figura 3

Successivamente ho deciso di reificare l'attributo "località", avendo ragionato su come questo potesse essere considerato come un concetto autonomo; infatti, anche il concetto di "albergo" fa riferimento al concetto di "Località". Nella reificazione ho introdotto l'attributo "Nome" che la identifica univocamente e "Stato" che indica la nazione della località. Si osservi che non è richiesta una partecipazione minima dell'entità "Località" alla relazione "in" poiché assumiamo che nel nostro sistema informativo possano essere registrate anche località a cui ancora non afferisce nessuna tappa. Quindi, ottengo:

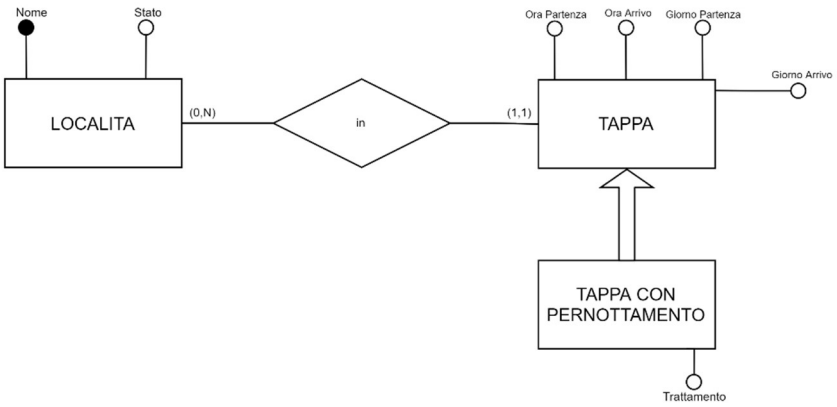


Figura 4

Per quanto riguarda l'entità "Albergo", come richiesto esplicitamente dalle specifiche, ho introdotto gli attributi "Capienza max persone", "Referente", "Indirizzo", "Costo Per Notte Per Persona" e l'attributo composto "Recapiti", che raggruppa gli attributi "fax", "email" e "telefono". Ho poi introdotto poi un attributo "Nome", che indica il nome dell'albergo in questione. Inoltre, dovuto alla reificazione fatta nel passo precedente, introduco una relazione "alberghi presenti", tra "Località" e "Albergo". Infine, ho introdotto un identificatore esterno all'entità Albergo. Quindi, ottengo:

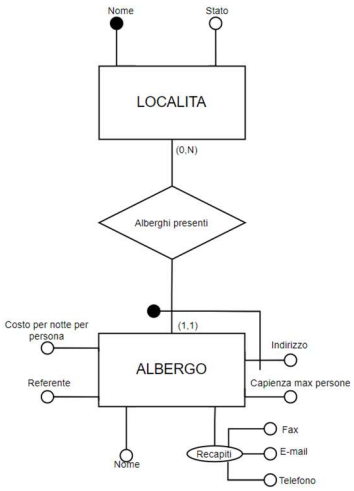


Figura 5

Come ultimo passo dell'analisi di questo minimondo ho considerato l'entità "Viaggio". Come descritto dalle specifiche ho introdotto gli attributi viaggio "Data di partenza", "Data di rientro" e "Costo". Inoltre, ho aggiunto un attributo "Numero Partecipanti" per tenere il conto dei clienti che si sono correttamente prenotati e non disdetti. Ho poi introdotto una relazione "ha" per esplicitare che ogni "Viaggio" fa riferimento un "Programma Di Viaggio". Le cardinalità sono: (1,1) per "Viaggio", in quanto nel nostro sistema informativo ogni viaggio fa riferimento a uno e un solo "Programma Di Viaggio"; (0, N) per "Programma Di Viaggio" poiché possono esistere più viaggi che fanno riferimento allo stesso programma e possono esistere programmi a cui non afferisce ancora nessun viaggio. Successivamente ho introdotto la relazione "pernottamenti" per associare correttamente ad ogni viaggio un insieme di "Alberghi" per consentire il pernottamento di tutti i partecipanti, con attributo di relazione "numero notti" che indica il numero di notti previste per uno specifico pernottamento in albergo. Le cardinalità sono: (0, N) per "Viaggio" poiché non necessariamente un viaggio ha dei pernottamenti ma può averne fino a un numero generico N; (0, N) in quanto non necessariamente un albergo ospita i partecipanti di qualche viaggio ma potrebbe anche averne di viaggi diversi. Infine, ho introdotto un identificatore esterno per l'entità "viaggio" .

Quindi ottengo:

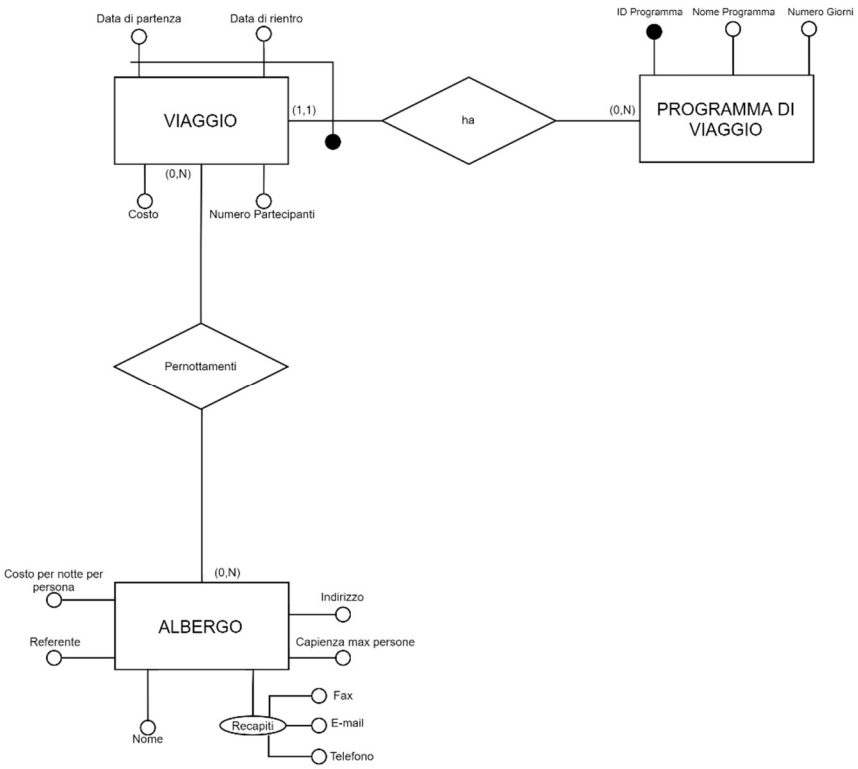


Figura 6

Infine, mettendo insieme queste entità finora analizzate ed aggiungendo un identificatore esterno per l'entità "tappa", ottengo un diagramma E-R per il minimondo **Viaggio-Programma di Viaggio-Tappa-Albergo** considerato:

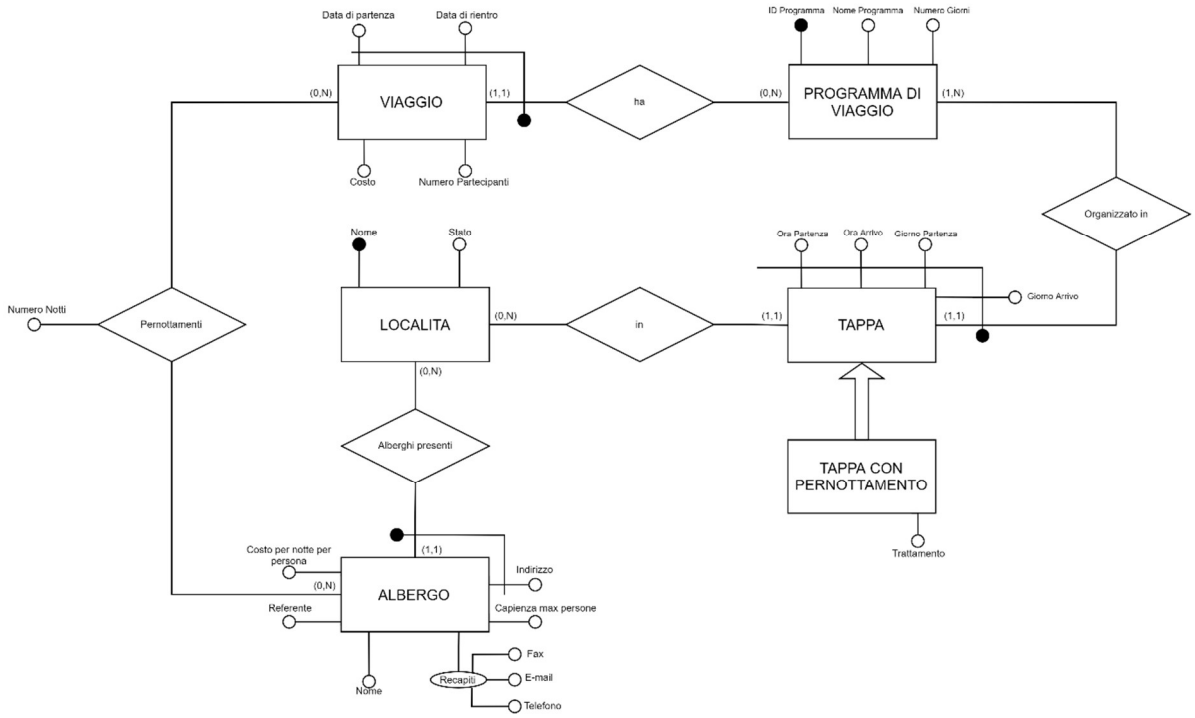


Figura 7 : minimondo Viaggio-Programma di Viaggio-Tappa-Albergo

Come **terzo passo** ho deciso di considerare il minimondo **Viaggio-Partecipante.** La prima entità che ho considerato è "partecipante". Le specifiche riguardo questa entità sono vaghe, infatti viene semplicemente richiesto di tenere traccia delle loro informazioni. Di conseguenza, seppur non richiesto esplicitamente dalle specifiche ho deciso di introdurre gli attributi "E-mail", "Telefono", "Nazionalità", "Età", "Nome", "Cognome" e "Codice Fiscale", quest'ultimo identificativo dell'entità. Ho poi introdotto una relazione "prenotazione" con l'entità "Viaggio, ottenendo lo schema:

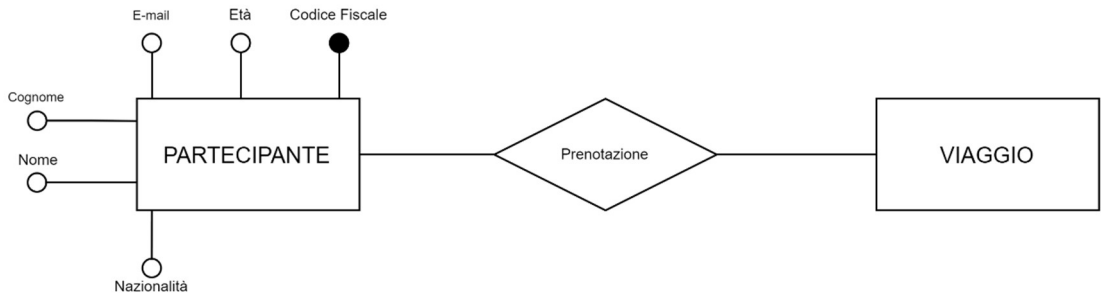


Figura 8

Successivamente però, ho ritenuto conveniente, al fine di rappresentare adeguatamente il concetto di prenotazione così come descritto dalle specifiche, reificare la relazione “Prenotazione” rendendola un’entità. A questa, ho aggiunto gli attributi : “Data”, che indica la data della prenotazione e “ID Prenotazione”, che la identifica univocamente.

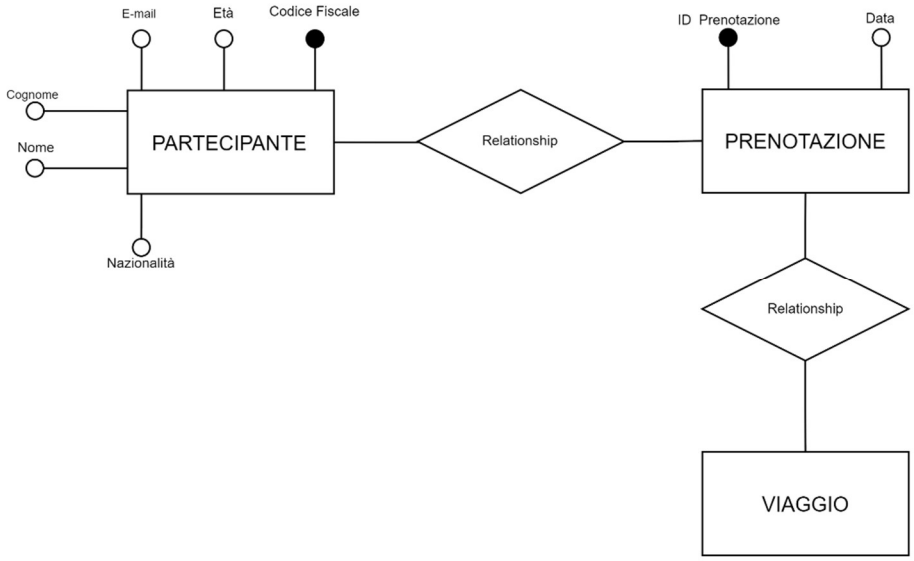


Figura 9

Come passo successivo, ho introdotto l’entità “Prenotazione Disdicibile”, generalizzata da “Prenotazione”, al fine di catturare il concetto di prenotazioni disdicibili che emerge dalle specifiche. Ho poi aggiunto l’attributo “codice per disdire”, che è un codice che gli utenti possono usare per disdire la propria prenotazione, così come richiesto dalle specifiche. Infine, ho introdotto le cardinalità delle relazioni. Nello specifico, per quanto riguarda la relazione tra “Partecipante” e “Prenotazione” ho cardinalità: (1,N) per “Partecipante”, poiché ogni partecipante può rientrare in una o più prenotazioni; (1, N) per “Prenotazione” in quanto, come descritto dalle specifiche, è ammesso richiedere una prenotazione indicando uno o più partecipanti. Dopo questa analisi ho ottenuto:

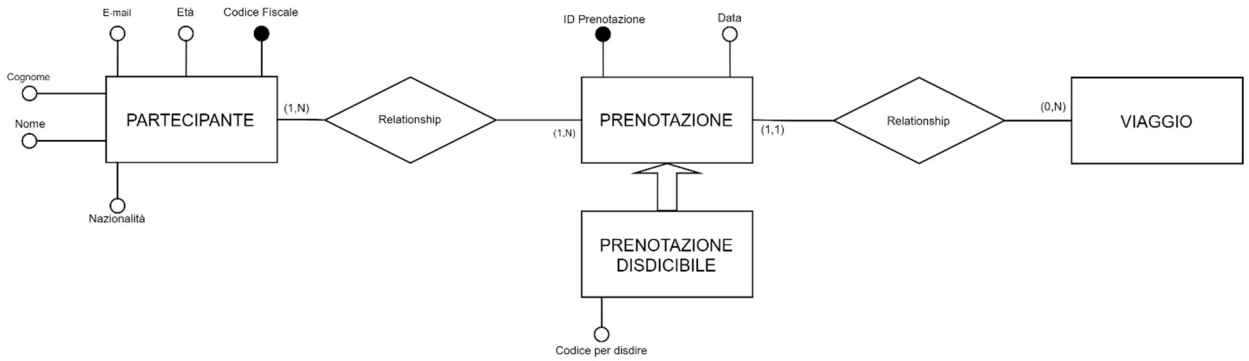


Figura 10 : minimondo Viaggio-Partecipante-Report

Come **quarto passo**, ho considerato il minimondo **Viaggio-Guida-Autobus Privato.**

Come primo step di questo passo ho considerato l'entità "Autobus Privato", a questa ho aggiunto attributi "capienza massima di persone" e "costo giornaliero" così come richiesto esplicitamente dalle specifiche. Successivamente, ho aggiunto un attributo "targa" che lo identifica univocamente e un attributo "Nome Deposito" che potrebbe risultare utile durante l'assegnazione dell'autobus privato al viaggio. Infine, ho aggiunto la relazione "Utilizzo" tra "Viaggio" e "Autobus Privato", con cardinalità: (0, N) per "Viaggio", poiché ad ogni viaggio vengono assegnati uno o più autobus privati; (0, N) per "Autobus Privato" in quanto assumo che esistano autobus privati ancora non sono assegnati a nessun viaggio, ma che in generale possono consentire il trasporto dei partecipanti per più viaggi. Alla fine di questa prima analisi ottengo lo schema:

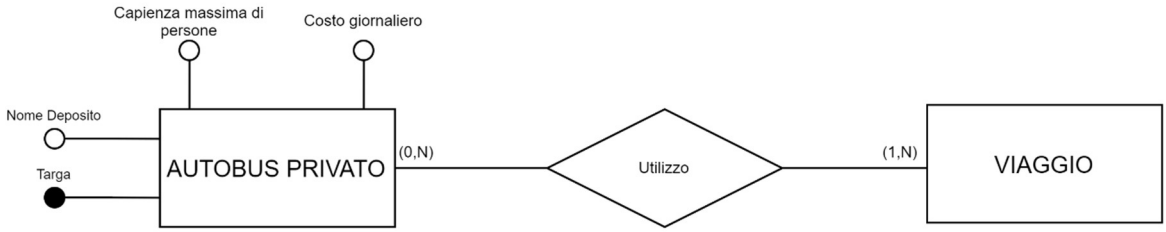


Figura 11

Come secondo step sono passato all'analisi dell'entità "Guida". Non disponendo di indicazioni precise da parte delle specifiche, ho deciso di introdurre gli attributi " "Età", "E-mail", "Nome", "Cognome" e "Codice fiscale", che la identifica univocamente. Infine, ho aggiunto una relazione tra "Guida" e "Viaggio", con cardinalità: (1,1) per "Viaggio", poiché dalle specifiche emerge che ad ogni viaggio viene assegnata una e una sola guida; (0, N) per "Guida" poiché una guida può essere assegnata a più viaggi oppure a nessuno.

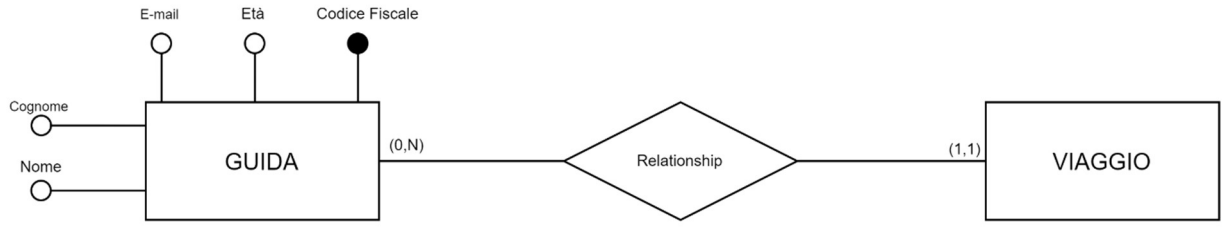


Figura 12

Quindi, alla fine di questo quarto passo ottengo lo schema:

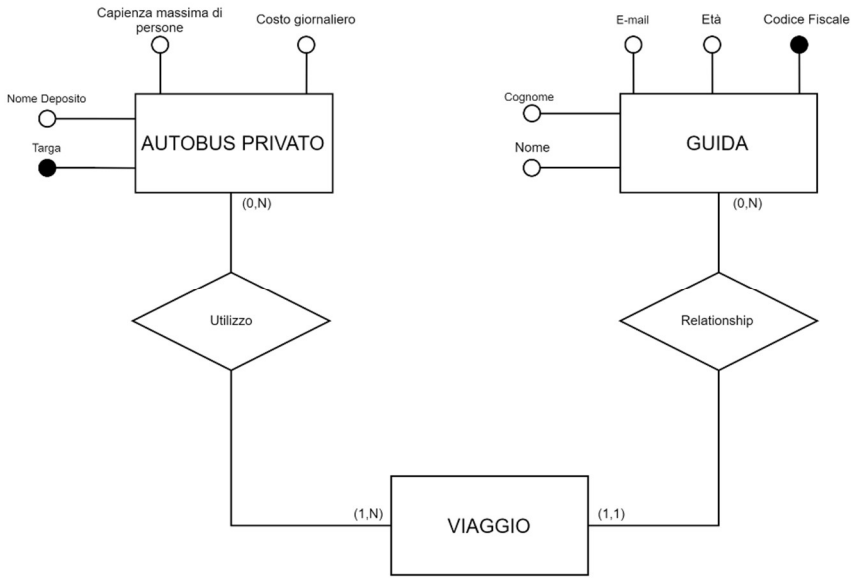


Figura 13: minimondo Autobus Privato-Viaggio-Guida

Integrazione finale

Dopo la prima analisi ho ottenuto tre schemi parziali principali, rappresentati in *Figura 7, Figura 12* e in *Figura 15*. Per risolvere i conflitti strutturali:

Ho introdotto un entità "Persona" che generalizza le entità "Guida" e "Partecipante" poiché in queste erano presenti attributi ripetuti quali: "Età", "Codice Fiscale", "Email" e "Telefono". Si osservi come questa generalizzazione è stata intesa come totale in quanto nel nostro minimondo l'unione dei sottoinsieme dei figli ("Guida" e "Partecipante") costituisce l'insieme padre ("Persona"). Infine, per questa generalizzazione ho ritenuto opportuno sostituire l'attributo "Età" con un attributo "Data Nascita".

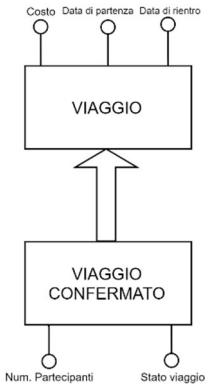
Ho considerato nello schema finale l'entità "Viaggio" come quella rappresentata in *Figura 7*, avendo definito i vari attributi e identificatori in maniera più specifica.

Ho poi però ritenuto opportuno introdurre una nuova entità, nello specifico "Viaggio Confermato" che è generalizzato dall'entità "Viaggio" e che rappresenta un viaggio alla cui partenza mancano meno di 20 giorni, al quale quindi la segreteria può assegnare un insieme di Alberghi, di Autobus Privati ed una Guida. Questa nuova entità modella quindi il concetto di viaggio espresso in *Figura 14*, quindi sostituirò quell'entità "Viaggio" con quella appena introdotta. Successivamente ho ritenuto necessario, avendo introdotto questa entità, far riferire la relazione "Pernottamenti", in *Figura* 7, all nuova entità "Viaggio Confermato".

Ho poi deciso di spostare l'attributo "num. partecipanti" , che era presente nell'entità padre "Viaggio" , solo nell'entità figlia. Infine ho modellato la seguente situazione: un viaggio confermato è "in programma"/ "attivo"/ "terminato", aggiungendo l'attributo "stato viaggio".

Seppur espresso anche successivamente, segue una descrizione più accurata degli stati dei viaggi, essendo un concetto fondamentale per il corretto funzionamento del nostro sistema informativo. Appena creato, supponendo una data di partenza successiva di più di 20 giorni alla data odierna, un viaggio è "prenotabile", cioè quello che abbiamo definito come entità "Viaggio". Un viaggio prenotabile permette ai clienti di prenotarsi ed eventualmente, finchè il viaggio persiste in quest stato, anche di disdire una loro prenotazione. Quando invece mancano 20 giorni alla partenza, il viaggio è "confermato"; per i viaggi confermati non è più possibile prenotarsi ne disdire prenotazioni. Inizialmente un viaggio confermato è nello stato "in programma", cioè la segretaria conosce il numero effettivo di partecipanti e può procedere con l'assegnazione di Autobus Privati, Alberghi e di una Guida. Nel momento in cui la data di partenza coincide con la data attuale, il viaggio confermato è nello stato "attivo" e vi persisterà finchè la data di rientro non sarà successiva alla data attuale, diventa poi "terminato".

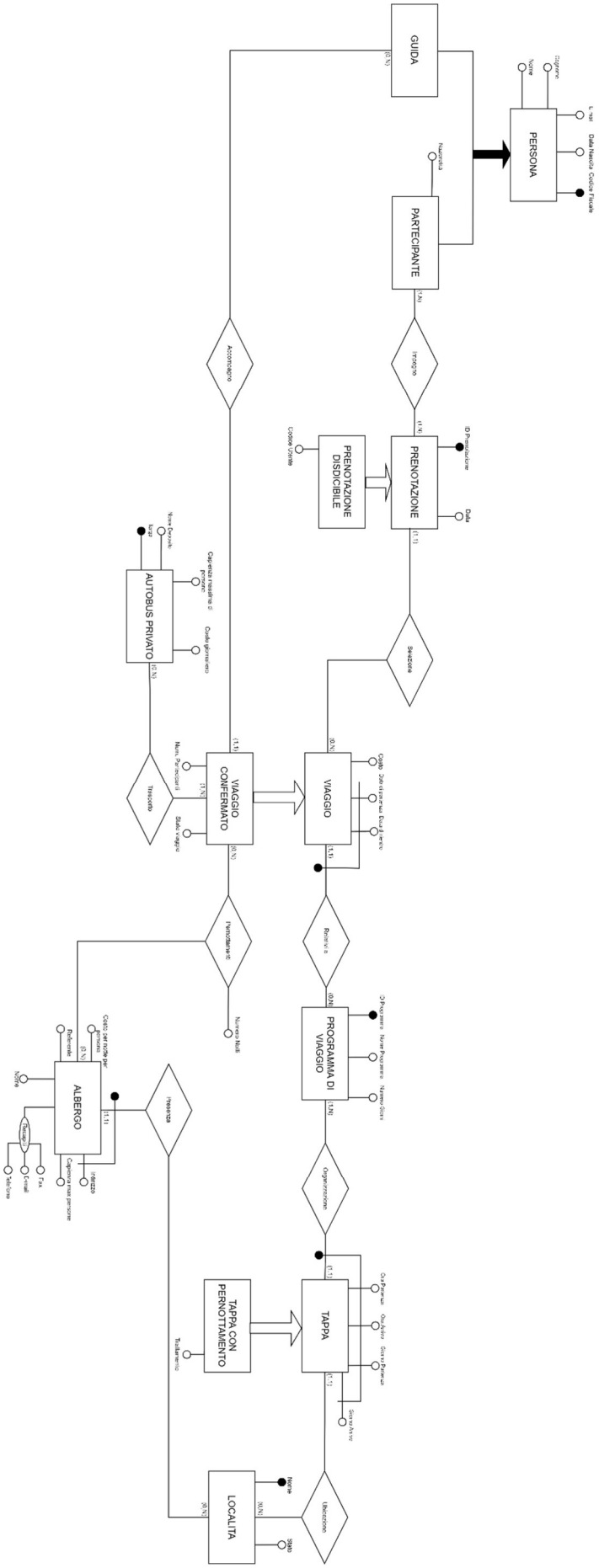
Dunque, ottengo:



Non sono presenti conflitti sui nomi.

Infine ad alcune delle relazioni e attributi è stato cambiato il nome al fine di rendere più comprensibile l'intero schema E-R. Lo schema finale risultante è:

(vedere pagina successiva)


GUIDA
PERSONA
PARTECIPANTE
Accompagno
Impegno
PRENOTAZIONE
PRENOTAZIONE DISDICIBILE
Selezione
AUTOBUS PRIVATO
VIAGGIO CONFERMATO
VIAGGIO
Tipologia
Prenotazione
Riserva
PROGRAMMA DI VIAGGIO
Organizzazione
ALBERGO
Pernottamento
TAPPA
TAPPA CON PERNOTTAMENTO
Trattamento
LOCALITA
Ubicazione

Regole aziendali

1. Durata Viaggio: agisce sull'attributo "Numero Giorni" dell'entità "Programma Di Viaggio". Il numero di giorni di ogni programma di viaggio deve essere compreso tra 1 e 7.
2. Località Viaggio: agisce sull'attributo "stato" dell'entità "Località". Lo stato delle località visitate deve essere "Italia" oppure un altro stato europeo.
3. Prenotazioni Disdicibili: agisce sull'entità generalizzata dell'entità "Prenotazione". Una prenotazione è disdicibile fino a venti giorni prima della data di partenza del viaggio a cui fa riferimento (associata tramite relazione "Selezione") .
4. Viaggio Confermato: agisce sull'entità generalizzata dall'entità "Viaggio". Un viaggio è considerato confermato quando le prenotazioni di un viaggio sono concluse (ovverosia, 20 giorni prima della data di partenza).
5. Inserimento Viaggio: agisce sull'attributo "Data di Partenza" dell'entità "Viaggio". Quando viene inserito un nuovo viaggio, la data di partenza deve essere successiva di più di 20 giorni dalla data odierna, in modo tale da permettere agli utenti di prenotarsi.
6. Assegnazione Guida: agisce sulla relazione "Accompagno" tra "Viaggio Confermato" e "Guida". Per assegnare una guida ad un viaggio, lo stato del viaggio deve essere "in programma".
7. Assegnazione Guida bis: agisce sulla relazione "Accompagno" tra "Viaggio Confermato" e "Guida". Per assegnare una guida ad un viaggio, la guida non deve essere già stata assegnata (tramite relazione "Accompagno") ad un altro viaggio avente date sovrapposte a quello a cui la guida si vuole assegnare.
8. Assegnazione Autobus Privati: agisce sulla relazione "Trasporto" tra "Viaggio Confermato" e "Autobus Privato". Per assegnare un autobus privato ad un viaggio, lo stato del viaggio deve essere "in programma".
9. Assegnazione Autobus Privati-bis : agisce sulla relazione "Trasporto" tra "Viaggio Confermato" e "Autobus Privato". Per assegnare un autobus privato ad un viaggio, l'autobus non deve essere associato (tramite relazione "Trasporto") ad un altro viaggio avente date sovrapposte a quello a cui l'autobus privato si vuole assegnare.
10. Assegnazione Autobus Privati-tris: : agisce sulla relazione "Trasporto" tra "Viaggio Confermato" e "Autobus Privato". L'insieme degli autobus privati assegnati al viaggio devono avere capacità massima di persone totale (ricavata sommando le singole capacità di ogni autobus privato) maggiore o uguale al numero di partecipanti del viaggio.

11. Assegnazione Alberghi: agisce sulla relazione “Pernottamenti” tra “Viaggio Confermato” e “Albergo”. Per assegnare un albergo ad un viaggio, lo stato del viaggio deve essere “in programma”.
12. Assegnazione Alberghi-bis: agisce sulla relazione “Pernottamenti” tra “Viaggio Confermato” e “Albergo”. Per assegnare un albergo ad un viaggio, il programma di viaggio a lui associato (tramite relazione “Relativi a”) deve possedere una tappa con pernottamento (tramite relazione “Organizzazione”) nella stessa località (tramite relazione “Ubicazione”) nella quale si trova l’albergo (tramite relazione “Presenza”) che si vuole assegnare al viaggio.
13. Assegnazione Alberghi-tris: agisce sulla relazione “Pernottamenti” tra “Viaggio Confermato” e “Albergo”. L’albergo assegnato al viaggio deve essere unico per ogni tappa ed avere capacità massima di persone maggiore o uguale al numero di partecipanti del viaggio.
14. Date Viaggio: agisce sugli attributi “Data di partenza” e “Data di rientro” dell’entità “Viaggio”. La data di rientro deve essere successiva alla data di partenza e il numero di giorni che intercorrono tra la data di partenza e la data di rientro devono essere pari al numero di giorni previsti dal programma di viaggio a cui fa riferimento.
15. Date Tappa :agisce sugli attributi “Giorno Partenza” e “Giorno Arrivo” dell’entità “Tappa”. Il giorno di partenza deve essere maggiore o uguale al giorno di arrivo. Sia il giorno di partenza che il giorno di arrivo devono essere minori o uguali del numero di giorni previsti dal programma di viaggio a cui fanno riferimento.
16. Date Tappa Pernottamento: agisce sugli attributi “Giorno Partenza” e “Giorno Arrivo” dell’entità “Tappa con pernottamento”. Il giorno di partenza deve essere strettamente maggiore del giorno di arrivo.
17. Stato Viaggio : agisce sull’attributo “stato viaggio” dell’entità “Viaggio Confermato”. Un viaggio confermato è “terminato” se la sua data di rientro è passata. Un viaggio confermato è invece è “attivo” se la sua data di partenza è passata ma la sua data di rientro è successiva alla data odierna. Altrimenti, un viaggio confermato è “in programma”.
18. Generazione Report: un report può essere generato solo per viaggio confermato terminato. Il costo delle guide non concorre al calcolo del guadagno di un viaggio. Il guadagno viene ricavato moltiplicando il costo del viaggio per il numero dei partecipanti e sottraendo da questa quantità il costo totale degli autobus privati (somma dei costi giornalieri di ogni autobus assgenato moltiplicato per il numero di giorni) e il costo totale degli alberghi (somma dei costi per notte per persona moltiplicati per il numero di participanti e per il numero di notti previste per lo specifico pernottamento).

Dizionario dei dati

Entità	Descrizione	Attributi	Identificatori
Persona	Entità usata per mantenere informazioni personali	Codice Fiscale Data Nascita E-mail Nome Cognome	Codice Fiscale
Partecipante	Entità figlia di Persona, usata per mantenere le informazioni dei partecipanti ai viaggi.	Codice Fiscale Nazionalità Data Nascita E-mail Nome Cognome	Codice Fiscale
Guida	Entità figlia di Persona, usata per mantenere le informazioni delle guide che lavorano per l'agenzia.	Codice Fiscale Data Nascita E-mail Nome Cognome	Codice Fiscale
Prenotazione	Entità usata per mantenere le informazioni delle prenotazioni dei clienti ai viaggi.	ID Prenotazione Data	ID Prenotazione
Prenotazione Disdicibile	Entità figlia di Prenotazione. In particolare, rappresenta una prenotazione che può essere disdetta.	ID Prenotazione Data Codice Utente	ID Prenotazione
Programma di Viaggio	Entità usata per mantenere le informazioni sui programmi su cui i viaggi si basano.	ID Programma Numero Giorni Nome Programma	ID Programma
Località	Entità usata per rappresentare una località di una tappa del programma di viaggio.	Nome Stato	Nome
Tappa	Entità usata per rappresentare l'organizzazione di una tappa del programma di viaggio.	Ora Partenza Ora Arrivo Giorno Partenza Giorno Arrivo	Ora Partenza Ora Arrivo Giorno Partenza Giorno Arrivo Programma Di Viaggio
Tappa con Pernottamento	Entità figlia di Tappa. È una particolare tappa che prevede il pernottamento.	Ora Partenza Ora Arrivo Giorno Partenza Giorno Arrivo Trattamento	Ora Partenza Ora Arrivo Giorno Partenza Giorno Arrivo Località

Viaggio	Entità usata per rappresentare un viaggio offerto dall'agenzia che usa il sistema.	Costo Data di Partenza Data di Rientro	Data di Partenza Data di Rientro Programma di Viaggio
Viaggio Confermato	Entità figlia di Viaggio. Rappresenta un viaggio le cui prenotazioni sono concluse ed a cui è possibile assegnare un insieme di Autobus Privati, di Alberghi e una Guida.	Costo Num. Partecipanti Data di Partenza Data di Rientro	Data di Partenza Data di Rientro Programma di Viaggio
Autobus Privato	Entità che rappresenta un autobus privato per il trasporto dei partecipanti di un viaggio	Targa Capienza massima di persone Costo giornaliero Nome Deposito	Targa
Albergo	Entità che rappresenta un albergo per consentire il pernottamento dei partecipanti di un viaggio	Costo per notte per persona Referente Nome Recapiti(fax, e-mail, telefono) Capienza max persone Indirizzo	Indirizzo Località

4.

4. Progettazione logica

Volume dei dati

Concetto nello schema	Tipo[1]	Volume atteso
Programma di Viaggio	E	250
Viaggio	E	1.000 (in media 4 viaggi per ogni programma di viaggio)
Viaggio Confermato	E	750 (cioè 250 viaggi sono "prenotabili", inoltre per i viaggi confermati, che abbiamo detto si distinguono in "terminati", "in programma" e "attivi")
Relativi a	R	1.000
Persona	E	60.075
Guida	E	75
Accompagno	R	750
Partecipante	E	60.000 (in media 60 partecipanti per ogni viaggio)
Prenotazione	E	20.000 (in media prenotazioni da 3 partecipanti l'una e 20 prenotazioni per ogni viaggio)
Prenotazioni Disdicibili	E	5.000
Impegno	R	60.000
Selezione	R	20.000
Tappa	E	1.000 (in media 4 tappe per ogni programma di viaggio)
Tappa con Pernottamento	E	500 (in media 1 tappa su 2 prevede il pernottamento)
Organizzazione	R	1.000
Località	E	500
Ubicazione	R	1.000
Albergo	E	1.500 (in media 3 alberghi per ogni località)
Presenza	R	1.500

[1] Indicare con E le entità, con R le relazioni

Pernottamenti	R	1.500 (in media 2 alberghi per ogni viaggio)
Autobus Privato	E	150
Trasporto	R	1.500 (in media 2 autobus privati per ogni viaggio)

Tavola delle operazioni

Cod.	Descrizione	Frequenza attesa
OP 1	Aggiungi un nuovo programma di viaggio con le relative tappe	1/giorno
OP 2	Aggiungi un nuovo viaggio	4/giorno
OP 3	Aggiungi una nuova località	20/anno
OP 4	Aggiungi una nuova guida	10/anno
OP 5	Aggiungi un nuovo autobus privato	10/anno
OP 6	Aggiungi un nuovo albergo	5/mese
OP 7	Per un programma di viaggio, trova l'elenco delle tappe da cui è composto con tutti i relativi dati	10.000/giorno
OP 8	Per un programma di viaggio, trova tutti i viaggi relativi disponibili con la relativa data di partenza e di rientro.	4.000/giorno
OP 9	Prenota un viaggio	40/giorno
OP 10	Disdici prenotazione di un viaggio	10/giorno
OP 11	Assegnazione alberghi ad un viaggio	1/giorno
OP 12	Assegna una guida ad un viaggio	1/giorno
OP 13	Assegnazione autobus privati a un viaggio	1/giorno
OP 14	Per un viaggio terminato, genera un report	5/settimana
OP 15	Per tutti i viaggi del sistema, verifica se mancano 20 giorni alla partenza ed eventualmente segna il viaggio come confermato e le prenotazioni come concluse (non disdicibili).	1/giorno
OP 16	Per tutti i viaggi confermati del sistema, verifica lo stato del viaggio ed eventualmente aggiornalo; se è trascorso un anno dalla data di rientro di un viaggio, elimina i dati relativi a quel viaggio	1/giorno

	terminato	

Costo delle operazioni

OP 1: Aggiungi un nuovo programma di viaggio con le relative tappe , con frequenza 1/giorno.

Si consideri che abbiamo stimato che in media ogni programma di viaggio sia organizzato in 4 tappe e che 1 tappa su 2 preveda il pernottamento.

Concetto	Costrutto	Accessi	Tipo
Programma di Viaggio	E	1	S
Organizzazione	R	4	S
Tappa	E	4	S
Tappa con pernottamento	E	2	S
Ubicazione	R	4	S

Costo OP 1: (1S * 1) + (4S * 1) + (4S * 1) + (2S * 1) + (4S * 1) = 30 /giorno

OP 2: Aggiungi un nuovo viaggio, con frequenza 4/giorno.

Concetto	Costrutto	Accessi	Tipo
Viaggio	E	1	S
Relativo a	R	1	S

Costo OP 2**:** (1S * 4) + (1S * 4) = 16/giorno

OP 3: Aggiungi una nuova località, con frequenza 20/anno.

Si osservi come qusta operazione differsica dal creare una nuova tappa ma registri solamente una nuova località nel sistema informativo, a cui poi potranno far riferimento tappe e/o alberghi.

Concetto	Costrutto	Accessi	Tipo
Località	E	1	S

Costo OP 3: (1S * 20) = 40/anno

OP 4: Aggiungi una nuova guida, con frequenza 10/anno

Si osservi come questa operazione differisca dall'assegnamento di una guida ad un viaggio, riguarda invece il registrare una nuova guida nel sistema informativo, che sarà poi assegnabile ad un viaggio.

Concetto	Costrutto	Accessi	Tipo

Persona	E	1	S
Guida	E	1	S

Costo OP 4: (1S * 10) + (1S * 10) = 40/anno

OP 5: Aggiungi un nuovo autobus privato, con frequenza 10/anno

Si osservi come questa operazione differisca dall'assegnamento di un autobus privato ad un viaggio, riguarda invece il registrare un nuovo auotobus privato nel sistema informativo, che sarà poi assegnabile ad un viaggio.

Concetto	Costrutto	Accessi	Tipo
Autobus Privato	E	1	S

Costo OP 5: (1S * 10) = 20/anno

OP 6: Aggiungi un nuovo albergo, con frequenza 5/mese

Si osservi come questa operazione differisca dall'assegnamento di un albergo ad un viaggio, riguarda invece il registrare un nuovo albergo nel sistema informativo, che sarà poi assegnabile ad un viaggio.

Concetto	Costrutto	Accessi	Tipo
Albergo	E	1	S
Presenza	R	1	S

Costo OP 6: (1S * 5) + (1S * 5) =20/mese

OP 7: Per un programma di viaggio, trova l'elenco delle tappe da cui è composto con tutti i relativi dati , con frequenza 10.000/giorno.

Si consideri che abbiamo stimato che in media ogni programma di viaggio è organizzato in 4 tappe e che 1 tappa su 2 preveda il pernottamento.

Concetto	Costrutto	Accessi	Tipo
Programma di viaggio	E	1	L
Organizzazione	R	4	L
Tappa	E	4	L
Tappa con pernottamento	E	2	L
Ubicazione	R	4	L
Località	E	4	L

Costo OP 7: (1L * 10.000) + (4L * 10.000) + (4L*10.000) + (2L * 10.000) + (4L*10.000) + (4L*10.000) = 190.000/giorno

OP 8: Per un programma di viaggio, trova tutti i viaggi relativi disponibili con la relativa data di partenza e di rientro, con frequenza 4.000/giorno.

Si consideri che abbiamo stimato che in media ci siano 4 viaggi relativi ad ogni programma di viaggio.

Si osservi inoltre che i viaggi mostrati sono tutti e soli i viaggi a cui è possibile ancora prenotarsi, cioè non confermati e ai quali mancano quindi più di 20 giorni alla partenza. Quindi, avendo supposto una distribuzione uniforme, avremo che in media un solo viaggio è nello stato "prenotabile".

Concetto	Costrutto	Accessi	Tipo
Programma di viaggio	E	1	L
Relativi a	R	1	L
Viaggio	E	1	L

Costo OP 8: (1L * 4.000) + (1L * 4.000) + (1L*4.000) =12.000/giorno

OP 9: Prenota un viaggio, con frequenza 40/giorno.

Si consideri che abbiamo stimato che in ogni prenotazione vengano coinvolti in media 3 partecipanti.

Concetto	Costrutto	Accessi	Tipo	Spiegazione
Viaggio	E	1	L	Leggo il costo del viaggio
Prenotazione	E	1	S	
Prenotazione Disdicibile	E	1	S	Appena avvenuta, ogni prenotazione è da considerarsi disdicibile.
Impegno	R	3	S	
Partecipante	E	3	S	
Persona	E	3	S	
Selezione	R	1	S	

Costo OP 9: (1L*40)+ (1S * 40) + (1S * 40) + (3S * 40) + (3S * 40) + (3S * 40) + (1S * 40) = 1.000/giorno

OP 10: Disdici una prenotazione di un viaggio. Con frequenza 10/giorno.

Si consideri che abbiamo stimato che in ogni prenotazione vengano coinvolti in media 3 partecipanti. Si osservi come durante questa operazione si è deciso di non eliminare i dati relativi ai partecipanti ma di mantenerli nel sistema informativo; nel caso in cui questi dovessero prenotarsi nuovamente ad un viaggio i loro dati sarebbero già stati registrati.

Concetto	Costrutto	Accessi	Tipo	Spiegazione
Prenotazione Disdicibile	E	1	L	Leggo codice per disdire e verifico sia corretto
Prenotazione Disdicibile	E	1	S	
Prenotazione	E	1	L	
Prenotazione	E	1	S	
Impegno	R	3	S	
Selezione	R	1	S	

Costo OP 10: (1L * 10) + (1S * 10) + (1L * 10) + (1S * 10) + (3S * 10) + (1S * 10) = 140/giorno

OP 11: Assegnazione alberghi ad un viaggio, con frequenza 1/giorno.

Si consideri che abbiamo stimato di avere in media 4 tappe per programma di viaggio, di cui 2 con pernottamento e 3 alberghi per località.

Si osservi inoltre che per far si che l'assegnamento sia corretto è richiesto sapere il numero di partecipanti al viaggio e le date di partenza e di rientro, poi consultare l'elenco di tutti alberghi presenti in una data località.

Concetto	Costrutto	Accessi	Tipo	Spiegazione
Viaggio	E	1	L	
Viaggio Confermato	E	1	L	Leggo il numero di partecipanti al viaggio e ne verifico lo stato
Relativi a	R	1	L	
Programma di Viaggio	E	1	L	
Organizzazione	E	4	L	
Tappa	E	4	L	
Tappa con Pernottamento	E	2	L	Leggo le tappe con pernottamento
Ubicazione	R	2	L	
Località	E	2	L	Leggo le località nelle quali è previsto il pernottamento
Presenza	R	6	L	

Albergo	E	6	L	Leggo gli alberghi disponibili in ogni località e la relativa capienza
Pernottamenti	R	2	S	

Costo OP 11: (1L * 1) + (1L * 1) + (1L * 1) + (1L * 1) + (4L * 1) + (4L * 1) + (2L * 1)+ (2L * 1) + (2L * 1) + (6L * 1) + (6L * 1) + (2S * 1)= 34/giorno

OP 12: Assegnazione guida ad un viaggio, con frequenza 1/giorno.

Si consideri che abbiamo stimato che nel sistema siano presenti in media 75 guide e che siano presenti 750 viaggi confermati, di cui supponiamo 250 terminati, quindi dei non risulta utile verificare eventuali collisioni.

Si osservi che per far si che l'assegnamento sia corretto è richiesto leggere i dati di tutte le guide e sapere se queste sono state già assegnate ad altri viaggi aventi date sovrapposte a quello a cui la guida si vuole assegnare, è quindi opportuno analizzare i vari assegnamenti e le relative date dei viaggi.

Concetto	Costrutto	Accessi	Tipo	Spiegazione
Persona	E	75	L	
Guida	E	75	L	Leggo le guide presenti all'interno del sistema
Accompagno	R	500	L	
Viaggio	E	500	L	Leggo le date dei viaggi a cui le guide sono state già assegnate
Accompagno	R	1	S	

Costo OP 12: (75L * 1) + (75L * 1) + (500L * 1) + (500L * 1) + (1S * 1) = 1152/giorno

OP 13: Assegnazione autobus privati a un viaggio, con frequenza 1/giorno.

Si consideri che abbiamo stimato che nel sistema siano presenti in media 75 guide e che siano presenti 750 viaggi confermati, di cui supponiamo 250 terminati, quindi dei non risulta utile verificare eventuali collisioni.

Si osservi che per far si che l'assegnamento sia corretto è richiesto leggere i dati di tutte gli autobus privati e sapere se questi sono stati già assegnati ad altri viaggi aventi date sovrapposte a quello a cui si vogliono assegnare, è quindi opportuno analizzare i vari assegnamenti e le relative date dei viaggi.

Ricordiamo infine di aver supposto di avere in media necessità di 2 Autobus Privati per viaggio.

Concetto	Costrutto	Accessi	Tipo	Spiegazione
Viaggio	E	1	L	
Viaggio Confermato	E	1	L	Leggo il numero di partecipanti al viaggio e ne verifico lo stato
Autobus Privato	E	150	L	Leggo gli autobus privati presenti all'interno del sistema
Trasporto	E	1.000	L	
Viaggio Confermato	E	500	L	
Viaggio	E	500	L	Leggo le date dei viaggi a cui gli autobus privati sono stati già assegnati
Trasporto	R	2	S	

Costo OP 13: (1L*1) + (1L * 1) + (150L * 1) + (1.000L * 1) + (500L * 1) + (500L * 1) +(2S * 1) = 2.156/giorno

OP 14 : Per un viaggio terminato, genera un report, con frequenza 5/settimana

Si consideri che abbiamo stimato di avere in media 20 prenotazioni per ogni viaggio, quindi in media 60 partecipanti per viaggio. Per quanto riguarda il calcolo del guadagno netto ricordiamo le seguenti assunzioni:

-si è stimato di avere in media 2 pernottamenti per ogni viaggio, il costo totale di ogni pernottamento si ricava moltiplicando il costo totale per notte, dato dal costo per notte per persona moltiplicato per il numero di persone, per il numero di notti per cui è previsto il pernottamento.

-si è stimato di avere in media 2 Autobus Privati per ogni viaggio, il costo totale di ogni Autobus Privato si ricava moltiplicando il costo giornaliero per il numero di giorni della durata del viaggio; Infatti, si è assunto che ogni autobus privato si assegnato per tutta la durata di un viaggio.

-si è considerato il costo della guida non rientrante nel calcolo; Infatti si è assunto che le guide che lavorano per l'agenza non abbiano un contratto a chiamata ma uno stipendio fisso che fa si che non concorra nel calcolo.

Concetto	Costrutto	Accessi	Tipo	Spiegazione
Viaggio Confermato	E	1	L	Leggo lo stato del viaggio e controllo sia terminato
Viaggio	E	1	L	
Relativi a	R	1	L	

Programma di Viaggio	E	1	L	Leggo il numero di giorni
Selezione	R	20	L	
Prenotazione	E	20	L	
Impegno	R	60	L	
Partecipante	E	60	L	Leggo le informazioni di ogni partecipante
Persona	E	60	L	Leggo le informazioni di ogni partecipante
Pernottamenti	R	2	L	Leggo il numero di notti previste per il pernottamento
Albergo	E	2	L	Leggo costo per notte
Trasporto	E	2	L	
Autobus Privato	E	2	L	Leggo costo giornaliero

Costo OP 14: (1L * 5) + (1L * 5) +(1L * 5) + (1L * 5) + (20L * 5) + (20L * 5) + (60L * 5) + (60L * 5) + (60L * 5) + (2L * 5) + (2L * 5) + (2L * 5) + (2L * 5) = 1.160/settimana

OP 15: Per tutti i viaggi del sistema, verifica se mancano 20 giorni alla partenza ed eventualmente segna il viaggio come confermato e le prenotazioni come concluse (non disdicibili), con frequenza 1/giorno.

Si ricordi che abbiamo stimato di avere 1.000 viaggi nel nostro sistema informativo di cui 750 sono confermati, dunque abbiamo 250 viaggi che non sono confermati (prenotabili) e di cui va controllato il numero di giorni mancanti alla partenza; supponiamo che ogni giorno 1 viaggio raggiunga la soglia dei 20 giorni. Infine, ricordiamo di aver stimato di avere in media 20 prenotazioni per ciascun viaggio, ognuna delle quali in media di 3 partecipanti l'una (in media 60 partecipanti per ogni viaggio)

Concetto	Costrutto	Accessi	Tipo	Spiegazione
Viaggio	E	250	L	Leggo la data di partenza e verifico se mancano 20 giorni alla partenza
Selezione	R	20	L	
Prenotazione	E	20	L	
Prenotazione Disdicibile	E	20	L	
Impegno	R	60	L	Leggo il numero di partecipanti
Prenotazione Disdicibile	E	20	S	Elimino le prenotazioni disdicibili
Prenotazione	E	20	S	
Impegno	R	60	S	
Viaggio	E	1	S	
Viaggio Confermato	E	1	S	Scrivo il numero di partecipanti e lo stato del

				viaggio "in programma"
Selezione	R	20	S	
Relativi a	R	1	S	

Costo OP 15: (250L * 1) + (20L * 1) + (20L * 1) + (20L * 1) + (60L * 1) + (20S * 1) + (20S * 1) + (60S * 1) + (1S * 1) + (1S * 1) + (20S * 1) + (1S * 1) = 616/giorno

OP 16: Per tutti i viaggi confermati del sistema, verifica lo stato del viaggio ed eventualmente aggiornalo; se è trascorso un anno dalla data di rientro di un viaggio, elimina i dati relativi a quel viaggio terminato, con frequenza 1/giorno.

Si ricordi che abbiamo stimato di avere 750 viaggi confermati nel sistema; inoltre supponiamo che ogni giorno 1 viaggio confermato necessiti di aggiornare il proprio stato e che 1 viaggio si trovi nella condizione tale da eliminare i dati relativi ad esso (trascorso un anno dalla data di rientro).

Si osservi che tramite questa operazione stiamo supponendo che non sia possibile generare un report per viaggio terminati dalla cui data di partenza sia trascorso più di un anno, poiché questi vengono eliminati

Concetto	Costrutto	Accessi	Tipo	Spiegazione
Viaggio	E	750	L	Leggo la data di partenza e di rientro
Viaggio Confermato	E	750	L	Leggo stato viaggio e verifico se l'attuale stato è quello corretto.
Viaggio Confermato	E	1	S	Supponendo che 3 viaggi richiedano un cambiamento, li aggiorno
Viaggio Confermato	E	1	S	Supponendo che solo 1 viaggio debba essere eliminato, lo elimino.
Viaggio	E	1	S	

Costo OP 16: (750L * 1) + (750L * 1) + (3S * 1) + (1S * 1) + (1S * 1) = 1.510/giorno

Ristrutturazione dello schema E-R

Analisi delle ridondanze:

La prima ed unica ridondanza presente nel nostro schema E-R è quella che coinvolge le entità "Viaggio Confermato" e "Prenotazione", nello specifico l'attributo "Num. Partecipanti" è un attributo derivabile dalle relazioni "Selezioni" e "Impegno" attraverso un operazione di conteggio di

occorrenze. Infatti, il numero dei partecipanti di un viaggio si può ottenere sommando il numero di persone coinvolte (tramite relazione "Impegno") in ogni prenotazione che fa riferimento (tramite relazione "Selezione") a quel viaggio. Risulta ora opportuno analizzare i costi delle differenti soluzioni, di quella con ridondanza e non, al fine di guidare la scelta finale. Innanzitutto, le operazioni che coinvolgono l'attributo "Num. Partecipanti" sono:

OP 11: Assegnazione alberghi ad un viaggio, con frequenza 1/giorno.

OP 13: Assegnazione autobus privati a un viaggio, con frequenza 1/giorno.

OP 15: Per tutti i viaggi del sistema, verifica se mancano 20 giorni alla partenza ed eventualmente segna il viaggio come confermato e le prenotazioni come concluse (non disdicibili), con frequenza 1/giorno.

Osserviamo che OP 14 non rientra in questo calcolo, anche se necessita di sapere il numero di partecipanti, poiché comunque deve accedere in lettura alle informazioni dei partecipanti.

Costo della soluzione **con ridondanza:**

Operazione	Frequenza	Accessi	Costo
OP 11	1/giorno	30L+2S	34/giorno
OP 13	1/giorno	2.152L+2S	2.156/giorno
OP 15	1/giorno	370L + 123S	616/giorno

Costo totale con ridondanza 3.538 /giorno

L'analisi più specifica di queste operazioni è stata descritta sopra, nella sezione "Costo delle operazioni".

Ora andiamo ad analizzare il costo della soluzioni senza ridondanza, quindi supponendo di non avere l'attributo "Num. Partecipanti" nell'entità "Viaggio Confermato".

Si osservi che, senza ridondanza : OP 11 aumenta il suo costo, in quanto leggeva direttamente l'attributo dall'entità viaggio confermato; OP 13 aumenta il suo costo, in quanto leggeva direttamente l'attributo dall'entità viaggio confermato; OP 15 diminuisce il suo costo, in quanto non si deve più preoccupare di aggiornare il numero di partecipanti.

Costo della soluzione **senza ridondanza**:

Ricordiamo di aver stimato di avere in media 20 prenotazioni per ciascun viaggio, ognuna delle quali ha in media di 3 partecipanti l'una (in media 60 partecipanti per ogni viaggio)

OP 11:

Concetto	Costrutto	Accessi	Tipo	Spiegazione
Viaggio	E	1	L	
Viaggio Confermato	E	1	L	Non posso più leggere direttamente il numero di partecipanti
Selezione	R	20	L	Comporta aumento di costo nel caso senza ridondanza
Prenotazione	E	20	L	Comporta aumento di costo nel caso senza ridondanza
Impegno	R	60	L	Comporta aumento di costo nel caso senza ridondanza
Relativi a	R	1	L	
Programma di Viaggio	E	1	L	
Organizzazione	E	4	L	
Tappa	E	4	L	
Tappa con Pernottamento	E	2	L	
Ubicazione	R	2	L	
Località	E	2	L	
Presenza	R	6	L	
Albergo	E	6	L	
Pernottamenti	R	2	S	

OP 11 aumenta il suo costo di (20L * 1) + (20L * 1) + (60L * 1) = 100/giorno

Con un ragionamento analogo, OP 13 aumenta il suo costo di (20L * 1) + (20L * 1) + (60L * 1) = 100/giorno

OP 15:

Concetto	Costrutto	Accessi	Tipo	Spiegazione
Viaggio	E	250	L	
~~Selezione~~	~~R~~	~~20~~	~~L~~	Non necessito più di calcolarmi il numero di partecipanti nella soluzione senza ridondanza.
~~Prenotazione~~	~~E~~	~~20~~	~~L~~	Non necessito più di calcolarmi il numero di partecipanti nella soluzione senza ridondanza.
~~Prenotazione Disdicibile~~	~~E~~	~~20~~	~~L~~	Non necessito più di calcolarmi il numero di partecipanti nella soluzione senza ridondanza.

~~Impegno~~	~~R~~	~~60~~	~~L~~	Non necessito più di calcolarmi il numero di partecipanti nella soluzione senza ridondanza.
Prenotazione Disdicibile	E	20	S	
Prenotazione	E	20	S	
Impegno	R	60	S	
Viaggio	E	1	S	
Viaggio Confermato	E	1	S	
Selezione	R	20	S	
Relativi a	R	2	S	

Quindi, OP 15 diminuisce il suo costo di (120L * 1) = 120/giorno

Mettendo le informazioni insieme nella soluzione senza ridondanza abbiamo un bilancio di +100+100-120 = 80/giorno operazioni in più.

Quindi, avendo tenuto in considerazione le stime sui volumi dei dati e sulla frequenza delle operazioni svolte, si è deciso per il mantenimento di questa ridondanza, anche se porta un guadagno minimo.

Sucessivamente ho valutato se inserire appositamente una ridondanza, nello specifico un attributo "guadagno viaggio" nell'entità "Viaggio Confermato" che renda **OP 14** ("Per un viaggio terminato, genera un report, con frequenza 5/settimana") più snella. Questo richiederebbe di appesantire le operazioni: OP 11, OP 13 e OP 22; Infatti, ognuna di queste dovrebbe aggiornare l'attributo sottraendo/sommando i costi/guadagni tenuti in considerazioni nella operazioni. Siccome OP 21 non è stata stimata essere un operazione così frequente come lo sono OP 11, OP 13 e OP 22 si è concluso che questa ridondanza non avrebbe portato alcun guadagno e quindi si è proceduto per non introdurla.

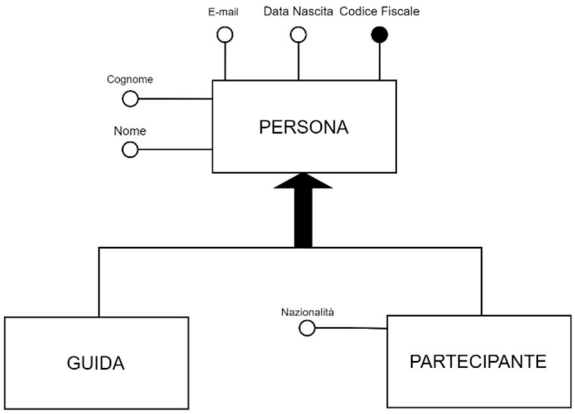
Eliminazione delle generalizzazioni:

Iniziamo dalla gerarchia presente è tra "Persona", "Guida" e "Partecipante".

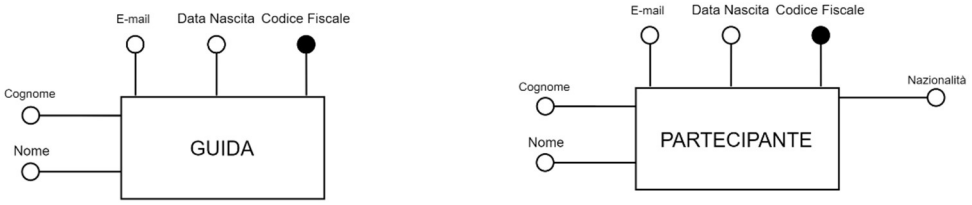
Considerando che la generalizzazione è totale , e che le operazioni precedentemente elencate si riferiscono solo a occorrenze di "Guida" oppure di "Partecipante", si è deciso di adottare una ristrutturazione di tipo "Accorpamento del genitore della generalizzazione nelle figlie" .

Questo comporta che l'entità "Persona" venga eliminata e per l'ereditarietà i suoi attributi vengano aggiunti alle entità figlie.

Lo schema ER prima della ristrutturazione:



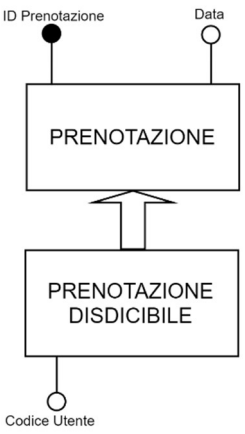
Lo schema ER dopo la ristrutturazione :



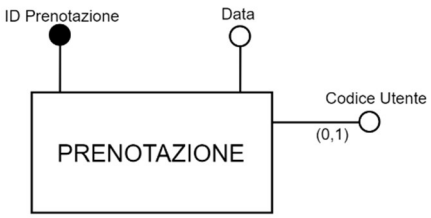
Consideriamo ora la seconda gerarchia presente, tra “Prenotazione” e “Prenotazione Disdicibile”. In questo caso le operazione non fanni distinzione tra le occorrenze di “Prenotazione” e di “Prenotazione Disdicibile” e quindi si è deciso di adottare una ristrutturazione di tipo “Accorpamento delle figlie della generalizzazione nel genitore”.

Qusto comporta che l’entità “Prenotazione Disdicibile” venga eliminata e per l’ereditarietà il suo unico attributo venga aggiunto all’entità padre.

Lo schema ER prima della ristrutturazione:



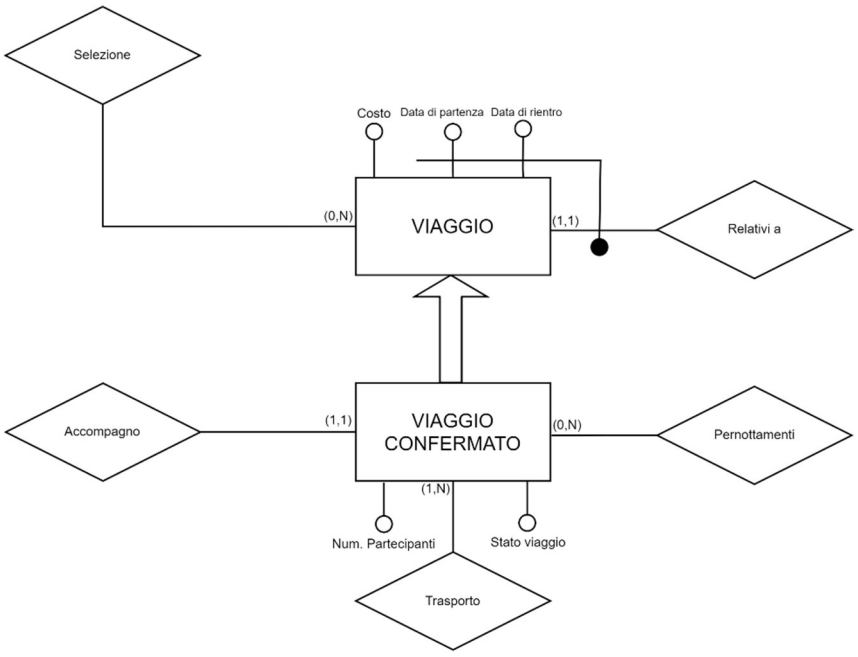
Lo schema ER dopo la ristrutturazione:



Per questa ristrutturazione distinguiamo il tipo di un occorrenza basandondoci sull'attributo "codice per disdire" ed assumiamo che qual'ora questo valore sia nullo la prenotazione sia non disdicibile, e quindi concorrerà al calcolo del numero dei partecipanti di un viaggio.

La terza gerarchia presente che andiamo a considerare è quella tra "Viaggio" e "Viaggio confermato". Per questo caso si è deciso di adottare ancora una volta una ristrutturazione di tipo "Accorpamento delle figlie della generalizzazione nel genitore".

Lo schema ER prima della ristrutturazione:

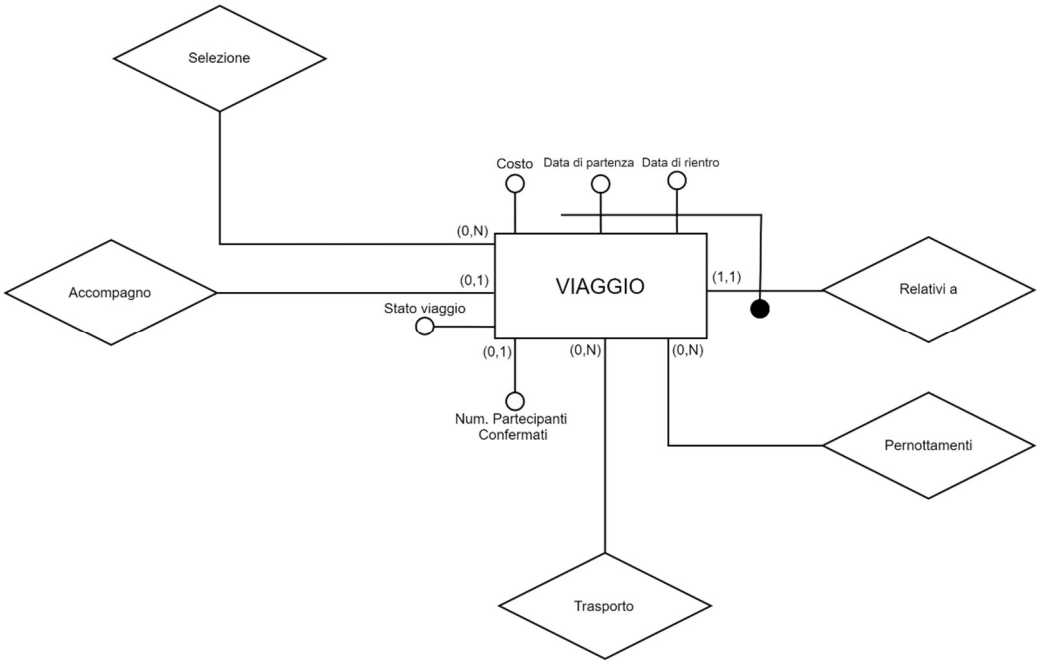


In questo caso si è deciso di utilizzare un attributo già presente nell'entità "Viaggio Confermato", ovvero "Stato Viaggio", al fine di distinguere il tipo di un occorrenza dell'entità accorpata; all'attributo, che prima della ristrutturazione poteva assumere i valori "in programma", "attivo" e "terminato", aggiungiamo la possibilità di assumere un nuovo valore "prenotabile" (ricordiamo che un viaggio confermato non è prenotabile, mentre un viaggio non confermato lo è, per questo si è

deciso di adottare questo termine) che evidenzia l'appartenenza di un occorrenza all'entità "Viaggio" (intesa come entità prima della ristrutturazione).

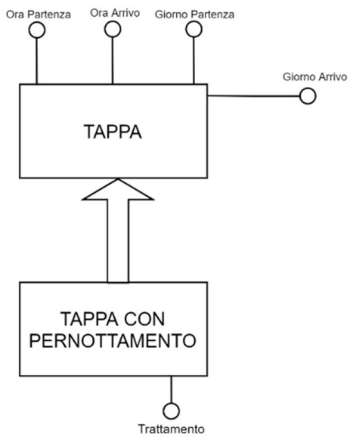
Infine, ho deciso di cambiare il nome all'attributo da "Num. Partecipanti" a "Num. Partecipanti Confermati" solo per una questione di facilitare la comprensione del nostro schema ER.

Lo schema ER dopo la ristrutturazione:



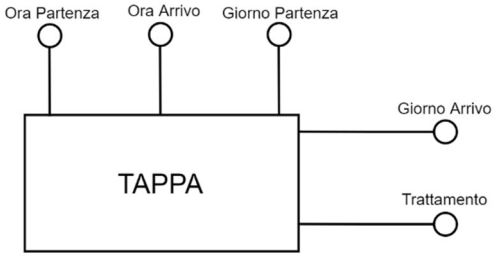
La quarta ed ultima gerarchia presente che andiamo a considerare è quella tra "Tappa" e "Tappa Con Pernottamento". Per questo caso si è deciso di adottare ancora una volta una ristrutturazione di tipo "Accorpamento delle figlie della generalizzazione nel genitore".

Lo schema ER prima della ristrutturazione:



Anche in questo caso si è deciso di utilizzare un attributo già presente nell'entità, ovvero "Trattamento" , al fine di distinguere il tipo di un occorrenza dell'entità accorpata. all'attributo, che prima della ristrutturazione poteva assumere i valori "FB", "HB", "BB" e "OB" , aggiungiamo la possibilità di assumere un nuovo valore "NB" (Not Bed) che evidenzia l'appartenenza di un occorrenza all'entità "Tappa" (senza pernottamento, intesa come entità prima della ristrutturazione).

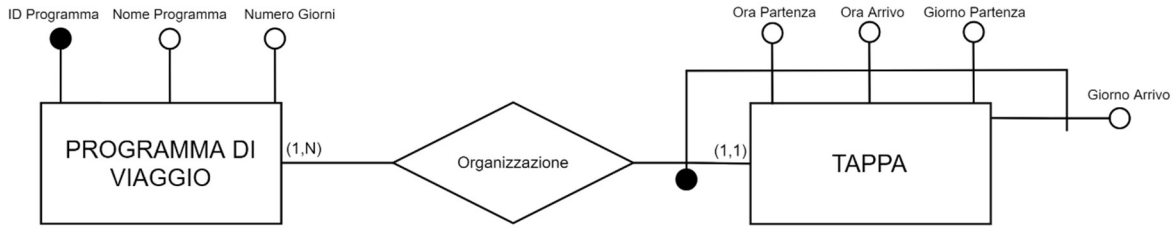
Lo schema ER dopo la ristrutturazione:



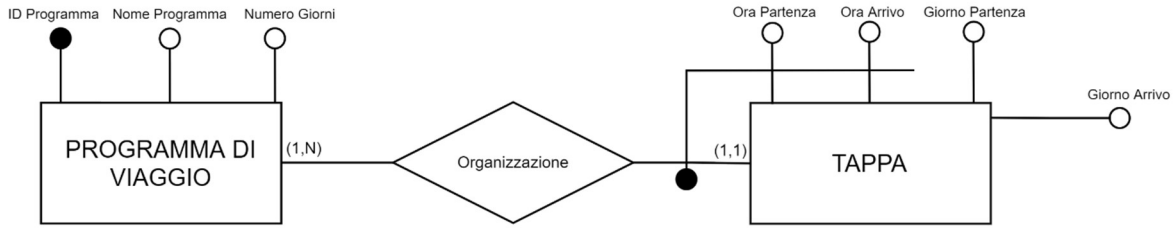
Scelta degli identificatori primari

Sono quelli presenti nel modello E-R proposto. Prima correzione: nell'entità "Tappa" si è deciso solamente di mettere come chiave solamente "Ora Arrivo" e "Giorno Arrivo" e l'ID del programma di viaggio, si è ragionato come queste informazioni fossero sufficienti ad identificare univocamente una tappa di un programma.

Lo schema ER prima del cambiamento dell'identificatore:

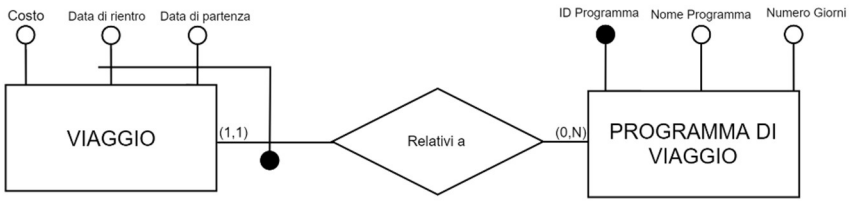


Lo schema ER dopo il cambiamento dell'identificatore:

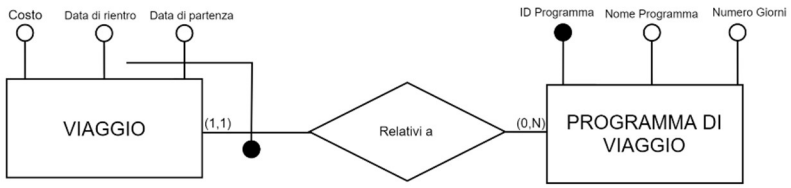


Un ragionamento analogo è stato fatto per l'entità "Viaggio" , si è deciso solamente di mettere come chiave solamente "Data di Partenza" e l'ID del programma di viaggio, si è ragionato come queste informazioni fossero sufficienti ad identificare univocamente una tappa di un programma, basandosi sul fatto che il numero di giorni è comune a tutti i viaggi e che quindi non è necessaria la "data di rientro" ai fini dell'identificazione.

Lo schema ER prima del cambiamento dell'identificatore:



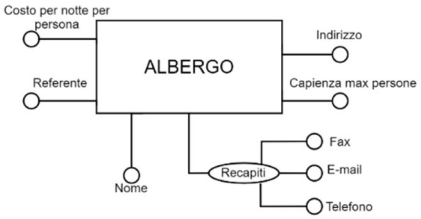
Lo schema ER dopo il cambiamento dell'identificatore:



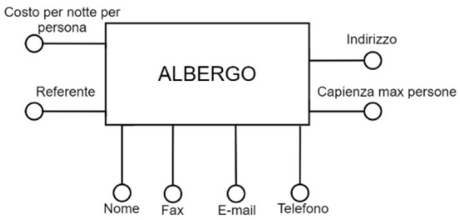
Trasformazione di attributi e identificatori

È opportuno trattare l'eliminazione dell'attributo composto "Recapiti" dell'entità "Albergo".

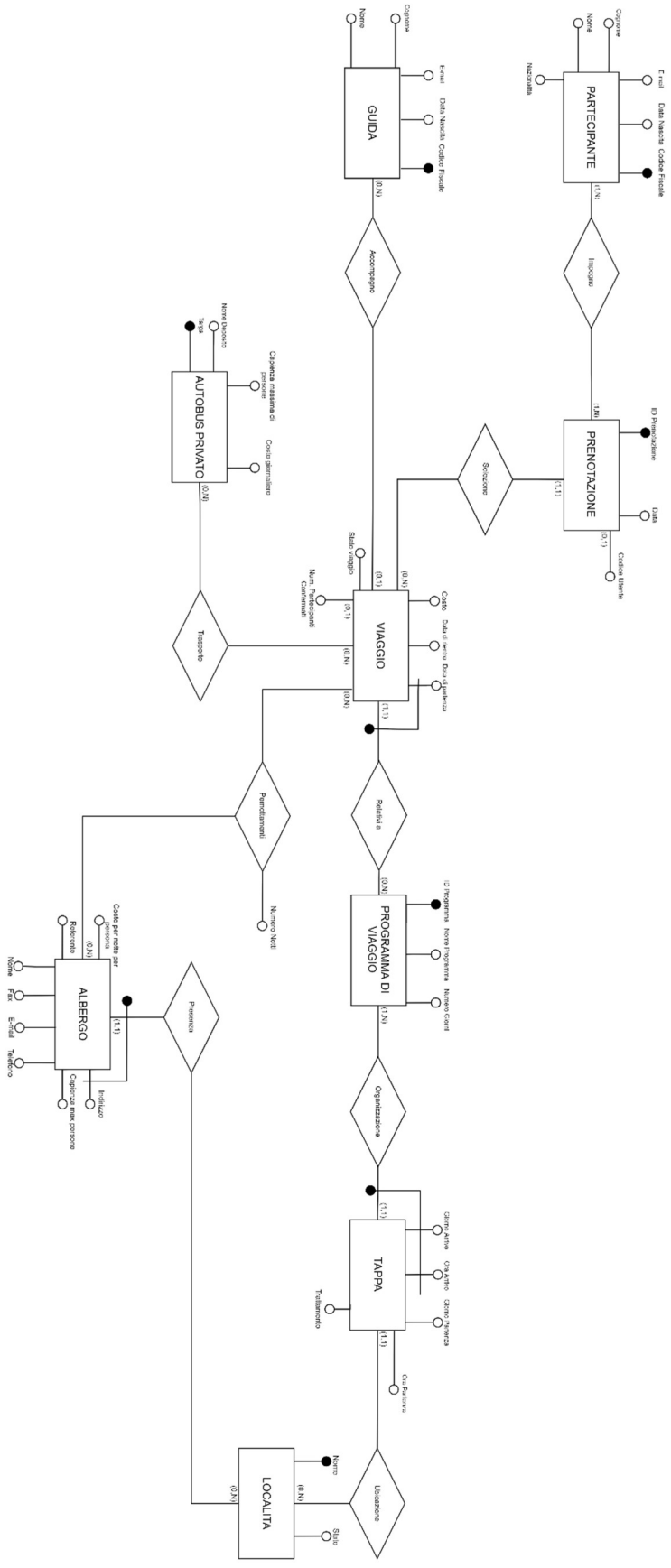
Lo schema ER prima della ristrutturazione:



Lo schema ER dopo la ristrutturazione:



Quindi, lo schema ER che otteniamo dopo la fase di ritrutturazione è il seguente:



Traduzione di entità e associazioni

Partecipante (**Codice Fiscale** , Nome , Cognome , E-mail , Data Nascita , Nazionalità)

Prenotazione (**ID_Prenotazione** , Data , Codice Utente * , Data di Partenza , ID Programma)

Impegno (**Partecipante** , **Prenotazione**)

Programma di Viaggio (**ID Programma** , Numero Giorni , Nome Programma)

Viaggio (**Data_di_Partenza** , **Programma** , Data di rientro , Costo , Stato Viaggio , Num. Partecipanti Confermati *)

Guida (**Codice Fiscale** , Nome , Cognome , E-mail , Data Nascita)

Accompagno (**Viaggio** , CFGuida)

Località (**Nome** , Stato)

Tappa (**Giorno Arrivo** , **Ora Arrivo** , **Programma** , Giorno Partenza , Ora Partenza , Trattamento, NomeLocalità)

Albergo (**Indirizzo** , **NomeLocalità** , Costo per notte per persona , Referente , Nome , Fax, E-mail , Telefono , Capienza max. persone)

Pernottamenti (**Viaggio** , **Albergo** , Numero Notti)

Autobus Privato (**Targa** , Nome Deposito , Capienza massima persone , Costo giornaliero)

Trasporto (**Viaggio**, **AutobusPrivato**)

Con i seguenti vincoli di integrità referenziale:

Prenotazione (Data di Partenza , ID Programma) ⊆ Viaggio (Data di Partenza , ID Programma)

Impegno (Partecipante) ⊆ Partecipante (CodiceFiscale)

Impegno (Prenotazione) ⊆ Prenotazione (ID Prenotazione)

Viaggio (Programma) ⊆ Programma (ID Programma)

Accompagno (Viaggio) ⊆ Viaggio (Data di Partenza , ID Programma)

Accompagno (CFGuida) ⊆ Guida (CodiceFiscale)

Tappa (NomeLocalità) ⊆ Località (Nome)

Tappa (Programma) ⊆ Programma (ID Programma)

Albergo (NomeLocalità) ⊆ Località (Nome)

Pernottamenti (Viaggio) ⊆ Viaggio (Data di Partenza , ID Programma)

Pernottamenti (Albergo) ⊆ Albergo (Indirizzo , NomeLocalità)

Trasporto (Viaggio) ⊆ Viaggio (Data di Partenza , ID Programma)

Trasporto (AutobusPrivato) ⊆ Autobus Privato (Targa)

Normalizzazione del modello relazionale

1 FORMA NORMALE:

- È presente una chiave primaria (tuple non duplicate)
- Non vi sono gruppi di attributi che si ripetono
- Colonne indivisibili (attributi non composti)

2 FORMA NORMALE:

- Nessuna dipendenza parziale (i valori dipendono dalle chiavi minime)

3 FORMA NORMALE:

- Non vi sono dipendenze transitive tra le relazioni

Lo schema proposto rispetta la 1NF, la 2NF e la 3NF.

5. Progettazione fisica

Utenti e privilegi

Gli utenti del sistema sono due: clienti e segreteria. I clienti non hanno bisogno di autenticazione, poiché è ragionevole che questi possano fare indagini di mercato consultando solamente i programmi di viaggi e le relative edizioni dei viaggi a loro associate, e quindi non sarebbe corretto identificare loro tramite username e password. I segretari invece si occupano della creazione e gestione di viaggi; affinchè queste attività possano essere svolte solo da un particolare insieme di utenti autorizzati, è stato ritenuto necessario fornire loro delle credenziali di accesso composte da username e password. I clienti visualizzano le varie offerte dell'agenzia, che constano di un insieme di programmi di viaggio disponibili, di un itineario a loro associato e di un insieme di edizioni di viaggio per quel programma, creati dai segretari, ed eventualmente si prenotano ad un viaggio; i clienti però non hanno l'autorizzazione a conoscere informazioni riguardo gli assegnamenti di auotobus privati / guide / alberghi oppure riguardo i resoconti dei viaggi, che sono invece informazioni gestite dai segretari. Ai clienti che si prenotano, sono associati codici che permettono di ricondursi alla prenotazione da loro effettuata, ed eventualmente disdirla. Quindi, i clienti hanno la possibilità di accedere in lettura alle tabelle: "Programma Di Viaggio", per consultare i programmi di viaggio disponibili; "Tappa" e "Località", per visualizzare l'itinerario di un programma di viaggio; "Viaggio", per conoscere le edizioni di un programma di viaggio disponibili; ed in scrittura alle tabelle "Prenotazione", "Impegno" e "Partecipante" per gestire la prenotazione ad un viaggio e per inserire i dati dei partecipanti alla prenotazione. I segretari invece hanno accesso in scrittura e in lettura a tutte le tabelle del sistema, tranne che a "Prenotazione" , "Impegno" e "Partecipante". Infine, l'utente di login rappresenta il punto di contatto del sistema e permette di accedere al sistema con il minimo dei privilegi necessari.

```
CREATE USER 'loginTA' IDENTIFIED BY 'login';

GRANT EXECUTE ON procedure `agenzia_viaggi`.`login_segretario` TO 'loginTA';
CREATE USER 'segretarioTA' IDENTIFIED BY 'segretario';

GRANT EXECUTE ON procedure `agenzia_viaggi`.`aggiungi_localita` TO 'segretarioTA';
GRANT EXECUTE ON procedure `agenzia_viaggi`.`aggiungi_guida` TO 'segretarioTA';
GRANT EXECUTE ON procedure `agenzia_viaggi`.`aggiungi_albergo` TO 'segretarioTA';
GRANT EXECUTE ON procedure `agenzia_viaggi`.`aggiungi_programma_di_viaggio` TO 'segretarioTA';
GRANT EXECUTE ON procedure `agenzia_viaggi`.`aggiungi_tappa` TO 'segretarioTA';
GRANT EXECUTE ON procedure `agenzia_viaggi`.`aggiungi_autobus_privato` TO 'segretarioTA';
GRANT EXECUTE ON procedure `agenzia_viaggi`.`aggiungi_viaggio` TO 'segretarioTA';
GRANT EXECUTE ON procedure `agenzia_viaggi`.`lista_programmi_di_viaggio_cliente` TO 'segretarioTA';
GRANT EXECUTE ON procedure `agenzia_viaggi`.`lista_programmi_viaggi_segretari` TO 'segretarioTA';
```

```
GRANT EXECUTE ON procedure `agenzia_viaggi`.`assegna_guida` TO 'segretarioTA';
GRANT EXECUTE ON procedure `agenzia_viaggi`.`lista_viaggi_input_stato` TO 'segretarioTA';
GRANT EXECUTE ON procedure `agenzia_viaggi`.`assegna_albergo` TO 'segretarioTA';
GRANT EXECUTE ON procedure `agenzia_viaggi`.`assegna_autobus_privato` TO 'segretarioTA';
GRANT EXECUTE ON procedure `agenzia_viaggi`.`lista_assegnamenti_guide` TO 'segretarioTA';
GRANT EXECUTE ON procedure `agenzia_viaggi`.`lista_guide_all` TO 'segretarioTA';
GRANT EXECUTE ON procedure `agenzia_viaggi`.`lista_autobus_all` TO 'segretarioTA';
GRANT EXECUTE ON procedure `agenzia_viaggi`.`lista_assegnamenti_autobus` TO 'segretarioTA';
GRANT EXECUTE ON procedure `agenzia_viaggi`.`lista_alberghi_all` TO 'segretarioTA';
GRANT EXECUTE ON procedure `agenzia_viaggi`.`report_viaggio` TO 'segretarioTA';
GRANT EXECUTE ON procedure `agenzia_viaggi`.`get_itinerario_input_id` TO 'segretarioTA';
GRANT EXECUTE ON procedure `agenzia_viaggi`.`lista_localita_all` TO 'segretarioTA';
CREATE USER 'clienteTA' IDENTIFIED BY 'cliente';

GRANT EXECUTE ON procedure `agenzia_viaggi`.`lista_programmi_di_viaggio_cliente` TO 'clienteTA';
GRANT EXECUTE ON procedure `agenzia_viaggi`.`lista_viaggi_input_stato` TO 'clienteTA';
GRANT EXECUTE ON procedure `agenzia_viaggi`.`get_itinerario_input_id` TO 'clienteTA';
GRANT EXECUTE ON procedure `agenzia_viaggi`.`aggiungi_prenotazione` TO 'clienteTA';
GRANT EXECUTE ON procedure `agenzia_viaggi`.`aggiungi_partecipante` TO 'clienteTA';
GRANT EXECUTE ON procedure `agenzia_viaggi`.`disdici_prenotazione` TO 'clienteTA';
GRANT EXECUTE ON procedure `agenzia_viaggi`.`get_prenotazione_info` TO 'clienteTA';
GRANT EXECUTE ON procedure `agenzia_viaggi`.`lista_viaggi_prenotabili_input_id` TO 'clienteTA';
```

Strutture di memorizzazione

Tabella Partecipante		
Colonna	**Tipo di dato**	**Attributi**
Codice Fiscale	CHAR(16)	PK, NN
Nome	VARCHAR(45)	NN
Cognome	VARCHAR(45)	NN
E-mail	VARCHAR(45)	NN
Data Nascita	DATE	NN
Nazionalità	VARCHAR(45)	NN

Tabella Prenotazione		
Colonna	**Tipo di dato**	**Attributi**
ID Prenotazione	INT	PK, NN, AI
Data	DATE	NN
Codice Utente	VARCHAR(45)	
Data di partenza	DATE	NN

ID Programma	INT	NN

Tabella Impegno		
Colonna	**Tipo di dato**	**Attributi**
Codice Fiscale Partecipante	CHAR(16)	PK,NN
ID Prenotazione	INT	PK, NN

Tabella Programma Di Viaggio		
Colonna	**Tipo di dato**	**Attributi**
ID Programma	INT	PK, NN, AI
Numero Giorni	INT	NN
Nome Programma	VARCHAR(45)	NN

Tabella Viaggio		
Colonna	**Tipo di dato**	**Attributi**
Data di Partenza	DATA	PK, NN
ID Programma	INT	PK, NN
Data di Rientro	DATE	NN
Costo	DOUBLE	NN
Stato Viaggio	ENUM('in programma', 'attivo', 'terminato', 'prenotabile')	NN
Num. Partecipanti Confermati	INT	

Tabella Guida		
Colonna	**Tipo di dato**	**Attributi**
Codice Fiscale	CHAR(16)	PK, NN
Nome	VARCHAR(45)	NN
Cognome	VARCHAR(45)	NN
E-mail	VARCHAR(45)	NN
Data Nascita	DATE	NN

Tabella Accompagno		
Colonna	**Tipo di dato**	**Attributi**
Data di Partenza	DATE	PK, NN
ID Programma	INT	PK, NN
Codice Fiscale Guida	CHAR(16)	NN

Tabella Località		
Colonna	**Tipo di dato**	**Attributi**
Nome	VARCHAR(45)	PK, NN
Stato	VARCHAR(45)	NN

Tabella Tappa		
Colonna	**Tipo di dato**	**Attributi**
Giorno Arrivo	INT	PK, NN
Ora Arrivo	TIME	PK, NN
ID Programma	INT	PK, NN
Giorno Partenza	INT	NN
Ora Partenza	TIME	NN
Trattamento	ENUM('FB', 'HB', 'BB', 'OB', 'NB')	NN
Nome Località	VARCHAR(45)	NN

Tabella Albergo		
Colonna	**Tipo di dato**	**Attributi**
Indirizzo	VARCHAR(45)	PK, NN
Nome Località	VARCHAR(45)	PK, NN
Costo per notte per persona	DECIMAL(10,2)	NN
Referente	VARCHAR(45)	NN
Nome	VARCHAR(45)	NN

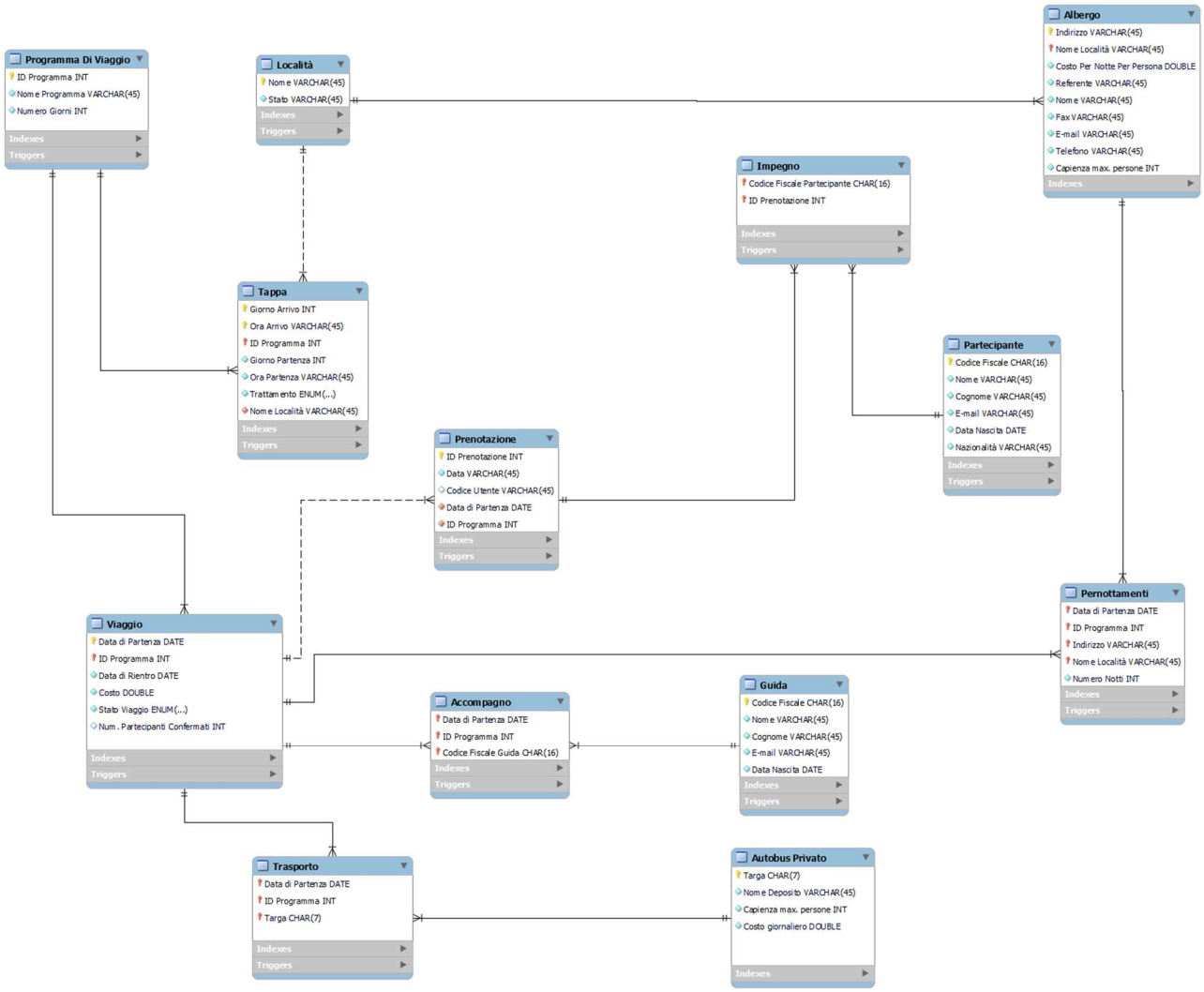
Fax	VARCHAR(45)	NN
Email	VARCHAR(45)	NN
Telefono	VARCHAR(45)	NN
Capienza max. persone	INT	NN

Tabella Pernottamenti		
Colonna	**Tipo di dato**	**Attributi**
Data di Partenza	DATE	PK, NN
ID Programma	INT	PK, NN
Indirizzo	VARCHAR(45)	PK, NN
Nome Località	VARCHAR(45)	PK, NN
Numero Notti	INT	NN

Tabella Autobus Privato		
Colonna	**Tipo di dato**	**Attributi**
Targa	CHAR(7)	PK, NN
Nome Deposito	VARCHAR(45)	NN
Capienza max. persone	INT	NN
Costo giornaliero	DOUBLE	NN

Tabella Trasporto		
Colonna	**Tipo di dato**	**Attributi**
Data di Partenza	DATA	PK, NN
ID Programma	INT	PK, NN
Targa	CHAR(8)	PK, NN

Viene riportato nella pagina successiva, al fine di facilitarne la comprensione, l'implementazione di queste tabelle realizzata con MySQL Workbench.



Indici

Per quanto riguarda le primary sono state create per identificare univocamente una tabella. Per quanto riguarda le fk sono state inserite per far 'dialogare' le varie tabelle

Tabella Partecipante	
Indice PRIMARY	**Tipo:**
Codice Fiscale	PR

Tabella Prenotazione	
Indice PRIMARY	**Tipo:**
ID Prenotazione	PR

Tabella Prenotazione	
Indice fk_Prenotazione_1_idx	**Tipo:**
Data di Partenza ID Programma	IDX

Tabella Impegno	
Indice PRIMARY	**Tipo:**
Codice Fiscale Partecipante ID Prenotazione	PR

Tabella Impegno	
Indice fk_Impegno_1_idx	**Tipo:**
Codice Fiscale Partecipante	IDX

Tabella Impegno	
Indice fk_Impegno_2_idx	**Tipo:**
ID Prenotazione	IDX

Tabella Programma Di Viaggio	
Indice PRIMARY	**Tipo:**
ID Programma	PR

Tabella Viaggio	
Indice PRIMARY	**Tipo:**
ID Programma Data di Partenza	PR

Tabella Viaggio	
Indice fk_Viaggio_1_idx	**Tipo:**

ID Programma	IDX

Tabella Guida	
Indice PRIMARY	**Tipo:**
Codice Fiscale	PR

Tabella Accompagno	
Indice PRIMARY	**Tipo:**
Data Di Partenza ID Programma Codice Fiscale Guida	PR

Tabella Accompagno	
Indice fk_Accompagno_1_idx	**Tipo:**
Codice Fiscale Guida	IDX

Tabella Accompagno	
Indice fk_Accompagno_2_idx	**Tipo:**
Data Di Partenza ID Programma	IDX

Tabella Località	
Indice PRIMARY	**Tipo:**
Nome	PR

Tabella Tappa	
Indice PRIMARY	**Tipo:**
Giorno Arrivo Ora Arrivo	PR

ID Programma	

Tabella Tappa	
Indice fk_Tappa_1_idx	**Tipo:**
ID Programma	IDX

Tabella Tappa	
Indice fk_Tappa_2_idx	**Tipo:**
Nome Località	IDX

Tabella Albergo	
Indice PRIMARY	**Tipo:**
Indirizzo Nome Località	PR

Tabella Albergo	
Indice fk_Albergo_1_idx	**Tipo:**
Nome Località	IDX

Tabella Pernottamenti	
Indice PRIMARY	**Tipo:**
Data Di Partenza ID Programma Indirizzo Nome Località	PR

Tabella Pernottamenti	
Indice fk_Pernottamenti_1_idx	**Tipo:**
Data Di Partenza	IDX

ID Programma	

Tabella Pernottamenti	
Indice fk_Pernottamenti_2_idx	**Tipo:**
Indirizzo Nome Località	IDX

Tabella Autobus Privato	
Indice PRIMARY	**Tipo:**
Targa	PR

Tabella Trasporto	
Indice PRIMARY	**Tipo:**
Data Di Partenza ID Programma Targa	PR

Tabella Trasporto	
Indice fk_Trasporto_1_idx	**Tipo:**
Targa	IDX

Tabella Trasporto	
Indice fk_Trasporto_2_idx	**Tipo:**
Data Di Partenza ID Programma	IDX

Trigger

Si osservi che in questa fase, seppur necessari, non è stato specificato alcun livello di isolamento. Questi verranno invece specificati nelle stored procedure ed i trigger erediteranno il livello di isolamento dalla procedura che li invoca.

Programma Di Viaggio_BEFORE_INSERT

Verifica che il numero di giorni inseriti per un programma di viaggio sia un numero valido

```
CREATE DEFINER = CURRENT_USER TRIGGER `agenzia_viaggi`.`Programma Di Viaggio_BEFORE_INSERT` BEFORE
INSERT ON `Programma Di Viaggio` FOR EACH ROW
BEGIN
if (NEW.`Numero Giorni`<1 or NEW.`Numero Giorni`>7)  THEN
signal sqlstate '45001' set message_text="Non è stato inserito un numero di giorni valido";
END if;
END
```

Viaggio_BEFORE_INSERT

Verifica che i dati inseriti per un viaggio siano coerenti a quelli del programma di viaggio a lui associato; inoltre controlla anche che la data inserita sia tale da permettere agli utenti di prenotarsi.

```
CREATE DEFINER = CURRENT_USER TRIGGER `agenzia_viaggi`.`Viaggio_BEFORE_INSERT` BEFORE INSERT ON
`Viaggio` FOR EACH ROW
BEGIN
declare num_giorni int;
declare num_giorni_input int;
declare cur_date date;
declare supp int;

select curdate() into cur_date;

select if(NEW.`Data di Partenza`> DATE(DATE_ADD(cur_date, INTERVAL 20 DAY)), 1, 0) into supp;

if(supp=0)then
signal sqlstate '45001' set message_text="Per favore, inserisci una data di partenza valida, deve
essere almeno successiva di 20 giorni alla data odierna ";
END if;

select `Numero Giorni` from `Programma Di Viaggio`
where `ID Programma`=NEW.`ID Programma` into num_giorni;

select DATEDIFF(NEW.`Data di Rientro`,NEW.`Data di Partenza`) into num_giorni_input;

if(num_giorni_input<>num_giorni)THEN
signal sqlstate ‘45001’ set message_text=”Date di partenza e rientro non compatibili con il numero di
giorni del viaggio”;
END IF;
```

```
END
```

Località_BEFORE_INSERT

Verifica che la località inserita appartenga ad uno stato europeo, in quanto l'agenzia prevede viaggi solo in Italia e in Europa.

```
CREATE DEFINER = CURRENT_USER TRIGGER `agenzia_viaggi`.`Località_BEFORE_INSERT` BEFORE INSERT ON
`Località` FOR EACH ROW
BEGIN
declare supp  int;
select if( NEW.`Stato` in
    ('Italia','Albania','Andorra','Armenia','Austria','Azerbaigian','Bielorussia','Belgio','Bosnia-
Erzegovina','Bulgaria','Cipro','Croazia','Danimarca','Estonia','Ex-Repubblica jugoslava di
Macedonia','Finlandia','Francia','Georgia','Germania','Grecia','Irlanda','Islanda','Lettonia','Liechtens
tein','Lituania','Lussemburgo','Malta','Moldavia','Monaco','Montenegro','Norvegia','Paesi
Bassi','Polonia','Portogallo','Regno Unito','Repubblica ceca','Romania','Federazione di Russia','San
Marino','Serbia','Slovacchia','Slovenia','Spagna','Svezia','Svizzera','Turchia','Ucraina','Ungheria' )
          , 1, 0) into supp;
if (supp=0) THEN
signal sqlstate '45001' set message_text="Inserire per favore una località italiana o europea";
END if;
END
```

Tappa_BEFORE_INSERT

Verifica che i giorni di arrivo e di partenza di una tappa siano compatibili tra di loro, inoltre verifica anche se è stato specificato un servizio corretto in base al fatto che la tappa preveda un pernottamento o meno. Infine verifica che il giorno di arrivo non sia successivo ai giorni previsti dal programma di viaggio

```
CREATE DEFINER = CURRENT_USER TRIGGER `agenzia_viaggi`.`Tappa_BEFORE_INSERT` BEFORE INSERT ON `Tappa`
FOR EACH ROW
BEGIN
declare num_giorni int;

if(NEW.`Giorno Partenza`< NEW.`Giorno Arrivo`) THEN
signal sqlstate '45001' set message_text="Giorno di Arrivo e Giorno di Partenza inseriti non sono
compatibili";
END if;

if(NEW.`Giorno Arrivo`< NEW.`Giorno Partenza` AND NEW.`Trattamento`='NB') THEN
signal sqlstate '45001' set message_text="La tappa prevede un pernottamento ma non è stato specificato
un trattamento valido.";
END if;

if(NEW.`Giorno Arrivo`= NEW.`Giorno Partenza` AND NEW.`Trattamento`<>'NB') THEN
```

```
signal sqlstate '45001' set message_text="La tappa non prevede un pernottamento, non è quindi corretto
specificare un trattamento.";
END if;

select `Numero Giorni` from `Programma Di Viaggio`
where `ID Programma`=NEW.`ID Programma`
into num_giorni;

if(NEW.`Giorno Arrivo`>num_giorni) THEN
signal sqlstate '45001' set message_text="Giorno di arrivo non valido, supera il numero  di giorni
previsti dal piano di viaggio";
END if;

END
```

Partecipante_BEFORE_INSERT

Verifica che il formato del codice fiscale e della email inserita per un partecipante siano corretti.

```
CREATE DEFINER = CURRENT_USER TRIGGER `agenzia_viaggi`.`Partecipante_BEFORE_INSERT` BEFORE INSERT ON
`Partecipante` FOR EACH ROW
BEGIN
if (NEW.`Codice Fiscale` not regexp'^[A-Z]{6}[0-9]{2}[A-Z][0-9]{2}[A-Z][0-9]{3}[A-Z]$') then
signal sqlstate '45001' set message_text = 'Formato codice fiscale errato';
end if;
if (NEW.`E-mail` not regexp'^[a-zA-Z0-9][a-zA-Z0-9._-]*@[a-zA-Z0-9][a-zA-Z0-9._-]*\\.[a-zA-Z]{2,4}$')
then
signal sqlstate '45001' set message_text = 'Formato email  errato';
end if;
END
```

Prenotazione_BEFORE_INSERT

Verifica che la prenotazione faccia riferimento ad un viaggio che è prenotabile.

```
CREATE DEFINER = CURRENT_USER TRIGGER `agenzia_viaggi`.`Prenotazione_BEFORE_INSERT` BEFORE INSERT ON
`Prenotazione` FOR EACH ROW
BEGIN

declare stato_viaggio ENUM('in programma', 'attivo', 'terminato', 'prenotabile');

select `Stato Viaggio` from Viaggio
where `ID Programma`=NEW.`ID Programma` and `Data di Partenza`= NEW.`Data di Partenza`
into stato_viaggio;

if(stato_viaggio<>'prenotabile')then
signal sqlstate '45001' set message_text="Non è più possibile prenotarsi a questo viaggio.";
end if;
```

```
END
```

Impegno_BEFORE_INSERT

Verifica che la prenotazione sia ancora disdicibile.

```
CREATE DEFINER = CURRENT_USER TRIGGER `agenzia_viaggi`.`Impegno_BEFORE_INSERT` BEFORE INSERT ON
`Impegno` FOR EACH ROW
BEGIN
declare supp int;
select if(NEW.`ID Prenotazione`in(
            select `ID Prenotazione`
            from Prenotazione
            where `Codice Utente` is not null ), 1, 0) into supp;
if(supp=0)then
signal sqlstate '45001' set message_text="Mi dispiace, le prenotazioni per questo viaggio sono
terminate.";
end if;
END
```

Guida_BEFORE_INSERT

Verifica che il formato del codice fiscale e della email inserita per una guida siano corretti.

```
BEGIN
if (NEW.`Codice Fiscale` not regexp'^[A-Z]{6}[0-9]{2}[A-Z][0-9]{2}[A-Z][0-9]{3}[A-Z]$') then
signal sqlstate '45001' set message_text = 'Formato codice fiscale errato';
end if;
if (NEW.`E-mail` not regexp'^[a-zA-Z0-9][a-zA-Z0-9._-]*@[a-zA-Z0-9][a-zA-Z0-9._-]*\\.[a-zA-Z]{2,4}$')
then
signal sqlstate '45001' set message_text = 'Formato email  errato';
end if;
END
```

Accompagno_BEFORE_INSERT

Verifica che lo stato del viaggio permetta l'assegnamento, inoltre verifica se il viaggio a cui si vuole assegnare la guida abbia già una guida assegnata. Infine, verifica che le date del viaggio a cui la guida si vuole assegnare non si sovrappongano ad altre date di viaggi a cui la guida è stata precedentemente assegnata.

```
CREATE DEFINER = CURRENT_USER TRIGGER `agenzia_viaggi`.`Accompagno_BEFORE_INSERT` BEFORE INSERT ON
`Accompagno` FOR EACH ROW
BEGIN
DECLARE done INT DEFAULT FALSE;
DECLARE data_rientro_in DATE;
```

```
DECLARE viaggio_id INT;
DECLARE viaggio_date DATE;
DECLARE data_rientro DATE;
declare assegnamento_guida int;
declare stato_viaggio ENUM('in programma', 'attivo', 'terminato', 'prenotabile');

DECLARE cur CURSOR FOR select `ID Programma`,`Data di Partenza` FROM `Accompagno`
where `Codice Fiscale Guida` = NEW.`Codice Fiscale Guida`;

DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

select `Data di Rientro`
from `Viaggio`
where `Data di Partenza`=NEW.`Data di Partenza` and `ID Programma`=NEW.`ID Programma`
into data_rientro_in;

select `Stato Viaggio`
from Viaggio
where `ID Programma`=NEW.`ID Programma` and `Data di Partenza`= NEW.`Data di Partenza`
into stato_viaggio;

if(stato_viaggio<>'in programma')then
signal sqlstate '45001' set message_text="Lo stato attuale del viaggio non permette l'assegnamento";
end if;

select count(*) from `Accompagno`
where `ID Programma`=NEW.`ID Programma` and `Data di Partenza`= NEW.`Data di Partenza`
into assegnamento_guida;

if(assegnamento_guida<>0)then
signal sqlstate '45001' set message_text="Il viaggio selezionato ha già  una guida assegnata";
end if;

OPEN cur;
 read_loop: loop
 FETCH cur INTO viaggio_id,viaggio_date;
 if done THEN
 LEAVE read_loop;
 END IF;
 select `Data di Rientro` FROM `Viaggio` WHERE `Data di Partenza`=viaggio_date AND `ID Programma`=
viaggio_id  into data_rientro;

  if((NEW.`Data di Partenza` BETWEEN viaggio_date AND data_rientro) OR (data_rientro_in BETWEEN
viaggio_date AND data_rientro))THEN
 signal sqlstate '45001' set message_text="La guida è stata già assegnata in questo periodo";
  END if;
  END LOOP;
CLOSE cur;
END
```

Pernottamenti_BEFORE_INSERT

Verifica che lo stato del viaggio permetta l'assegnamento, inoltre verifica se la tappa del viaggio a cui si vuole assegnare l'albergo ne ha già uno assegnato. Inoltre, verifica che il numero di notti inserite sia coerenti con il numero di notti previste dalla tappa. Infine verifica che la capienza dell'albergo sia tale da permettere il pernottamento di tutti i partecipanti al viaggio.

```
CREATE DEFINER = CURRENT_USER TRIGGER `agenzia_viaggi`.`Pernottamenti_BEFORE_INSERT` BEFORE INSERT ON
`Pernottamenti` FOR EACH ROW
BEGIN
declare stato_viaggio ENUM('in programma', 'attivo', 'terminato', 'prenotabile');
declare count_pernottamenti int;
declare num_notti int;
declare assegnamento_albergo int;
declare capienza_max int;
declare num_partecipanti int;

select `Stato Viaggio`from Viaggio
where `ID Programma`=NEW.`ID Programma` and `Data di Partenza`= NEW.`Data di Partenza`
into stato_viaggio;

if(stato_viaggio<>'in programma')then
signal sqlstate '45001' set message_text="Lo stato attuale del viaggio non permette l'assegnamento";
end if;

select count(*) from Tappa
where `ID Programma`=NEW.`ID Programma` AND  `Nome Località`=NEW.`Nome Località` AND `Trattamento`<>'NB'
into count_pernottamenti;

if not(count_pernottamenti>0) then
signal sqlstate '45001' set message_text="Non esiste una tappa con pernottamento in questa località per
questo viaggio";
END if;

select `Tappa`.`Giorno Partenza`-`Tappa`.`Giorno Arrivo`from Tappa
where `ID Programma`=NEW.`ID Programma` AND  `Nome Località`=NEW.`Nome Località` AND `Trattamento`<>'NB'
into num_notti;

if (num_notti<>NEW.`Numero Notti`) then
signal sqlstate '45001' set message_text="Non è stato inserito un numero di notti valido per questa
tappa";
END if;

select count(*) from `Pernottamenti`
where `ID Programma`=NEW.`ID Programma` and `Data di Partenza`= NEW.`Data di Partenza` and `Nome
Località`=NEW.`Nome Località`
into assegnamento_albergo;

if (assegnamento_albergo<>0) then
signal sqlstate '45001' set message_text="Un albergo è già  stato assegnato per questa tappa";
END IF;
```

```
select  `Capienza max. persone` from Albergo
where `Nome Località`=NEW.`Nome Località` and `Indirizzo`=NEW.`Indirizzo`
into capienza_max;

select  `Num. Partecipanti Confermati`from Viaggio
where `ID Programma`=NEW.`ID Programma` and `Data di Partenza`= NEW.`Data di Partenza`
into num_partecipanti;

if(num_partecipanti>capienza_max) then
signal sqlstate '45001' set message_text="L'albergo scelto non dispone di una capienza sufficente ad
ospitare tutti i partecipanti al viaggio";
END if;

END
```

Trasporto_BEFORE_INSERT

Verifica che lo stato del viaggio permetta l'assegnamento, inoltre verifica se il viaggio a cui si vuole assegnare l'autobus privato non ha già assegnati un numero sufficiente di autobus privati tali da permettere il trasporto di tutti i partecipanti. Infine, verifica che le date del viaggio a cui l'autobus privato si vuole assegnare non si sovrappongano ad altre date di viaggi a cui l'auotbus privato era stato precedentemente assegnato.

```
CREATE DEFINER = CURRENT_USER TRIGGER `agenzia_viaggi`.`Trasporto_BEFORE_INSERT` BEFORE INSERT ON
`Trasporto` FOR EACH ROW
BEGIN
DECLARE done INT DEFAULT FALSE;
DECLARE data_rientro_in DATE;
DECLARE viaggio_id INT;
DECLARE viaggio_date DATE;
DECLARE data_rientro DATE;
DECLARE capienza_attuale int;
DECLARE num_partecipanti int;
DECLARE stato_viaggio ENUM('in programma', 'attivo', 'terminato', 'prenotabile');

DECLARE cur CURSOR FOR select `ID Programma`,`Data di Partenza` FROM `Trasporto`
where `Targa` = NEW.`Targa`;

DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

select `Stato Viaggio` from Viaggio
where `ID Programma`=NEW.`ID Programma` AND `Data di Partenza`= NEW.`Data di Partenza`
into stato_viaggio;

if(stato_viaggio<>'in programma')THEN
signal sqlstate '45001' set message_text="Lo stato attuale del viaggio non permette l'assegnamento";
END if;

select `Data di Rientro` FROM `Viaggio`
```

```
where `Data di Partenza`=NEW.`Data di Partenza` AND `ID Programma`=NEW.`ID Programma`
into data_rientro_in;

OPEN cur;
 read_loop: loop
 FETCH cur INTO viaggio_id,viaggio_date;
 if done THEN
 LEAVE read_loop;
 END if;
 select `Data di Rientro` from `Viaggio` where `Data di Partenza`=viaggio_date and `ID Programma`=
viaggio_id into data_rientro;

 if(
  ( NEW.`Data di Partenza` BETWEEN viaggio_date AND data_rientro)
      or (data_rientro_in BETWEEN viaggio_date AND data_rientro)
  )
  THEN
  signal sqlstate '45001' set message_text="L'autobus privato è stato già assegnato in questo periodo";
  END if;
  END LOOP;
CLOSE cur;

select  `Num. Partecipanti Confermati` from Viaggio
where `ID Programma`=NEW.`ID Programma` and `Data di Partenza`= NEW.`Data di Partenza`
into num_partecipanti;

select sum(`Capienza max. persone`) from Trasporto
join `Autobus Privato` on Trasporto.Targa=`Autobus Privato`.Targa
where `ID Programma`=NEW.`ID Programma` and `Data di Partenza`= NEW.`Data di Partenza`
into capienza_attuale;

if(capienza_attuale >num_partecipanti)THEN
signal sqlstate '45001' set message_text="Non sono necessari ulteriori autobus privati per il viaggio
selezionato";
END if;

END
```

Eventi

Questo evento è schedulato per ogni giorno a mezzanotte. Gioca un ruolo di fondamentale importanza nella nostra applicazione, infatti si occupa di verificare che i viaggi siano in uno stato corretto, che ricordiamo essere “prenotabile”, “in programma”, “attivo” e “terminato”. Qualora un viaggio necessiti di cambiare il proprio stato, l’evento si occupa di svolgere tutte le attività necessarie. Si occupa anche di fare della pulizia nella base di dati: se è passato più di un anno dalla data di rientro di un viaggio, questo e tutti i dati relativi ad esso verranno eliminati.

```
set global event_scheduler = on;
delimiter |
CREATE EVENT IF NOT EXISTS aggiorna_stato
```

```
  ON SCHEDULE EVERY 1 DAY STARTS (TIMESTAMP(CURRENT_DATE) + INTERVAL 1 DAY )
  ON COMPLETION PRESERVE
  DO
   BEGIN
    DECLARE data_odierna DATE;
    DECLARE done INT DEFAULT FALSE;
    DECLARE viaggio_id INT;
    DECLARE viaggio_partenza DATE;
    DECLARE viaggio_rientro DATE;
    DECLARE num_partecipanti int;
    declare stato_viaggio ENUM('in programma', 'attivo', 'terminato', 'prenotabile');
    DECLARE cur CURSOR FOR
    select `ID Programma`,`Data di Partenza`,`Data di Rientro`,`Stato Viaggio` from `Viaggio`;
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

    declare exit handler for sqlexception
    begin
        rollback;
        resignal;
    end;

    select curdate() into data_odierna;

    set transaction isolation level serializable;
    start transaction;

OPEN cur;
        read_loop: loop
                    FETCH cur INTO viaggio_id,viaggio_partenza,viaggio_rientro,stato_viaggio;
        IF done THEN
                    LEAVE read_loop;
        END IF;

        if((data_odierna BETWEEN viaggio_partenza AND viaggio_rientro ) AND stato_viaggio <>'attivo')
THEN
        update Viaggio set `Stato Viaggio`='attivo'
        where `ID Programma`=viaggio_id AND `Data di Partenza`=viaggio_partenza;
        END if;

        if(data_odierna > viaggio_rientro AND stato_viaggio <>'terminato')THEN
        update Viaggio set `Stato Viaggio`='terminato'
        where `ID Programma`=viaggio_id AND `Data di Partenza`=viaggio_partenza ;
        END if;

        if((DATE_ADD(data_odierna, INTERVAL 20 DAY) >= viaggio_partenza)
           AND data_odierna < viaggio_partenza AND stato_viaggio<>'in programma')
           THEN
          select count(`Codice Fiscale Partecipante`) from  Impegno
          join Prenotazione on Impegno.`ID Prenotazione`=Prenotazione.`ID Prenotazione`
          join Viaggio on  Viaggio.`ID Programma`=Prenotazione.`ID Programma`
          AND Viaggio.`Data di Partenza`=Prenotazione.`Data di Partenza`
          where Viaggio.`ID Programma`=viaggio_id and Viaggio.`Data di Partenza`=viaggio_partenza
          into num_partecipanti;
```

```
        update Viaggio
        set `Stato Viaggio`='in programma',`Num. Partecipanti Confermati` = num_partecipanti
        where `ID Programma`=viaggio_id AND `Data di Partenza`=viaggio_partenza;

        update Prenotazione set `Codice Utente`= NULL
        where `ID Programma`=viaggio_id AND `Data di Partenza`=viaggio_partenza;

       END IF;

         IF(DATE_ADD( viaggio_rientro, INTERVAL 1 YEAR) <= data_odierna)THEN
         DELETE from Trasporto
         where `ID Programma`=viaggio_id AND `Data di Partenza`=viaggio_partenza ;
         DELETE from Accompagno
         where `ID Programma`=viaggio_id AND `Data di Partenza`=viaggio_partenza ;
         DELETE from Pernottamenti
         where `ID Programma`=viaggio_id AND `Data di Partenza`=viaggio_partenza ;
         DELETE from Impegno
         where `ID Prenotazione` in (
                select `ID Prenotazione`
                where `ID Programma`=viaggio_id AND `Data di  Partenza`=viaggio_partenza);
         DELETE from Prenotazione
         where `ID Programma`=viaggio_id AND `Data di Partenza`=viaggio_partenza;
         DELETE from Viaggio
         where `ID Programma`=viaggio_id AND `Data di Partenza`=viaggio_partenza;
         END if;

         END LOOP;

    commit;

CLOSE cur;
   END |
delimiter ;
```

Viste

Non previste

Stored Procedures e transazioni

login_segretario

Permette l'accesso al sistema verificando l'identità dei segretari

```
CREATE PROCEDURE `login_segretario` (in var_username varchar(45), in var_pass varchar(45), out var_role
INT)
BEGIN
SELECT IF( EXISTS(
             SELECT *
             FROM Segretari
             WHERE `username` = var_username and `password` = md5(var_pass)), 1, 0) into var_role;
```

```
        #return 1 if the user exist, else 0
END
```

aggiungi_localita

Usata dai segretari per inserire una località nel sistema

```
CREATE PROCEDURE `aggiungi_localita` (IN var_nome varchar(45),IN var_stato varchar(45))
BEGIN
INSERT INTO `agenzia_viaggi`.`Località` (`Nome`, `Stato`)
VALUES (var_nome, var_stato);
END
```

aggiungi_guida

Usata dai segretari per inserire una guida nel sistema

```
CREATE PROCEDURE `aggiungi_guida` (IN var_cf char(16),IN var_nome varchar(45),IN var_cognome char(45),IN
var_email varchar(45),IN var_data date)
BEGIN
INSERT INTO `agenzia_viaggi`.`Guida` (`Codice Fiscale`, `Nome`, `Cognome`, `E-mail`, `Data Nascita`)
VALUES (var_cf, var_nome, var_cognome, var_email, var_data);
END
```

aggiungi_albergo

Usata dai segretari per inserire un albergo nel sistema

```
CREATE PROCEDURE `aggiungi_albergo` (IN var_ind varchar(45),IN var_loc varchar(45),IN var_cpn DOUBLE,IN
var_ref varchar(45),IN var_nome varchar(45),IN var_fax varchar(45),IN var_email varchar(45),IN var_tel
varchar(45),IN var_capmax int)
BEGIN
insert into `agenzia_viaggi`.`Albergo` (`Indirizzo`, `Nome Località`, `Costo Per Notte Per Persona`,
`Referente`, `Nome`, `Fax`, `E-mail`, `Telefono`, `Capienza max. persone`)
values (var_ind, var_loc, var_cpn, var_ref, var_nome, var_fax, var_email, var_tel, var_capmax);
END
```

aggiungi_tappa

Usata dai segretari per aggiungere una tappa ad un programma di viaggio

```
CREATE PROCEDURE `aggiungi_tappa` (in var_giorno_arrivo int, in var_ora_arrivo time, in var_id_programma
int, in var_giorno_partenza int, in var_ora_partenza time, in var_trattamento enum('FB', 'HB', 'BB',
'OB', 'NB'), in var_nome_località varchar(45))
BEGIN
```

```
declare exit handler for sqlexception
    begin
        rollback;
        resignal;
    end;

    set transaction isolation level read committed;
    start transaction;
INSERT INTO `agenzia_viaggi`.`Tappa` (`Giorno Arrivo`, `Ora Arrivo`, `ID Programma`, `Giorno Partenza`,
`Ora Partenza`, `Trattamento`, `Nome Località`)
VALUES (var_giorno_arrivo, var_ora_arrivo, var_id_programma, var_giorno_partenza, var_ora_partenza,
var_trattamento, var_nome_località);
    commit;
END
```

aggiungi_programma_di_viaggio

Usata dai segretari per inserire un programma di viaggio nel sistema, restituisce in output l'id del programma di viaggio creato.

```
CREATE PROCEDURE `aggiungi_programma_di_viaggio` (in var_nome varchar(45),in var_numero_giorni int, out
id_programma int)
BEGIN
INSERT INTO `agenzia_viaggi`.`Programma Di Viaggio` (`Nome Programma`, `Numero Giorni`)
VALUES (var_nome,var_numero_giorni);
set id_programma=last_insert_id();
END
```

aggiungi_autobus_privato

Usata dai segretari per inserire un autobus privato nel sistema

```
CREATE PROCEDURE `aggiungi_autobus_privato` (in var_targa char(7), in var_deposito char(45) , in
var_capmax int,in  var_costo_g double)
BEGIN
INSERT INTO `agenzia_viaggi`.`Autobus Privato` (`Targa`, `Nome Deposito`, `Capienza max. persone`,
`Costo giornaliero`)
VALUES (var_targa, var_deposito,var_capmax, var_costo_g);
END
```

aggiungi_viaggio

Usata dai segretari per inserire un viaggio nel sistema.

```
CREATE PROCEDURE `aggiungi_viaggio`(in var_data_p date, in var_id_programma int, in var_data_r date, in
var_costo double)
BEGIN
```

```
declare exit handler for sqlexception
    begin
        rollback;
        resignal;
    end;

    set transaction isolation level read committed;
    start transaction;
INSERT INTO `agenzia_viaggi`.`Viaggio` (`Data di Partenza`, `ID Programma`, `Data di Rientro`, `Costo`)
VALUES (var_data_p, var_id_programma, var_data_r, var_costo);
   commit;
END
```

lista_programmi_di_viaggio_cliente

Usata dai cienti per visualizzare i programmi di viaggi disponibili

```
CREATE PROCEDURE `lista_programmi_di_viaggio_cliente` ()
BEGIN
declare exit handler for sqlexception
    begin
        rollback;
        resignal;
    end;
 -- anche se il costo minimo potrebbe non essere corretto si è deciso di non appesantire questa procedura con un livello di isolamento più stretto anche tenendo in considerazione la sua frequenza elevata
    set transaction isolation level read committed;
    set transaction read only;
    start transaction;
    select `Nome Programma`,`Numero Giorni`,p.`ID Programma`, `Costo`
    from `Programma Di Viaggio` as p
    join `Viaggio` on p.`ID Programma`=`Viaggio`.`ID Programma`
    WHERE `Stato Viaggio`='prenotabile' AND
    Costo <= all( select Costo
                  from Viaggio
                  where `ID Programma`=p.`ID Programma`);
    commit;
END
```

lista_viaggi_input_stato

Usata dai segretari per mostrare la lista dei viaggi che si trovano in un certo stato.

```
CREATE PROCEDURE `lista_viaggi_input_stato` (in var_stato ENUM('in programma', 'attivo', 'terminato', 'prenotabile'))
BEGIN
declare exit handler for sqlexception
    begin
        rollback;
```

```
        resignal;
    end;

    set transaction isolation level read committed;
    set transaction read only;
    start transaction;
    SELECT `Data di Partenza`,`Data di Rientro`,`Costo`,`Num. Partecipanti Confermati`,`ID Programma`
    FROM `Viaggio`
  WHERE `Stato Viaggio`=var_stato;
 commit;
END
```

get_itinerario_input_id

Usata sia dai clienti che dai segretari per conoscere l'itinerario di un programma di viaggio, noto il suo id.

```
CREATE PROCEDURE `get_itinerario_input_id` (in var_id_programma int)
BEGIN
declare exit handler for sqlexception
    begin
        rollback;
        resignal;
    end;

    set transaction isolation level read committed;
    start transaction;
 SELECT `Nome Località`,`Giorno Arrivo`,`Ora Arrivo`,`Giorno Partenza`,`Ora Partenza`,`Trattamento`
 FROM `Tappa`
 WHERE `ID Programma`=var_id_programma
 ORDER BY `Giorno Arrivo`,`Ora Arrivo`;
    commit;
END
```

lista_programmi_viaggi_segretari

Usata dai segretari per visualizzare l'elenco completo dei programmi di viaggio.

```
CREATE PROCEDURE `lista_programmi_viaggi_segretari` ()
BEGIN
declare exit handler for sqlexception
    begin
        rollback;
        resignal;
    end;

    set transaction isolation level read committed;
    set transaction read only;
    start transaction;
```

```
        select `Nome Programma`,`Numero Giorni`,`ID Programma` from `Programma Di Viaggio`;
    commit;
END
```

assegna_guida

Usata dai segretari per assegnare una guida ad un viaggio.

```
CREATE PROCEDURE `assegna_guida` (in var_cf CHAR(16),in id_programma_in int,in data_partenza_in date)
BEGIN
declare exit handler for sqlexception
    begin
        rollback;
        resignal;
    end;
set transaction isolation level serializable;
start transaction;
 INSERT INTO `agenzia_viaggi`.`Accompagno` (`Data di Partenza`, `ID Programma`, `Codice Fiscale Guida`)
  VALUES (data_partenza_in, id_programma_in, var_cf);
commit;
END
```

assegna_autobus_privato

Usata dai segretari per assegnare un auotbus privato ad un viaggio.

```
CREATE PROCEDURE `assegna_autobus_privato` (in id_programma_in int,in data_partenza_in date,in targa
char(7))
BEGIN
declare exit handler for sqlexception
    begin
        rollback;
        resignal;
    end;

    set transaction isolation level serializable;
    start transaction;
 INSERT INTO `agenzia_viaggi`.`Trasporto` (`Data di Partenza`, `ID Programma`, `Targa`)
 VALUES (data_partenza_in, id_programma_in, targa);
 commit;
END
```

assegna_albergo

Usata dai segretari per assegnare un albergo ad un viaggio.

```
CREATE PROCEDURE `assegna_albergo` (in id_programma_in int,in data_partenza_in date,in indirizzo
varchar(45),in nome_localita varchar(45),in numero_notti int)
BEGIN
declare exit handler for sqlexception
    begin
        rollback;
        resignal;
    end;

    set transaction isolation level serializable;
    start transaction;
INSERT INTO `agenzia_viaggi`.`Pernottamenti` (`Data di Partenza`, `ID Programma`, `Indirizzo`, `Nome
Località`, `Numero Notti`)
VALUES (data_partenza_in, id_programma_in, indirizzo, nome_localita, numero_notti);
    commit;
END
```

lista_localita_all

Usata dai segretari per visualizzare l'elenco completo delle località presenti nel sistema.

```
BEGIN
declare exit handler for sqlexception
    begin
        rollback;
        resignal;
    end;

    set transaction isolation level read committed;
    set transaction read only;
    start transaction;
          select * from `Località`
          order by `Stato`;
    commit;
END
```

lista_assegnamenti_guide

Usata dai segretari per mostrare tutti gli assegnamenti delle guide, con date di partenza dei viaggi successive alla data odierna.

```
CREATE PROCEDURE `lista_assegnamenti_guide` ()
BEGIN
declare cur_date date;
declare exit handler for sqlexception
    begin
        rollback;
        resignal;
    end;
```

```
select curdate() into cur_date;
set transaction isolation level read committed;
set transaction read only;
 start transaction;
    select `Codice Fiscale`,`Viaggio`.`Data di Partenza`,`Data di Rientro`
    from Guida
    join Accompagno on `Codice Fiscale`=`Codice Fiscale Guida`
    join Viaggio on (Accompagno.`Data di Partenza`=Viaggio.`Data di Partenza` and Accompagno.`ID Programma`=Viaggio.`ID Programma`)
    where `Viaggio`.`Data di Partenza`>= cur_date;
  commit;
END
```

lista_guide_all

Usata dai segretari per leggere i dati di tutte le guide.

```
CREATE PROCEDURE `lista_guide_all`()
BEGIN
declare exit handler for sqlexception
    begin
        rollback;
        resignal;
    end;

    set transaction isolation level read committed;
    set transaction read only;
    start transaction;
    select * from Guida ;
    commit;
END
```

lista_assegnamenti_autobus

Usata dai segretari per mostrare tutti gli assegnamenti degli autobus privati, con date di partenza dei viaggi successive alla data odierna.

```
CREATE PROCEDURE `lista_assegnamenti_autobus` ()
BEGIN
declare cur_date date;
declare exit handler for sqlexception
    begin
        rollback;
        resignal;
    end;
    set transaction isolation level read committed;
    set transaction read only;
    start transaction;
    select Trasporto.`Targa`,`Viaggio`.`Data di Partenza`,`Data di Rientro`
```

```
    from `Autobus Privato`
    join Trasporto on Trasporto.`Targa`=`Autobus Privato`.`Targa`
    join Viaggio on (Trasporto.`Data di Partenza`=Viaggio.`Data di Partenza` and Trasporto.`ID Programma`=Viaggio.`ID Programma`)
    where `Viaggio`.`Data di Partenza`>= cur_date;
    commit;
END
```

lista_autobus_all

Usata dai segretari per leggere i dati di tutti gli autobus privati.

```
CREATE PROCEDURE `lista_autobus_all`()
BEGIN
declare exit handler for sqlexception
    begin
        rollback;
        resignal;
    end;

    set transaction isolation level read committed;
    set transaction read only;
    start transaction;
    select *
    from `agenzia_viaggi`.`Autobus Privato` ;
    commit;
END
```

lista_alberghi_all

Usata dai segretari per leggere i dati di tutti gli alberghi.

```
CREATE PROCEDURE `lista_alberghi_all`()
BEGIN
declare exit handler for sqlexception
    begin
        rollback;
        resignal;
    end;

    set transaction isolation level read committed;
    set transaction read only;
    start transaction;
    select * from Albergo ;
    commit;
END
```

aggiungi_prenotazione

Usata dai clienti per prenotarsi ad un viaggio, resituisce l'id della prenotazione, il codice per gestirla ed infine il prezzo totale per il numero di partecipanti inserito.

```
CREATE PROCEDURE `aggiungi_prenotazione` (in data_p date, in id_programma int,in num_persone int,out
codice varchar(45),out newid int,out costo_tot_prenotazione double)
BEGIN

    declare costo_viaggio double;
    declare dataprenotazione date;
    declare exit handler for sqlexception
    begin
        rollback;
        resignal;
    end;

    select left(UUID(), 8) into codice; -- 8 character pseudo-random string
    select curdate() into dataprenotazione;


    set transaction isolation level read committed;
    start transaction;

   insert into `agenzia_viaggi`.`Prenotazione`(`Data di Partenza`,`ID Programma`,`Codice Utente`,`Data`)
    values (data_p,id_programma,md5(codice),dataprenotazione);

    select Costo
    from Viaggio
    where `ID Programma`=id_programma AND `Data di Partenza`=data_p into  costo_viaggio;

    commit;
    set newid=last_insert_id();
    set costo_tot_prenotazione=costo_viaggio*num_persone;
END
```

aggiungi_partecipante

Usata dai clienti per aggiungere i dati di un partecipante alla prenotazione

```
CREATE PROCEDURE `aggiungi_partecipante` (in id_prenotazione int, IN var_cf char(16),IN var_nome
varchar(45),IN var_cognome char(45),IN var_email varchar(45),IN var_data_n date,IN nazionalita
varchar(45))
BEGIN
declare exit handler for sqlexception
    begin
        rollback;
        resignal;
    end;
```

```
    set transaction isolation level read committed;
    start transaction;
    IF EXISTS( SELECT *
               FROM Partecipante
               WHERE `Codice Fiscale`= var_cf) THEN
     update Partecipante set `Nome`=var_nome,`Cognome`=var_cognome,`E-mail`=var_email,`Data Nascita` =
var_data_n,`Nazionalità`=nazionalita
    where `Codice Fiscale`= var_cf;
    ELSE
    insert into Partecipante VALUES (var_cf,var_nome,var_cognome,var_email,var_data_n,nazionalita);
END IF;

insert into Impegno VALUES (var_cf,id_prenotazione);
commit;
END
```

disdici_prenotazione

Usata dai clienti per disdire una prenotazione.

```
CREATE PROCEDURE `disdici_prenotazione` (in id_prenotazione int,in codice varchar(45))
BEGIN
declare supp int;
declare exit handler for sqlexception
    begin
        rollback;
        resignal;
    end;

   set transaction isolation level read committed;
   start transaction;
   SELECT IF( EXISTS(
              SELECT *
              FROM Prenotazione
              WHERE `ID Prenotazione`= id_prenotazione and `Codice Utente`=md5(codice)
              ), 1, 0) into supp;

    IF (supp=0) THEN
    signal sqlstate '45001' set message_text="Dati inserti non validi. Prenotazione non disdicibile o
inesistente";
END IF;


DELETE FROM Impegno WHERE `ID Prenotazione`=id_prenotazione;
DELETE FROM Prenotazione WHERE `ID Prenotazione`=id_prenotazione;
    commit;

END
```

lista_viaggi_prenotabili_input_id

Usata dai clienti per visualizzare le edizioni dei viaggi prenotabili per un certo programma di viaggio.

```
CREATE PROCEDURE `lista_viaggi_prenotabili_input_id` (in var_id_programma int)
BEGIN
declare exit handler for sqlexception
    begin
        rollback;
        resignal;
    end;

    set transaction isolation level read committed;
    set transaction read only;
    start transaction;
 SELECT `Data di Partenza`,`Data di Rientro`,`Costo` FROM `Viaggio`
 WHERE `ID Programma`=var_id_programma AND (`Stato Viaggio`='prenotabile');
 commit;
END
```

get_prenotazione_info

Usato dai clienti per ottenere le informazioni di una prenotazione da loro effettuata.

```
CREATE PROCEDURE `get_prenotazione_info` (in id_prenotazione int, in codice varchar(45))
BEGIN
declare supp int;

declare exit handler for sqlexception
    begin
        rollback;
        resignal;
    end;

    set transaction isolation level read committed;
    set transaction read only;
    start transaction;
    SELECT IF(EXISTS(
             SELECT *
             FROM Prenotazione
             WHERE `ID Prenotazione`= id_prenotazione and `Codice Utente`=md5(codice)
             ), 1, 0) into supp;

IF (supp=0) THEN
signal sqlstate '45001' set message_text="Dati inserti non validi. Prenotazione non disdicibile o
inesistente";
END IF;

select `Nome`,`Cognome` from Impegno
```

```
join Partecipante on `Codice Fiscale`=`Codice Fiscale Partecipante`
where Impegno.`ID Prenotazione` = id_prenotazione;

select `Data`,`Nome Programma`,Viaggio.`Data di Partenza`,`Costo` from Prenotazione
join `Programma Di Viaggio` on Prenotazione.`ID Programma`=`Programma Di Viaggio`.`ID Programma`
join Viaggio on Viaggio.`ID Programma`=`Programma Di Viaggio`.`ID Programma`
where `ID Prenotazione` = id_prenotazione;
   commit;
END
```

report_viaggio

Usata dai segretari per generare un report di un viaggio terminato.

```
CREATE PROCEDURE `report_viaggio` (in id_programma int,in data_p date)
BEGIN
declare num_partecipanti int;
declare num_giorni int;
declare costo_viaggio_per_persona double;
declare costo_autobus_giornaliero_tot double;
declare costo_albergo_per_persona_tot double;
declare profitto double;
declare stato_viaggio ENUM('in programma', 'attivo', 'terminato', 'prenotabile');
declare exit handler for sqlexception
    begin
        rollback;
        resignal;
    end;
drop temporary table if exists `ListaCosti`;
          create temporary table `ListaCosti` (
                    `Costo Per Persona` double,
                    `Costo Totale Autobus` double,
                    `Costo Totale Alberghi` double,
                    `Profitto` double,
        `Num Partecipanti` int
          );

    set transaction isolation level read committed;
    start transaction;
    -- si osservi che il report può essere generato solo per viaggi 'terminati', a questi non è
possibile l assegnazione di alberghi/auotbus privati ne è possibile che avvenga  alcuna prenotazione

select `Stato Viaggio` from Viaggio
where `ID Programma`= id_programma and `Data di Partenza`= data_p into stato_viaggio;

if(stato_viaggio<>'terminato')then
signal sqlstate '45001' set message_text="Un report può essere generato solo per viaggi terminati.";
end if;

select `Nome`,`Cognome`,`Data Nascita`,`E-mail`,`Nazionalità` from Partecipante
join Impegno on `Codice Fiscale`=`Codice Fiscale Partecipante`
join Prenotazione on Impegno.`ID Prenotazione`=Prenotazione.`ID Prenotazione`
```

```
join Viaggio on  Viaggio.`ID Programma`=Prenotazione.`ID Programma` AND Viaggio.`Data di
Partenza`=Prenotazione.`Data di Partenza`
where Viaggio.`ID Programma`=id_programma and Viaggio.`Data di Partenza`=data_p;


select Cost from Viaggio
where`ID Programma`=id_programma and `Data di Partenza`=data_p
into costo_viaggio_per_persona;

select  `Num. Partecipanti Confermati` from Viaggio
where `ID Programma`=id_programma and `Data di Partenza`= data_p
into num_partecipanti;

select `Numero Giorni` from Viaggio
join `Programma Di Viaggio` on Viaggio.`ID Programma`=`Programma Di Viaggio`.`ID Programma`
where Viaggio.`ID Programma`=id_programma and Viaggio.`Data di Partenza`=data_p
into num_giorni;

select sum(`Costo Giornaliero`) from Trasporto
join `Autobus Privato` on Trasporto.`Targa`=`Autobus Privato`.`Targa`
where `ID Programma`=id_programma and `Data di Partenza`=data_p
into costo_autobus_giornaliero_tot;

select sum(tot) from(
select `Costo Per Notte Per Persona`*`Numero Notti` as tot
from Pernottamenti
join Albergo on Pernottamenti.`Indirizzo`=Albergo.`Indirizzo` and  Pernottamenti.`Nome
Località`=Albergo.`Nome Località`
where `ID Programma`=id_programma and `Data di Partenza`=data_p)src
into costo_albergo_per_persona_tot;

set profitto=(num_partecipanti*costo_viaggio_per_persona)-
(costo_albergo_per_persona_tot*num_partecipanti)-(costo_autobus_giornaliero_tot*num_giorni);

insert into `ListaCosti` VALUES
(costo_viaggio_per_persona,costo_autobus_giornaliero_tot*num_giorni,costo_albergo_per_persona_tot*num_pa
rtecipanti,profitto,num_partecipanti);

select * from `ListaCosti`;

   commit;

END
```

Appendice: Implementazione

Codice SQL per instanziare il database

```
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 agenzia_viaggi
-- -----------------------------------------------------

-- -----------------------------------------------------
-- Schema agenzia_viaggi
-- -----------------------------------------------------
CREATE SCHEMA IF NOT EXISTS `agenzia_viaggi` ;
USE `agenzia_viaggi` ;

-- -----------------------------------------------------
-- Table `agenzia_viaggi`.`Programma Di Viaggio`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `agenzia_viaggi`.`Programma Di Viaggio` (
  `ID Programma` INT NOT NULL AUTO_INCREMENT,
  `Nome Programma` VARCHAR(45) NOT NULL,
  `Numero Giorni` INT NOT NULL,
  PRIMARY KEY (`ID Programma`))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `agenzia_viaggi`.`Viaggio`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `agenzia_viaggi`.`Viaggio` (
  `Data di Partenza` DATE NOT NULL,
  `ID Programma` INT NOT NULL,
  `Data di Rientro` DATE NOT NULL,
  `Costo` DOUBLE NOT NULL,
  `Stato Viaggio` ENUM('in programma', 'attivo', 'terminato', 'prenotabile') NOT NULL DEFAULT
'prenotabile',
  `Num. Partecipanti Confermati` INT NULL,
  PRIMARY KEY (`Data di Partenza`, `ID Programma`),
  INDEX `fk_Viaggio_1_idx` (`ID Programma` ASC) VISIBLE,
  CONSTRAINT `fk_Viaggio_1`
    FOREIGN KEY (`ID Programma`)
    REFERENCES `agenzia_viaggi`.`Programma Di Viaggio` (`ID Programma`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

```
-- -----------------------------------------------------
-- Table `agenzia_viaggi`.`Località`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `agenzia_viaggi`.`Località` (
  `Nome` VARCHAR(45) NOT NULL,
  `Stato` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`Nome`))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `agenzia_viaggi`.`Tappa`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `agenzia_viaggi`.`Tappa` (
  `Giorno Arrivo` INT NOT NULL,
  `Ora Arrivo` VARCHAR(45) NOT NULL,
  `ID Programma` INT NOT NULL,
  `Giorno Partenza` INT NOT NULL,
  `Ora Partenza` VARCHAR(45) NOT NULL,
  `Trattamento` ENUM('FB', 'HB', 'BB', 'OB', 'NB') NOT NULL,
  `Nome Località` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`Giorno Arrivo`, `Ora Arrivo`, `ID Programma`),
  INDEX `fk_Tappa_1_idx` (`ID Programma` ASC) VISIBLE,
  INDEX `fk_Tappa_2_idx` (`Nome Località` ASC) VISIBLE,
  CONSTRAINT `fk_Tappa_1`
    FOREIGN KEY (`ID Programma`)
    REFERENCES `agenzia_viaggi`.`Programma Di Viaggio` (`ID Programma`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Tappa_2`
    FOREIGN KEY (`Nome Località`)
    REFERENCES `agenzia_viaggi`.`Località` (`Nome`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `agenzia_viaggi`.`Partecipante`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `agenzia_viaggi`.`Partecipante` (
  `Codice Fiscale` CHAR(16) NOT NULL,
  `Nome` VARCHAR(45) NOT NULL,
  `Cognome` VARCHAR(45) NOT NULL,
  `E-mail` VARCHAR(45) NOT NULL,
  `Data Nascita` DATE NOT NULL,
  `Nazionalità` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`Codice Fiscale`))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `agenzia_viaggi`.`Prenotazione`
-- -----------------------------------------------------
```

```
CREATE TABLE IF NOT EXISTS `agenzia_viaggi`.`Prenotazione` (
  `ID Prenotazione` INT NOT NULL AUTO_INCREMENT,
  `Data` VARCHAR(45) NOT NULL,
  `Codice Utente` VARCHAR(45) NULL,
  `Data di Partenza` DATE NOT NULL,
  `ID Programma` INT NOT NULL,
  PRIMARY KEY (`ID Prenotazione`),
  INDEX `fk_Prenotazione_1_idx` (`Data di Partenza` ASC, `ID Programma` ASC) VISIBLE,
  CONSTRAINT `fk_Prenotazione_1`
    FOREIGN KEY (`Data di Partenza` , `ID Programma`)
    REFERENCES `agenzia_viaggi`.`Viaggio` (`Data di Partenza` , `ID Programma`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `agenzia_viaggi`.`Impegno`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `agenzia_viaggi`.`Impegno` (
  `Codice Fiscale Partecipante` CHAR(16) NOT NULL,
  `ID Prenotazione` INT NOT NULL,
  PRIMARY KEY (`Codice Fiscale Partecipante`, `ID Prenotazione`),
  INDEX `fk_Impegno_1_idx` (`ID Prenotazione` ASC) VISIBLE,
  CONSTRAINT `fk_Impegno_1`
    FOREIGN KEY (`ID Prenotazione`)
    REFERENCES `agenzia_viaggi`.`Prenotazione` (`ID Prenotazione`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Impegno_2`
    FOREIGN KEY (`Codice Fiscale Partecipante`)
    REFERENCES `agenzia_viaggi`.`Partecipante` (`Codice Fiscale`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `agenzia_viaggi`.`Guida`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `agenzia_viaggi`.`Guida` (
  `Codice Fiscale` CHAR(16) NOT NULL,
  `Nome` VARCHAR(45) NOT NULL,
  `Cognome` VARCHAR(45) NOT NULL,
  `E-mail` VARCHAR(45) NOT NULL,
  `Data Nascita` DATE NOT NULL,
  PRIMARY KEY (`Codice Fiscale`))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `agenzia_viaggi`.`Accompagno`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `agenzia_viaggi`.`Accompagno` (
```

```
  `Data di Partenza` DATE NOT NULL,
  `ID Programma` INT NOT NULL,
  `Codice Fiscale Guida` CHAR(16) NOT NULL,
  PRIMARY KEY (`Data di Partenza`, `ID Programma`, `Codice Fiscale Guida`),
  INDEX `fk_Accompagno_2_idx` (`Codice Fiscale Guida` ASC) VISIBLE,
  CONSTRAINT `fk_Accompagno_1`
    FOREIGN KEY (`Data di Partenza` , `ID Programma`)
    REFERENCES `agenzia_viaggi`.`Viaggio` (`Data di Partenza` , `ID Programma`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Accompagno_2`
    FOREIGN KEY (`Codice Fiscale Guida`)
    REFERENCES `agenzia_viaggi`.`Guida` (`Codice Fiscale`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `agenzia_viaggi`.`Albergo`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `agenzia_viaggi`.`Albergo` (
  `Indirizzo` VARCHAR(45) NOT NULL,
  `Nome Località` VARCHAR(45) NOT NULL,
  `Costo Per Notte Per Persona` DOUBLE NOT NULL,
  `Referente` VARCHAR(45) NOT NULL,
  `Nome` VARCHAR(45) NOT NULL,
  `Fax` VARCHAR(45) NOT NULL,
  `E-mail` VARCHAR(45) NOT NULL,
  `Telefono` VARCHAR(45) NOT NULL,
  `Capienza max. persone` INT NOT NULL,
  PRIMARY KEY (`Indirizzo`, `Nome Località`),
  INDEX `fk_Albergo_1_idx` (`Nome Località` ASC) VISIBLE,
  CONSTRAINT `fk_Albergo_1`
    FOREIGN KEY (`Nome Località`)
    REFERENCES `agenzia_viaggi`.`Località` (`Nome`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `agenzia_viaggi`.`Pernottamenti`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `agenzia_viaggi`.`Pernottamenti` (
  `Data di Partenza` DATE NOT NULL,
  `ID Programma` INT NOT NULL,
  `Indirizzo` VARCHAR(45) NOT NULL,
  `Nome Località` VARCHAR(45) NOT NULL,
  `Numero Notti` INT NOT NULL,
  PRIMARY KEY (`Data di Partenza`, `ID Programma`, `Indirizzo`, `Nome Località`),
  INDEX `fk_Pernottamenti_1_idx` (`Data di Partenza` ASC, `ID Programma` ASC) VISIBLE,
  INDEX `fk_Pernottamenti_2_idx` (`Indirizzo` ASC, `Nome Località` ASC) VISIBLE,
  CONSTRAINT `fk_Pernottamenti_1`
```

```
    FOREIGN KEY (`Data di Partenza` , `ID Programma`)
    REFERENCES `agenzia_viaggi`.`Viaggio` (`Data di Partenza` , `ID Programma`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Pernottamenti_2`
    FOREIGN KEY (`Indirizzo` , `Nome Località`)
    REFERENCES `agenzia_viaggi`.`Albergo` (`Indirizzo` , `Nome Località`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `agenzia_viaggi`.`Autobus Privato`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `agenzia_viaggi`.`Autobus Privato` (
  `Targa` CHAR(7) NOT NULL,
  `Nome Deposito` VARCHAR(45) NOT NULL,
  `Capienza max. persone` INT NOT NULL,
  `Costo giornaliero` DOUBLE NOT NULL,
  PRIMARY KEY (`Targa`))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `agenzia_viaggi`.`Trasporto`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `agenzia_viaggi`.`Trasporto` (
  `Data di Partenza` DATE NOT NULL,
  `ID Programma` INT NOT NULL,
  `Targa` CHAR(7) NOT NULL,
  PRIMARY KEY (`Data di Partenza`, `ID Programma`, `Targa`),
  INDEX `fk_Trasporto_1_idx` (`Targa` ASC) VISIBLE,
  CONSTRAINT `fk_Trasporto_1`
    FOREIGN KEY (`Targa`)
    REFERENCES `agenzia_viaggi`.`Autobus Privato` (`Targa`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Trasporto_2`
    FOREIGN KEY (`Data di Partenza` , `ID Programma`)
    REFERENCES `agenzia_viaggi`.`Viaggio` (`Data di Partenza` , `ID Programma`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `agenzia_viaggi`.`Segretari`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `agenzia_viaggi`.`Segretari` (
  `username` VARCHAR(45) NOT NULL,
  `password` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`username`))
ENGINE = InnoDB;
```

```
SET SQL_MODE=@OLD_SQL_MODE;
SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS;
SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS;
```

Codice del Front-End

main.c

```
1.  #include <stdio.h>
2.  #include <stdlib.h>
3.  #include <string.h>
4.  #include <mysql.h>
5.  #include "model/model.h"
6.  #include "utils/db.h"
7.  #include "defines.h"
8.  struct configuration conf;
9.
10. static MYSQL *conn;
11.
12. static role_t attempt_login(MYSQL *conn, char *username, char *password) {
13.   MYSQL_STMT *login_procedure;
14.
15.   MYSQL_BIND param[3];
16.   int role = 0;
17.
18.   if(!setup_prepared_stmt(&login_procedure, "call login_segretario(?, ?, ?)", conn)) {
19.             print_stmt_error(login_procedure, "Unable to initialize login statement\n");
20.             goto err2;
21.   }
22.     set_binding_param(¶m[0], MYSQL_TYPE_VAR_STRING, username, strlen(username));
23.     set_binding_param(¶m[1], MYSQL_TYPE_VAR_STRING, password, strlen(password));
24.     set_binding_param(¶m[2], MYSQL_TYPE_LONG, &role, sizeof(role));
25.
26.   if (mysql_stmt_bind_param(login_procedure, param) != 0) {
27.             print_stmt_error(login_procedure, "Could not bind parameters for login");
28.             goto err;
29.   }
30.
31.   if (mysql_stmt_execute(login_procedure) != 0) {
32.             print_stmt_error(login_procedure, "Could not execute login procedure");
33.             goto err;
34.   }
35.
36.     set_binding_param(¶m[0], MYSQL_TYPE_LONG, &role, sizeof(role));
37.
38.     if(mysql_stmt_bind_result(login_procedure, param)) {
39.             print_stmt_error(login_procedure, "Could not retrieve output parameter");
40.             goto err;
41.   }
42.
43.   if(mysql_stmt_fetch(login_procedure)) {
44.             print_stmt_error(login_procedure, "Could not buffer results");
45.             goto err;
46.   }
47.
48.   mysql_stmt_close(login_procedure);
49.   return role;
50.
51.     err:
52.   mysql_stmt_close(login_procedure);
53.     err2:
54.   return FAILED_LOGIN;
55. }
```

```
56.
57. int main(void) {
58.     char options[3] = {'1','2', '3'};
59.     char op;
60.     role_t role;
61.     printf("\033[2J\033[H");
62.     printf("*** CONNECT TO TRAVEL AGENCY ***\n\n");
63.     printf("1) Sono un cliente\n");
64.     printf("2) Sono un segretario\n");
65.     printf("3) Quit\n");
66.
67.     op = multiChoice("Select an option", options, 3);
68.
69.     switch(op) {
70.         case '1':
71.             role=CLIENTE;
72.         case '2':
73.             break;
74.         case '3':
75.             exit(0);
76.         default:
77.             fprintf(stderr, "Invalid condition at %s:%d\n", __FILE__, __LINE__);
78.             abort();
79.     }
80.
81.   if(!parse_config("users/login.json", &conf)) {
82.             fprintf(stderr, "Unable to load login configuration\n");
83.             exit(EXIT_FAILURE);
84.   }
85.
86.   conn = mysql_init (NULL);
87.   if (conn == NULL) {
88.             fprintf (stderr, "mysql_init() failed (probably out of memory)\n");
89.             exit(EXIT_FAILURE);
90.   }
91.
92.   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) {
93.             fprintf (stderr, "mysql_real_connect() failed\n");
94.             mysql_close (conn);
95.             exit(EXIT_FAILURE);
96.   }
97.     if(op=='2') {
98.         printf("\033[2J\033[H");
99.         printf("*** Identificati con le tue credenziali da segretario *** \n\n");
100.          printf("Username: ");
101.          getInput(128, conf.username, false);
102.          printf("Password: ");
103.          getInput(128, conf.password, true);
104.
105.          role = attempt_login(conn, conf.username, conf.password);
106.      }
107.
108.           switch(role) {
109.                     case CLIENTE:
110.                               run_as_cliente(conn);
111.                               break;
112.
113.                     case SEGRETARIO:
114.                               run_as_segretario(conn);
115.                               break;
116.
117.                     case FAILED_LOGIN:
118.                               fprintf(stderr, "Invalid credentials\n");
119.                               exit(EXIT_FAILURE);
120.
121.                     default:
122.                               fprintf(stderr, "Invalid condition at %s:%d\n", __FILE__,
    __LINE__);
123.                               abort();
```

```
124.            }
125.
126.            printf("Bye!\n");
127.
128.            mysql_close (conn);
129.            return 0;
130.    }
131.
```

clientectrl.c

```
1.  #include <stdio.h>
2.  #include <stdlib.h>
3.  #include <string.h>
4.  #include <ctype.h>
5.  #include "model/model.h"
6.  #include "defines.h"
7.  #include "view/clienteview.c"
8.  #include "dao/clientedao.c"
9.
10. static void list_programmi_di_viaggio_ctrl(MYSQL *conn)
11. {
12.     struct
    programma_di_viaggio_support*programmaDiViaggioSupport=lista_programmi_di_viaggio_cliente(conn);
13.     if(programmaDiViaggioSupport != NULL) {
14.         print_programmi_available(programmaDiViaggioSupport);
15.         free(programmaDiViaggioSupport);
16.     }
17.         struct programma_di_viaggio programmaDiViaggio;
18.         memset(&programmaDiViaggio, 0, sizeof(programmaDiViaggio));
19.         get_id_info_for_tappe(&programmaDiViaggio);
20.         struct tappa_support *tappaSupport = list_tappe_for_a_program(conn,
    &programmaDiViaggio);
21.         if (tappaSupport != NULL) {
22.             print_itinerario(tappaSupport);
23.             free(tappaSupport);
24.         }
25.     }
26.
27. static void list_available_viaggi(MYSQL *conn)
28. {
29.     struct programma_di_viaggio programmaDiViaggio;
30.     memset(&programmaDiViaggio, 0, sizeof(programmaDiViaggio));
31.     get_id_info_for_viaggi(&programmaDiViaggio);
32.     struct viaggio_support * viaggioSupport=list_viaggi_for_a_program(conn,&programmaDiViaggio);
33.     if(viaggioSupport != NULL) {
34.         print_viaggi_available(viaggioSupport);
35.         free(viaggioSupport);
36.     }
37. }
38.
39. static void prenota_viaggio_ctrl(MYSQL *conn)
40. {
41.     struct prenotazione prenotazione;
42.     memset(&prenotazione,0,sizeof (prenotazione));
43.     get_prenotazione_info(&prenotazione);
44.     prenota_viaggio(conn,&prenotazione);
45. }
46. static void visualizza_prenotazione_ctrl(MYSQL *conn)
47. {
48.     struct prenotazione prenotazione;
49.     memset(&prenotazione,0,sizeof (prenotazione));
50.     get_prenotazione_to_manage_info(&prenotazione);
51.     struct prenotazione_support*prenotazioneSupport=load_prenotazione_info(conn,&prenotazione);
52.     if(prenotazioneSupport != NULL ) {
53.         print_info_prenotazione(prenotazioneSupport);
54.         free(prenotazioneSupport);
```

```
55.     }
56. }
57. static void gestisci_prenotazione_ctrl(MYSQL *conn)
58. {
59.     char op = prenotazione_get_action();
60.     struct partecipante partecipante;
61.     struct prenotazione prenotazione;
62.     if(op=='b') return;
63.     get_prenotazione_to_manage_info(&prenotazione);
64.     struct prenotazione_support*prenotazioneSupport=load_prenotazione_info(conn,&prenotazione);
65.     if(prenotazioneSupport!=NULL) {
66.         print_info_prenotazione(prenotazioneSupport);
67.         free(prenotazioneSupport);
68.         switch (op) {
69.             case '1':
70.                 memset(&partecipante, 0, sizeof(partecipante));
71.                 get_partecipante_info(&partecipante);
72.                 aggiungi_partecipante(conn, &partecipante, &prenotazione);
73.                 break;
74.             case '2':
75.                 disdici_prenotazione(conn, &prenotazione);
76.                 break;
77.         }
78.     }else{
79.         printf("\nQualcosa è  andato storto..Riprovare\n");
80.     }
81. }
82.
83.
84. void run_as_cliente(MYSQL *conn)
85. {
86.   printf("Switching to cliente role...\n");
87.
88.   if(!parse_config("users/cliente.json", &conf)) {
89.             fprintf(stderr, "Unable to load student configuration\n");
90.             exit(EXIT_FAILURE);
91.   }
92.
93.   if(mysql_change_user(conn, conf.db_username, conf.db_password, conf.database)) {
94.             fprintf(stderr, "mysql_change_user() failed\n");
95.             exit(EXIT_FAILURE);
96.   }
97.
98.   while(true) {
99.             char op=cliente_get_action();
100.                    switch(op) {
101.                                case '1':
102.                 list_programmi_di_viaggio_ctrl(conn);
103.                 break;
104.
105.                                case '2':
106.                 list_available_viaggi(conn);
107.                 break;
108.
109.                                case '3':
110.                 prenota_viaggio_ctrl(conn);
111.                 break;
112.                                case '4':
113.                 gestisci_prenotazione_ctrl(conn);
114.                 break;
115.              case '5':
116.                 visualizza_prenotazione_ctrl(conn);
117.                 break;
118.              case 'q':
119.                 return;
120.                                default:
121.                                          fprintf(stderr, "Invalid condition at %s:%d\n",
    __FILE__, __LINE__);
122.                                          abort();
123.                     }
```

```
124.
125.                     getchar();
126.             }
127.  }
128.
```

segretarioctrl.c

```
1.  #include <stdio.h>
2.  #include <stdlib.h>
3.  #include <string.h>
4.  #include "defines.h"
5.  #include "model/model.h"
6.  #include "utils/db.c"
7.  #include "view/segretarioview.c"
8.  #include "dao/segretariodao.c"
9.
10. static void register_localita(MYSQL *conn)
11. {
12.     struct localita localita;
13.     memset(&localita, 0, sizeof(localita));
14.     get_localita_info(&localita);
15.     aggiungi_localita(conn,&localita);
16.     return;
17. }
18.
19. static void do_report_ctrl(MYSQL *conn)
20. {
21.     struct viaggio viaggio;
22.     memset(&viaggio, 0, sizeof(viaggio));
23.     get_viaggio_info_for_report(&viaggio);
24.     struct report_support*reportSupport=do_report(conn,&viaggio);
25.     print_report(reportSupport);
26.     return;
27. }
28.
29. static void register_albergo(MYSQL *conn)
30. {
31.     struct albergo albergo;
32.     memset(&albergo, 0, sizeof(albergo));
33.     get_albergo_info(&albergo);
34.     aggiungi_albergo(conn,&albergo);
35.     return;
36. }
37.
38. static int  register_tappa(MYSQL *conn,struct programma_di_viaggio*programmaDiViaggio)
39. {
40.     struct tappa tappa;
41.     memset(&tappa, 0, sizeof(tappa));
42.     tappa.id_programma=programmaDiViaggio->id_programma;
43.     get_tappa_info(&tappa);
44.     return aggiungi_tappa(conn,&tappa);
45. }
46.
47. static void aggiungi_tappe_programma_di_viaggio(MYSQL *conn){
48.     struct programma_di_viaggio programmaDiViaggio;
49.     memset(&programmaDiViaggio, 0, sizeof(programmaDiViaggio));
50.     get_idprogramma_info(&programmaDiViaggio);
51.     struct tappa_support *tappaSupport = list_tappe_for_a_program(conn, &programmaDiViaggio);
52.     if (tappaSupport != NULL) {
53.         print_itinerario(tappaSupport);
54.         free(tappaSupport);
55.     }
56.     while (1) {
57.         register_tappa(conn, &programmaDiViaggio);
58.         if(!ask_for_another())break;
59.     }
```

```
60. }
61. static void register_guida(MYSQL *conn)
62. {
63.     struct  guida guida;
64.     memset(&guida, 0, sizeof(guida));
65.     get_guida_info(&guida);
66.     aggiungi_guida(conn,&guida);
67.     return;
68. }
69. static void register_programma_di_viaggio(MYSQL *conn)
70. {
71.     struct  programma_di_viaggio programmaDiViaggio;
72.     memset(&programmaDiViaggio, 0, sizeof(programmaDiViaggio));;
73.     get_programma_di_viaggio_info(&programmaDiViaggio);
74.     aggiungi_programma_di_viaggio(conn,&programmaDiViaggio);
75.     return;
76. }
77. static void register_autobus_privato(MYSQL *conn)
78. {
79.     struct  autobus_privato autobusPrivato;
80.     memset(&autobusPrivato, 0, sizeof(autobusPrivato));;
81.     get_autobus_privato_info(&autobusPrivato);
82.     aggiungi_autobus_privato(conn,&autobusPrivato);
83.     return;
84. }
85. static void register_viaggio(MYSQL *conn)
86. {
87.     list_programmi_di_viaggio_segretari(conn);
88.     struct  viaggio viaggio;
89.     memset(&viaggio, 0, sizeof(viaggio));
90.     get_viaggio_info(&viaggio);
91.     aggiungi_viaggio(conn,&viaggio);
92.     return;
93. }
94.
95.
96. static void assegna_guida_ctrl(MYSQL *conn){
97.     struct guida_support * guidaSupport=lista_assegnamenti_guide(conn);
98.     if(guidaSupport != NULL) {
99.         print_guide_assign(guidaSupport);
100.          free(guidaSupport);
101.      }
102.      struct guida guida;
103.      struct viaggio viaggio;
104.      memset(&viaggio, 0, sizeof(viaggio));
105.      memset(&guida, 0, sizeof(guida));
106.      get_info_for_assign_guida(&guida,&viaggio);
107.      assegna_guida(conn,&guida,&viaggio);
108.  }
109.  static void assegna_autobus_ctrl(MYSQL *conn){
110.      struct autobus_support*autobusSupport=lista_assegnamenti_autobus(conn);
111.      if(autobusSupport != NULL) {
112.          print_autobus_assign(autobusSupport);
113.          free(autobusSupport);
114.      }
115.      struct autobus_privato autobusPrivato;
116.      struct viaggio viaggio;
117.      memset(&viaggio, 0, sizeof(viaggio));
118.      memset(&autobusPrivato, 0, sizeof(autobusPrivato));
119.      get_info_for_assign_autobus(&autobusPrivato,&viaggio);
120.      assegna_autobus(conn,&autobusPrivato,&viaggio);
121.  }
122.  static void assegna_albergo_ctrl(MYSQL *conn){
123.      struct albergo albergo;
124.      struct viaggio viaggio;
125.      memset(&viaggio, 0, sizeof(viaggio));
126.      memset(&albergo, 0, sizeof(albergo));
127.      get_info_for_assign_albergo(&albergo,&viaggio);
128.      assegna_albergo(conn,&albergo,&viaggio);
129.  }
```

```c
static void list_available_viaggi(MYSQL *conn)
{
    struct viaggio viaggio;
    memset(&viaggio, 0, sizeof(viaggio));
    char r=ask_for_stato_richiesto();
    switch(r) {
        case '1':
            strcpy(viaggio.stato_viaggio, "in programma");
            break;
        case '2':
            strcpy(viaggio.stato_viaggio, "attivo");
            break;
        case '3':
            strcpy(viaggio.stato_viaggio, "terminato");
            break;
        case '4':
            strcpy(viaggio.stato_viaggio, "prenotabile");
            break;
        default:
            fprintf(stderr, "Invalid condition at %s:%d\n", __FILE__, __LINE__);
            abort();
    }
    struct viaggio_support * viaggioSupport=list_viaggi_by_status(conn,&viaggio);
    if(viaggioSupport != NULL) {
        print_viaggi_confermati(viaggioSupport,&viaggio);
        free(viaggioSupport);
    }
}
static void list_info_guide_ctrl(MYSQL *conn)
{
    struct guida_support*guidaSupport= list_info_guide(conn);
    if(guidaSupport != NULL) {
        print_info_guide(guidaSupport);
        free(guidaSupport);
    }
}
static void list_info_localita_ctrl(MYSQL *conn)
{
    struct localita_support*localitaSupport= list_info_localita(conn);
    if(localitaSupport != NULL) {
        print_info_localita(localitaSupport);
        free(localitaSupport);
    }
}
static void list_info_autobus_ctrl(MYSQL *conn)
{
    struct autobus_support*autobusSupport= list_info_autobus(conn);
    if(autobusSupport != NULL) {
        print_info_autobus(autobusSupport);
        free(autobusSupport);
    }
}
static void list_info_alberghi_ctrl(MYSQL *conn)
{
    struct albergo_support*albergoSupport= list_info_alberghi(conn);
    if(albergoSupport != NULL) {
        print_info_alberghi(albergoSupport);
        free(albergoSupport);
    }
}
static void list_programmi_di_viaggio_segretari_ctrl(MYSQL *conn){
    struct programma_di_viaggio_support*programmaDiViaggioSupport=
list_programmi_di_viaggio_segretari(conn);
    if(programmaDiViaggioSupport != NULL) {
        print_programmi_available_segretari(programmaDiViaggioSupport);
        free(programmaDiViaggioSupport);
    }
}
void run_as_segretario(MYSQL *conn) //segretario view
{
```

```
199.
200.      printf("Switching to segretario role...\n");
201.
202.      if(!parse_config("users/segretario.json", &conf)) {
203.          fprintf(stderr, "Unable to load administrator configuration\n");
204.          exit(EXIT_FAILURE);
205.      }
206.
207.      if(mysql_change_user(conn, conf.db_username, conf.db_password, conf.database)) {
208.          fprintf(stderr, "mysql_change_user() failed\n");
209.          exit(EXIT_FAILURE);
210.      }
211.
212.      while(true) {
213.          char op=segretario_get_action();
214.          switch(op) {
215.              case '1':
216.                  op=nested_get_action_1();
217.                  switch (op) {
218.                      case'1':
219.                          list_programmi_di_viaggio_segretari_ctrl(conn);
220.                          break;
221.                      case '2':
222.                          register_programma_di_viaggio(conn);
223.                          break;
224.                      case '3':
225.                          aggiungi_tappe_programma_di_viaggio(conn);
226.                          break;
227.                      case 'b':
228.                          break;
229.                  }
230.                  break;
231.              case '2':
232.                  op=nested_get_action_2();
233.                  switch (op) {
234.                      case '1':
235.                          list_available_viaggi(conn);
236.                          break;
237.                      case '2':
238.                          register_viaggio(conn);
239.                          break;
240.                      case '3':
241.                          assegna_guida_ctrl(conn);
242.                          break;
243.                      case '4':
244.                          assegna_autobus_ctrl(conn);
245.                          break;
246.                      case '5':
247.                          assegna_albergo_ctrl(conn);
248.                          break;
249.                      case 'b':
250.                          break;
251.                  }
252.                  break;
253.              case '3':
254.                  op=nested_get_action_6();
255.                  switch (op) {
256.                      case '1':
257.                          list_info_localita_ctrl(conn);
258.                          break;
259.                      case '2':
260.                          register_localita(conn);
261.                          break;
262.                      case 'b':
263.                          break;
264.                  }
265.                  break;
266.              case '4':
267.                  op=nested_get_action_3();
268.                  switch (op) {
```

```
                    case '1':
                        list_info_guide_ctrl(conn);
                        break;
                    case '2':
                        register_guida(conn);
                        break;
                    case 'b':
                        break;
                }
                break;
            case '5':
                op=nested_get_action_4();
                switch (op) {
                    case '1':
                        list_info_autobus_ctrl(conn);
                        break;
                    case '2':
                        register_autobus_privato(conn);
                        break;
                    case 'b':
                        break;
                }
                break;
            case '6':
                op=nested_get_action_5();
                switch (op) {
                    case '1':
                        list_info_alberghi_ctrl(conn);
                        break;
                    case '2':
                        register_albergo(conn);
                        break;
                    case 'b':
                        break;
                }
                break;
            case '7':
                do_report_ctrl(conn);
                break;
            case 'q':
                return;
            default:
                fprintf(stderr, "Invalid condition at %s:%d\n", __FILE__, __LINE__);
                abort();
        }

        getchar();
    }
}

```

dao\clientedao.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "../defines.h"
#include "../model/model.h"
#include "../utils/db.h"
#include "clientedao.h"

static void aggiungi_partecipante(MYSQL *conn,struct partecipante * partecipante,struct
prenotazione*prenotazione){
    MYSQL_STMT *prepared_stmt;
    MYSQL_BIND param[7];
    MYSQL_TIME datanascita;
```

```
13.     if(!setup_prepared_stmt(&prepared_stmt, "call aggiungi_partecipante(?, ?, ?, ?, ?, ?, ?)",
    conn)) {
14.         finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize aggiungi_partecipante
    statement\n", false);
15.     }
16.     date_to_mysql_time(partecipante->data_nascita, &datanascita);
17.     set_binding_param(¶m[0],MYSQL_TYPE_LONG,&(prenotazione->id_prenotazione),sizeof
    (prenotazione->id_prenotazione));
18.     set_binding_param(¶m[1],MYSQL_TYPE_VAR_STRING,partecipante->cf,strlen(partecipante->cf));
19.     set_binding_param(¶m[2],MYSQL_TYPE_VAR_STRING,partecipante->nome,strlen(partecipante-
    >nome));
20.     set_binding_param(¶m[3],MYSQL_TYPE_VAR_STRING,partecipante->cognome,strlen(partecipante-
    >cognome));
21.     set_binding_param(¶m[4],MYSQL_TYPE_VAR_STRING,partecipante->email,strlen(partecipante-
    >email));
22.     set_binding_param(¶m[5],MYSQL_TYPE_DATE,&datanascita,sizeof(datanascita));
23.     set_binding_param(¶m[6],MYSQL_TYPE_VAR_STRING,partecipante->nazionalità,strlen(partecipante-
    >nazionalità));
24.
25.     if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
26.         finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters for
    aggiungi_partecipante\n", true);
27.     }
28.
29.     if (mysql_stmt_execute(prepared_stmt) != 0) {
30.         print_stmt_error (prepared_stmt, "Si è  verificato un problema nell'aggiunta del
    partecipante");
31.     } else {
32.         printf("Partecipante correttamente aggiunto\n");
33.     }
34.
35.     mysql_stmt_close(prepared_stmt);
36. }
37. static void disdici_prenotazione(MYSQL *conn,struct prenotazione*prenotazione){
38.     MYSQL_STMT *prepared_stmt;
39.     MYSQL_BIND param[2];
40.     if(!setup_prepared_stmt(&prepared_stmt, "call disdici_prenotazione(?,?)", conn)) {
41.         finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize disdici_prenotazione
    statement\n", false);
42.     }
43.
44.     set_binding_param(¶m[0],MYSQL_TYPE_LONG,&(prenotazione->id_prenotazione),sizeof
    (prenotazione->id_prenotazione));
45.     set_binding_param(¶m[1],MYSQL_TYPE_VAR_STRING,&(prenotazione-
    >codice_per_gestire),strlen(prenotazione->codice_per_gestire));
46.
47.     if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
48.         finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters for
    disdici_prenotazione\n", true);
49.     }
50.
51.     if (mysql_stmt_execute(prepared_stmt) != 0) {
52.         print_stmt_error (prepared_stmt, "Si è  verificato un problema nel disdire la
    prenotazione");
53.     } else {
54.         printf("\nPrenotazione correttamente disdetta \n");
55.     }
56.
57.     mysql_stmt_close(prepared_stmt);
58. }
59. static struct programma_di_viaggio_support *  lista_programmi_di_viaggio_cliente(MYSQL *conn){
60.     MYSQL_STMT *prepared_stmt;
61.     MYSQL_BIND param[4];
62.     int status;
63.     size_t row = 0;
64.     double costomin;
65.     int id;
66.     char nomeprogramma[MAX_LEN];
67.     int numgiorni;
68.
```

```
69.     struct programma_di_viaggio_support *programmaDiViaggioSupport = NULL;
70.     if(!setup_prepared_stmt(&prepared_stmt, "call lista_programmi_di_viaggio_cliente()", conn))
    {
71.         finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize
    lista_programmi_di_viaggio_cliente statement\n", false);
72.     }
73.     if (mysql_stmt_execute(prepared_stmt) != 0) {
74.         finish_with_stmt_error(conn, prepared_stmt,"Si è  verificato un problema nel mostrare i
    programmi di viaggio", true);
75.     }
76.
77.     mysql_stmt_store_result(prepared_stmt);
78.
79.     programmaDiViaggioSupport= malloc(sizeof(*programmaDiViaggioSupport) + sizeof(struct
    programma_di_viaggio) * mysql_stmt_num_rows(prepared_stmt));
80.
81.     if(programmaDiViaggioSupport == NULL)
82.         goto out;
83.
84.     memset(programmaDiViaggioSupport, 0, sizeof(*programmaDiViaggioSupport) + sizeof(struct
    programma_di_viaggio) * mysql_stmt_num_rows(prepared_stmt));
85.
86.     programmaDiViaggioSupport->num_entries = mysql_stmt_num_rows(prepared_stmt);
87.
88.     set_binding_param(¶m[0], MYSQL_TYPE_VAR_STRING, &nomeprogramma, MAX_LEN);
89.     set_binding_param(¶m[1], MYSQL_TYPE_LONG, &numgiorni, sizeof (int));
90.     set_binding_param(¶m[2], MYSQL_TYPE_LONG, &id, sizeof (int));
91.     set_binding_param(¶m[3], MYSQL_TYPE_DOUBLE, &costomin, sizeof(costomin));
92.
93.     if(mysql_stmt_bind_result(prepared_stmt, param)) {
94.         print_stmt_error(prepared_stmt, "Unable to bind output parameters for
    lista_programmi_di_viaggio_cliente\n");
95.         free(programmaDiViaggioSupport);
96.         programmaDiViaggioSupport = NULL;
97.         goto out;
98.     }
99.
100.      while (true) {
101.          status = mysql_stmt_fetch(prepared_stmt);
102.
103.          if (status == 1 || status == MYSQL_NO_DATA)
104.              break;
105.          programmaDiViaggioSupport->programmiDiViaggio[row].id_programma=id;
106.          programmaDiViaggioSupport->programmiDiViaggio[row].num_giorni=numgiorni;
107.          programmaDiViaggioSupport->programmiDiViaggio[row].min_price=costomin;
108.          strcpy(programmaDiViaggioSupport-
    >programmiDiViaggio[row].nome_programma,nomeprogramma);
109.          row++;
110.      }
111.      out:
112.      while(mysql_stmt_next_result(prepared_stmt) != -1) {}
113.      mysql_stmt_free_result(prepared_stmt);
114.      mysql_stmt_reset(prepared_stmt);
115.      mysql_stmt_close(prepared_stmt);
116.      return programmaDiViaggioSupport;
117.
118.  }
119.  static struct tappa_support *  list_tappe_for_a_program(MYSQL *conn,struct
    programma_di_viaggio*programmaDiViaggio){
120.      int status;
121.      size_t row = 0;
122.      MYSQL_STMT *prepared_stmt;
123.      char nome_localita[46];
124.      int data_arrivo;
125.      MYSQL_TIME ora_arrivo;
126.      int data_partenza;
127.      MYSQL_TIME ora_partenza;
128.      char trattamento[3];
129.      struct tappa_support*tappaSupport= NULL;
130.
```

```
131.      MYSQL_BIND param[6];
132.      if(!setup_prepared_stmt(&prepared_stmt, "call get_itinerario_input_id(?)", conn)) {
133.          finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize
    get_itinerario_input_id statement\n", false);
134.      }
135.      set_binding_param(¶m[0],MYSQL_TYPE_LONG,&(programmaDiViaggio->id_programma),sizeof
    (programmaDiViaggio->id_programma));
136.      if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
137.          finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters for
    get_itinerario_input_id\n", true);
138.      }
139.
140.      if (mysql_stmt_execute(prepared_stmt) != 0) {
141.          finish_with_stmt_error(conn, prepared_stmt, "Si è  verificato un problema nell'
    ottenere l'itinerario\n", true);
142.      }
143.      mysql_stmt_store_result(prepared_stmt);
144.
145.      tappaSupport= malloc(sizeof(*tappaSupport) + sizeof(struct tappa) *
    mysql_stmt_num_rows(prepared_stmt));
146.
147.      if(tappaSupport == NULL)
148.          goto out;
149.
150.      memset(tappaSupport, 0, sizeof(*tappaSupport) + sizeof(struct tappa) *
    mysql_stmt_num_rows(prepared_stmt));
151.
152.      tappaSupport->num_entries = mysql_stmt_num_rows(prepared_stmt);
153.
154.      set_binding_param(¶m[0], MYSQL_TYPE_VAR_STRING, nome_localita, MAX_LEN);
155.      set_binding_param(¶m[1], MYSQL_TYPE_LONG, &data_arrivo, sizeof (data_arrivo));
156.      set_binding_param(¶m[2], MYSQL_TYPE_TIME, &ora_arrivo, sizeof (ora_arrivo));
157.      set_binding_param(¶m[3], MYSQL_TYPE_LONG, &data_partenza, sizeof (data_partenza));
158.      set_binding_param(¶m[4], MYSQL_TYPE_TIME, &ora_partenza, sizeof (ora_partenza));
159.      set_binding_param(¶m[5], MYSQL_TYPE_VAR_STRING, trattamento, 3);
160.
161.      if(mysql_stmt_bind_result(prepared_stmt, param)) {
162.          print_stmt_error(prepared_stmt, "Unable to bind output parameters for
    get_itinerario_input_id\n");
163.          free(tappaSupport);
164.          tappaSupport = NULL;
165.          goto out;
166.      }
167.
168.      while (true) {
169.          status = mysql_stmt_fetch(prepared_stmt);
170.
171.          if (status == 1 || status == MYSQL_NO_DATA)
172.              break;
173.          strcpy(tappaSupport->tappe[row].nome_località,nome_localita);
174.          tappaSupport->tappe[row].giorno_arrivo=data_arrivo;
175.          mysql_time_to_string(&ora_arrivo,tappaSupport->tappe[row].ora_arrivo);
176.          tappaSupport->tappe[row].giorno_partenza=data_partenza;
177.          mysql_time_to_string(&ora_partenza,tappaSupport->tappe[row].ora_partenza);
178.          strcpy(tappaSupport->tappe[row].trattamento,trattamento);
179.          row++;
180.      }
181.      out:
182.      mysql_stmt_free_result(prepared_stmt);
183.      while(mysql_stmt_next_result(prepared_stmt) != -1) {}
184.      mysql_stmt_reset(prepared_stmt);
185.      mysql_stmt_close(prepared_stmt);
186.      return tappaSupport;
187.
188.  }
189.  static void prenota_viaggio(MYSQL *conn,struct prenotazione*prenotazione){
190.      MYSQL_STMT *prepared_stmt;
191.      MYSQL_BIND param[6];
192.      MYSQL_TIME datapartenza;
```

```
193.      if(!setup_prepared_stmt(&prepared_stmt, "call aggiungi_prenotazione(? ,?, ?, ?, ?, ?)",
    conn)) {
194.          finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize
    aggiungi_prenotazione statement\n", false);
195.      }
196.      date_to_mysql_time(prenotazione->data_partenza, &datapartenza);
197.
198.      set_binding_param(¶m[0],MYSQL_TYPE_DATE,&datapartenza,sizeof (datapartenza));
199.      set_binding_param(¶m[1],MYSQL_TYPE_LONG,&(prenotazione->id_programma),sizeof
    (prenotazione->id_programma));
200.      set_binding_param(¶m[2],MYSQL_TYPE_LONG,&(prenotazione->num_partecipanti),sizeof
    (prenotazione->num_partecipanti));
201.      set_binding_param(¶m[3],MYSQL_TYPE_VAR_STRING,&(prenotazione->codice_per_gestire),
    CODE_LEN);
202.      set_binding_param(¶m[4],MYSQL_TYPE_LONG,&(prenotazione->id_prenotazione), sizeof
    (prenotazione->id_prenotazione));
203.      set_binding_param(¶m[5],MYSQL_TYPE_DOUBLE,&(prenotazione->prezzo_tot_prenotazione), sizeof
    (prenotazione->prezzo_tot_prenotazione));
204.
205.      if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
206.          finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters for
    aggiungi_prenotazione\n", true);
207.      }
208.
209.      // Run procedure
210.      if (mysql_stmt_execute(prepared_stmt) != 0) {
211.          print_stmt_error (prepared_stmt, "Si è verificato un problema nell'aggiungere la
    prenotazione");
212.      }
213.      set_binding_param(¶m[0],MYSQL_TYPE_VAR_STRING,&(prenotazione->codice_per_gestire),
    CODE_LEN);
214.      set_binding_param(¶m[1],MYSQL_TYPE_LONG,&(prenotazione->id_prenotazione), sizeof
    (prenotazione->id_prenotazione));
215.      set_binding_param(¶m[2],MYSQL_TYPE_DOUBLE,&(prenotazione->prezzo_tot_prenotazione), sizeof
    (prenotazione->prezzo_tot_prenotazione));
216.
217.      if(mysql_stmt_bind_result(prepared_stmt, param)) {
218.          print_stmt_error(prepared_stmt, "Could not retrieve output parameter");
219.          goto out;
220.      }
221.
222.      if(mysql_stmt_fetch(prepared_stmt)) {
223.          print_stmt_error(prepared_stmt, "Could not buffer results");
224.          goto out;
225.      }else{
226.          printf("\nPrenotazione avvenuta con successo!\n");
227.          printf("\n[Prezzo Stimato Prenotazione] %.2f $ (per %d partecipanti)\n",prenotazione-
    >prezzo_tot_prenotazione,prenotazione->num_partecipanti);
228.          printf("[ID: %d] [codice %s] con queste credenziali potrai aggiunge i dati relativi ai
    partecipanti oppure disdire la tua prenotazione\n",prenotazione->id_prenotazione,prenotazione-
    >codice_per_gestire);
229.      }
230.      out:
231.      mysql_stmt_close(prepared_stmt);
232.  }
233.  static struct viaggio_support * list_viaggi_for_a_program(MYSQL *conn,struct
    programma_di_viaggio*programmaDiViaggio){
234.      int status;
235.      size_t row = 0;
236.
237.      MYSQL_STMT *prepared_stmt;
238.      MYSQL_BIND param[3];
239.      MYSQL_TIME giorno_partenza;
240.      MYSQL_TIME giorno_rientro;
241.      double costo;
242.
243.      struct viaggio_support *viaggioSupport = NULL;
244.
245.      if(!setup_prepared_stmt(&prepared_stmt, "call lista_viaggi_prenotabili_input_id(?)",
    conn)) {
```

```
246.          finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize
    lista_viaggi_prenotabili_input_id statement\n", false);
247.      }
248.      set_binding_param(¶m[0],MYSQL_TYPE_LONG,&(programmaDiViaggio->id_programma),sizeof
    (programmaDiViaggio->id_programma));
249.
250.      if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
251.          finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters for
    lista_viaggi_prenotabili_input_id\n", true);
252.      }
253.      // Run procedure
254.      if (mysql_stmt_execute(prepared_stmt) != 0) {
255.          finish_with_stmt_error(conn, prepared_stmt, "Non è possibile mostrare la lista\n",
    true);
256.      }
257.
258.      mysql_stmt_store_result(prepared_stmt);
259.
260.      viaggioSupport= malloc(sizeof(*viaggioSupport) + sizeof(struct viaggio) *
    mysql_stmt_num_rows(prepared_stmt));
261.
262.      if(viaggioSupport == NULL)
263.          goto out;
264.
265.      memset(viaggioSupport, 0, sizeof(*viaggioSupport) + sizeof(struct viaggio) *
    mysql_stmt_num_rows(prepared_stmt));
266.
267.      viaggioSupport->num_entries = mysql_stmt_num_rows(prepared_stmt);
268.
269.      set_binding_param(¶m[0], MYSQL_TYPE_DATE, &giorno_partenza, sizeof(giorno_partenza));
270.      set_binding_param(¶m[1], MYSQL_TYPE_DATE, &giorno_rientro, sizeof (giorno_rientro));
271.      set_binding_param(¶m[2], MYSQL_TYPE_DOUBLE, &costo, sizeof(costo));
272.
273.      if(mysql_stmt_bind_result(prepared_stmt, param)) {
274.          print_stmt_error(prepared_stmt, "Unable to bind output parameters for
    lista_viaggi_prenotabili_input_id\n");
275.          free(viaggioSupport);
276.          viaggioSupport = NULL;
277.          goto out;
278.      }
279.
280.      /* assemble occupancy general information */
281.      while (true) {
282.          status = mysql_stmt_fetch(prepared_stmt);
283.
284.          if (status == 1 || status == MYSQL_NO_DATA)
285.              break;
286.          mysql_date_to_string(&giorno_partenza,viaggioSupport->viaggi[row].data_partenza);
287.          mysql_date_to_string(&giorno_rientro,viaggioSupport->viaggi[row].data_rientro);
288.          viaggioSupport->viaggi[row].costo = costo;
289.          row++;
290.      }
291.      out:
292.      while(mysql_stmt_next_result(prepared_stmt) != -1) {}
293.      mysql_stmt_free_result(prepared_stmt);
294.      mysql_stmt_reset(prepared_stmt);
295.      mysql_stmt_close(prepared_stmt);
296.      return viaggioSupport;
297.
298.  }
299.
300.  static struct prenotazione_support *
    extract_partecipanti_prenotazione_info(MYSQL_STMT*prepared_stmt){
301.      int status;
302.      size_t row = 0;
303.
304.      MYSQL_BIND param[2];
305.      char nome_cliente[MAX_LEN];
306.      char cognome_cliente[MAX_LEN];
307.      struct prenotazione_support *prenotazioneSupport = NULL;
```

```
308.      set_binding_param(¶m[0], MYSQL_TYPE_VAR_STRING, &nome_cliente,MAX_LEN);
309.      set_binding_param(¶m[1], MYSQL_TYPE_VAR_STRING, &cognome_cliente, MAX_LEN);
310.      if(mysql_stmt_bind_result(prepared_stmt, param)) {
311.          print_stmt_error(prepared_stmt, "Unable to bind column parameters \n");
312.          free(prenotazioneSupport);
313.          prenotazioneSupport = NULL;
314.          goto out;
315.      }
316.
317.      if (mysql_stmt_store_result(prepared_stmt)) {
318.          print_stmt_error(prepared_stmt, "Unable to store partecipanti info to result set.");
319.          goto out;
320.      }
321.      prenotazioneSupport= malloc(sizeof(*prenotazioneSupport)+  sizeof(struct partecipante) *
    mysql_stmt_num_rows(prepared_stmt));
322.      if(prenotazioneSupport == NULL)
323.          goto out;
324.      memset(prenotazioneSupport, 0, sizeof(*prenotazioneSupport) + sizeof(struct partecipante)
    * mysql_stmt_num_rows(prepared_stmt));
325.      prenotazioneSupport->num_entries = mysql_stmt_num_rows(prepared_stmt);
326.      while (true) {
327.          status = mysql_stmt_fetch(prepared_stmt);
328.          if (status == 1 || status == MYSQL_NO_DATA)
329.              break;
330.          strcpy(prenotazioneSupport->partecipanti[row].nome,nome_cliente);
331.          strcpy(prenotazioneSupport->partecipanti[row].cognome,cognome_cliente);
332.          row++;
333.      }
334.      out:
335.      return prenotazioneSupport;
336.  }
337.  static void extract_prenotazione_general_information(MYSQL_STMT*prepared_stmt,struct
    prenotazione_support*prenotazioneSupport){
338.      MYSQL_BIND param[4];
339.      MYSQL_TIME dataprenotazione;
340.      char nomeprogramma[MAX_LEN];
341.      MYSQL_TIME datapartenza;
342.      double prezzo_per_persona;
343.
344.      set_binding_param(¶m[0], MYSQL_TYPE_DATE, &dataprenotazione, sizeof (dataprenotazione));
345.      set_binding_param(¶m[1], MYSQL_TYPE_VAR_STRING, &nomeprogramma, MAX_LEN);
346.      set_binding_param(¶m[2], MYSQL_TYPE_DATE, &datapartenza, sizeof (datapartenza));
347.      set_binding_param(¶m[3], MYSQL_TYPE_DOUBLE, &prezzo_per_persona, sizeof
    (prezzo_per_persona));
348.
349.      if(mysql_stmt_bind_result(prepared_stmt, param)) {
350.          print_stmt_error(prepared_stmt, "Unable to bind column parameters\n");
351.          return;
352.      }
353.      if (mysql_stmt_store_result(prepared_stmt)) {
354.          print_stmt_error(prepared_stmt, "Unable to store general information to result set.");
355.          return;
356.      }
357.      mysql_stmt_fetch(prepared_stmt);
358.      mysql_date_to_string(&dataprenotazione,prenotazioneSupport->data_prenotazione);
359.      mysql_date_to_string(&datapartenza,prenotazioneSupport->data_partenza);
360.      strcpy(prenotazioneSupport->nome_programma,nomeprogramma);
361.      prenotazioneSupport->prezzo_per_persona=prezzo_per_persona;
362.      return;
363.  }
364.  static struct prenotazione_support * load_prenotazione_info(MYSQL *conn,struct
    prenotazione*prenotazione){
365.      struct prenotazione_support*prenotazioneSupport=NULL;
366.      int status;
367.      MYSQL_STMT *prepared_stmt;
368.      MYSQL_BIND param[2];
369.
370.      if(!setup_prepared_stmt(&prepared_stmt, "call get_prenotazione_info(?,?)", conn)) {
371.          finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize
    get_prenotazione_info statement\n", false);
```

```
372.      }
373.      set_binding_param(¶m[0],MYSQL_TYPE_LONG,&(prenotazione->id_prenotazione), sizeof
    (prenotazione->id_prenotazione));
374.      set_binding_param(¶m[1],MYSQL_TYPE_VAR_STRING,&(prenotazione->codice_per_gestire),
    strlen(prenotazione->codice_per_gestire));
375.
376.      if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
377.          finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters for
    get_prenotazione_info\n", true);
378.      }
379.      if (mysql_stmt_execute(prepared_stmt) != 0) {
380.          print_stmt_error (prepared_stmt,"");
381.          goto out;
382.      }
383.      prenotazioneSupport=extract_partecipanti_prenotazione_info(prepared_stmt);
384.      if(prenotazioneSupport==NULL) goto out;
385.      mysql_stmt_free_result(prepared_stmt);
386.      status = mysql_stmt_next_result(prepared_stmt);
387.      if (status > 0)
388.          finish_with_stmt_error(conn, prepared_stmt, "Unexpected condition", false);
389.      else if(!status)
    extract_prenotazione_general_information(prepared_stmt,prenotazioneSupport);
390.      out:
391.      mysql_stmt_free_result(prepared_stmt);
392.      while(mysql_stmt_next_result(prepared_stmt) != -1) {}
393.      mysql_stmt_close(prepared_stmt);
394.      mysql_stmt_reset(prepared_stmt);
395.      return prenotazioneSupport;
396.  }
397.
```

dao\clientedao.h

```
1.  #include <stdio.h>
2.  #include <stdlib.h>
3.  #include <string.h>
4.  #include "../defines.h"
5.  #include "../model/model.h"
6.  #include "../utils/db.h"
7.  static void aggiungi_partecipante(MYSQL *conn,struct partecipante * partecipante,struct
    prenotazione*prenotazione);
8.  static void disdici_prenotazione(MYSQL *conn,struct prenotazione*prenotazione);
9.  static struct programma_di_viaggio_support *  lista_programmi_di_viaggio_cliente(MYSQL *conn);
10. static struct tappa_support *  list_tappe_for_a_program(MYSQL *conn,struct
    programma_di_viaggio*programmaDiViaggio);
11. static void prenota_viaggio(MYSQL *conn,struct prenotazione*prenotazione);
12. static struct viaggio_support * list_viaggi_for_a_program(MYSQL *conn,struct
    programma_di_viaggio*programmaDiViaggio);
13. static struct prenotazione_support *
    extract_partecipanti_prenotazione_info(MYSQL_STMT*prepared_stmt);
14. static void extract_prenotazione_general_information(MYSQL_STMT*prepared_stmt,struct
    prenotazione_support*prenotazioneSupport);
15. static struct prenotazione_support * load_prenotazione_info(MYSQL *conn,struct
    prenotazione*prenotazione);
16.
```

dao\segretariodao.c

```
1.  #include <stdio.h>
2.  #include <stdlib.h>
3.  #include <string.h>
4.  #include "../defines.h"
5.  #include "../model/model.h"
6.  #include "segretariodao.h"
7.  static struct programma_di_viaggio_support * list_programmi_di_viaggio_segretari(MYSQL *conn){
```

```
8.      MYSQL_STMT *prepared_stmt;
9.      MYSQL_BIND param[3];
10.     int status;
11.     int id;
12.     size_t row = 0;
13.     char nomeprogramma[MAX_LEN];
14.     int numgiorni;
15.     struct programma_di_viaggio_support *programmaDiViaggioSupport = NULL;
16.
17.     if(!setup_prepared_stmt(&prepared_stmt, "call lista_programmi_viaggi_segretari()", conn)) {
18.         finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize
    lista_programmi_viaggi_segretari statement\n", false);
19.     }
20.
21.     if (mysql_stmt_execute(prepared_stmt) != 0) {
22.         finish_with_stmt_error(conn, prepared_stmt, "Errore nel restituire lista dei programmi
    di viaggio\n", true);
23.     }
24.     mysql_stmt_store_result(prepared_stmt);
25.
26.     programmaDiViaggioSupport= malloc(sizeof(*programmaDiViaggioSupport) + sizeof(struct
    programma_di_viaggio) * mysql_stmt_num_rows(prepared_stmt));
27.
28.     if(programmaDiViaggioSupport == NULL)
29.         goto out;
30.
31.     memset(programmaDiViaggioSupport, 0, sizeof(*programmaDiViaggioSupport) + sizeof(struct
    programma_di_viaggio) * mysql_stmt_num_rows(prepared_stmt));
32.
33.     programmaDiViaggioSupport->num_entries = mysql_stmt_num_rows(prepared_stmt);
34.
35.     set_binding_param(¶m[0], MYSQL_TYPE_VAR_STRING, &nomeprogramma, MAX_LEN);
36.     set_binding_param(¶m[1], MYSQL_TYPE_LONG, &numgiorni, sizeof (int));
37.     set_binding_param(¶m[2], MYSQL_TYPE_LONG, &id, sizeof (int));
38.
39.     if(mysql_stmt_bind_result(prepared_stmt, param)) {
40.         print_stmt_error(prepared_stmt, "Unable to bind output parameters
    lista_programmi_viaggi_segretari\n");
41.         free(programmaDiViaggioSupport);
42.         programmaDiViaggioSupport = NULL;
43.         goto out;
44.     }
45.
46.     while (true) {
47.         status = mysql_stmt_fetch(prepared_stmt);
48.
49.         if (status == 1 || status == MYSQL_NO_DATA)
50.             break;
51.         programmaDiViaggioSupport->programmiDiViaggio[row].id_programma=id;
52.         programmaDiViaggioSupport->programmiDiViaggio[row].num_giorni=numgiorni;
53.         strcpy(programmaDiViaggioSupport->programmiDiViaggio[row].nome_programma,nomeprogramma);
54.         row++;
55.     }
56.     out:
57.     while(mysql_stmt_next_result(prepared_stmt) != -1) {}
58.     mysql_stmt_free_result(prepared_stmt);
59.     mysql_stmt_reset(prepared_stmt);
60.     mysql_stmt_close(prepared_stmt);
61.     return programmaDiViaggioSupport;
62.
63.
64.
65. }
66.
67. static void aggiungi_guida(MYSQL *conn,struct guida * guida){
68.     MYSQL_STMT *prepared_stmt;
69.     MYSQL_BIND param[5];
70.     MYSQL_TIME datanascita;
71.     if(!setup_prepared_stmt(&prepared_stmt, "call aggiungi_guida(?, ?, ?, ?, ?)", conn)) {
```

```
72.         finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize
    aggiungi_guidastatement\n", false);
73.     }
74.
75.     date_to_mysql_time(guida->data_nascita, &datanascita);
76.     set_binding_param(¶m[0],MYSQL_TYPE_VAR_STRING,guida->cf,strlen(guida->cf));
77.     set_binding_param(¶m[1],MYSQL_TYPE_VAR_STRING,guida->nome,strlen(guida->nome));
78.     set_binding_param(¶m[2],MYSQL_TYPE_VAR_STRING,guida->cognome,strlen(guida->cognome));
79.     set_binding_param(¶m[3],MYSQL_TYPE_VAR_STRING,guida->email,strlen(guida->email));
80.     set_binding_param(¶m[4],MYSQL_TYPE_DATE,&datanascita,sizeof(datanascita));
81.
82.     if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
83.         finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters for
    aggiungi_guida\n", true);
84.     }
85.
86.     if (mysql_stmt_execute(prepared_stmt) != 0) {
87.         print_stmt_error (prepared_stmt, "Si è verificato un errore nell'aggiunta della
    guida.\n");
88.     } else {
89.         printf("Guida correttamente aggiunta!\n");
90.     }
91.
92.     mysql_stmt_close(prepared_stmt);
93. }
94. static void assegna_guida(MYSQL *conn,struct guida * guida,struct viaggio * viaggio){
95.     MYSQL_STMT *prepared_stmt;
96.     MYSQL_BIND param[3];
97.     MYSQL_TIME datapartenza;
98.     if(!setup_prepared_stmt(&prepared_stmt, "call assegna_guida(?, ?, ?)", conn)) {
99.         finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize assegna_guida
    statement\n", false);
100.      }
101.
102.      date_to_mysql_time(viaggio->data_partenza, &datapartenza);
103.      set_binding_param(¶m[0],MYSQL_TYPE_VAR_STRING,guida->cf,strlen(guida->cf));
104.      set_binding_param(¶m[1],MYSQL_TYPE_LONG,&(viaggio->id_programma),sizeof(viaggio-
    >id_programma));
105.      set_binding_param(¶m[2],MYSQL_TYPE_DATE,&datapartenza,sizeof(datapartenza));
106.
107.      if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
108.          finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters for
    assegna_guida\n", true);
109.      }
110.
111.      if (mysql_stmt_execute(prepared_stmt) != 0) {
112.          print_stmt_error (prepared_stmt, "Si è verificato un errore nell'assegnare la
    guida\n");
113.      } else {
114.          printf("Guida correttamente assegnata! \n");
115.      }
116.
117.      mysql_stmt_close(prepared_stmt);
118.  }
119.  static void assegna_autobus(MYSQL *conn,struct autobus_privato * autobusPrivato,struct viaggio
    * viaggio){
120.      MYSQL_STMT *prepared_stmt;
121.      MYSQL_BIND param[3];
122.      MYSQL_TIME datapartenza;
123.      if(!setup_prepared_stmt(&prepared_stmt, "call assegna_autobus_privato(?, ?, ?)", conn)) {
124.          finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize
    assegna_autobus_privato statement\n", false);
125.      }
126.
127.      date_to_mysql_time(viaggio->data_partenza, &datapartenza);
128.      set_binding_param(¶m[2],MYSQL_TYPE_VAR_STRING,autobusPrivato->targa,strlen(autobusPrivato-
    >targa));
129.      set_binding_param(¶m[0],MYSQL_TYPE_LONG,&(viaggio->id_programma),sizeof(viaggio-
    >id_programma));
130.      set_binding_param(¶m[1],MYSQL_TYPE_DATE,&datapartenza,sizeof(datapartenza));
```

```
131.
132.      if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
133.          finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters for
    assegna_autobus_privato\n", true);
134.      }
135.
136.      if (mysql_stmt_execute(prepared_stmt) != 0) {
137.          print_stmt_error (prepared_stmt, "Si è verificato un errore nell'assegnare l'autobus
    privato\n");
138.      } else {
139.          printf("Autobus correttamente assegnata! \n");
140.      }
141.
142.      mysql_stmt_close(prepared_stmt);
143.  }
144.  static void assegna_albergo(MYSQL *conn,struct albergo * albergo,struct viaggio * viaggio){
145.      MYSQL_STMT *prepared_stmt;
146.      MYSQL_BIND param[5];
147.      MYSQL_TIME datapartenza;
148.
149.      if(!setup_prepared_stmt(&prepared_stmt, "call assegna_albergo(?, ?, ?, ?, ?)", conn)) {
150.          finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize assegna_albergo
    statement\n", false);
151.      }
152.
153.      date_to_mysql_time(viaggio->data_partenza, &datapartenza);
154.      set_binding_param(00B6[0],MYSQL_TYPE_LONG,&(viaggio->id_programma),sizeof(viaggio-
    >id_programma));
155.      set_binding_param(¶m[1],MYSQL_TYPE_DATE,&datapartenza,sizeof(datapartenza));
156.      set_binding_param(¶m[2],MYSQL_TYPE_VAR_STRING,albergo->indirizzo,strlen(albergo-
    >indirizzo));
157.      set_binding_param(¶m[3],MYSQL_TYPE_VAR_STRING,albergo->nome_località,strlen(albergo-
    >nome_località));
158.      set_binding_param(¶m[4],MYSQL_TYPE_LONG,&(albergo->num_notti),sizeof(int));
159.
160.      if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
161.          finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters for
    assegna_albergo\n", true);
162.      }
163.
164.      if (mysql_stmt_execute(prepared_stmt) != 0) {
165.          print_stmt_error (prepared_stmt, "Si è verificato un errore nell'assegnazione
    dell'albergo\n");
166.      } else {
167.          printf("Albergo correttamente assegnato! \n");
168.      }
169.
170.      mysql_stmt_close(prepared_stmt);
171.  }
172.
173.  static struct tappa_support *  list_tappe_for_a_program(MYSQL *conn,struct
    programma_di_viaggio*programmaDiViaggio){
174.      int status;
175.      size_t row = 0;
176.      MYSQL_STMT *prepared_stmt;
177.      char nome_localita[46];
178.      int data_arrivo;
179.      MYSQL_TIME ora_arrivo;
180.      int data_partenza;
181.      MYSQL_TIME ora_partenza;
182.      char trattamento[3];
183.      struct tappa_support*tappaSupport= NULL;
184.
185.      MYSQL_BIND param[6];
186.      if(!setup_prepared_stmt(&prepared_stmt, "call get_itinerario_input_id(?)", conn)) {
187.          finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize
    get_itinerario_input_idstatement\n", false);
188.      }
189.      set_binding_param(¶m[0],MYSQL_TYPE_LONG,&(programmaDiViaggio->id_programma),sizeof
    (programmaDiViaggio->id_programma));
```

```
190.      if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
191.          finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters for
    get_itinerario_input_id\n", true);
192.      }
193.
194.      if (mysql_stmt_execute(prepared_stmt) != 0) {
195.          finish_with_stmt_error(conn, prepared_stmt, "Non è possibile resituire
    l'itinerario\n", true);
196.      }
197.      mysql_stmt_store_result(prepared_stmt);
198.
199.      tappaSupport= malloc(sizeof(*tappaSupport) + sizeof(struct tappa) *
    mysql_stmt_num_rows(prepared_stmt));
200.
201.      if(tappaSupport == NULL)
202.          goto out;
203.
204.      memset(tappaSupport, 0, sizeof(*tappaSupport) + sizeof(struct tappa) *
    mysql_stmt_num_rows(prepared_stmt));
205.      tappaSupport->num_entries = mysql_stmt_num_rows(prepared_stmt);
206.
207.      set_binding_param(¶m[0], MYSQL_TYPE_VAR_STRING, nome_localita, strlen(nome_localita));
208.      set_binding_param(¶m[1], MYSQL_TYPE_LONG, &data_arrivo, sizeof (data_arrivo));
209.      set_binding_param(¶m[2], MYSQL_TYPE_TIME, &ora_arrivo, sizeof (ora_arrivo));
210.      set_binding_param(¶m[3], MYSQL_TYPE_LONG, &data_partenza, sizeof (data_partenza));
211.      set_binding_param(¶m[4], MYSQL_TYPE_TIME, &ora_partenza, sizeof (ora_partenza));
212.      set_binding_param(¶m[5], MYSQL_TYPE_VAR_STRING, trattamento, 3);
213.
214.      if(mysql_stmt_bind_result(prepared_stmt, param)) {
215.          print_stmt_error(prepared_stmt, "Unable to bind output parameters for
    get_itinerario_input_id\n");
216.          free(tappaSupport);
217.          tappaSupport = NULL;
218.          goto out;
219.      }
220.
221.      while (true) {
222.          status = mysql_stmt_fetch(prepared_stmt);
223.
224.          if (status == 1 || status == MYSQL_NO_DATA)
225.              break;
226.          strcpy(tappaSupport->tappe[row].nome_località,nome_localita);
227.          tappaSupport->tappe[row].giorno_arrivo=data_arrivo;
228.          mysql_time_to_string(&ora_arrivo,tappaSupport->tappe[row].ora_arrivo);
229.          tappaSupport->tappe[row].giorno_partenza=data_partenza;
230.          mysql_time_to_string(&ora_partenza,tappaSupport->tappe[row].ora_partenza);
231.          strcpy(tappaSupport->tappe[row].trattamento,trattamento);
232.          row++;
233.      }
234.      out:
235.      mysql_stmt_free_result(prepared_stmt);
236.      while(mysql_stmt_next_result(prepared_stmt) != -1) {}
237.      mysql_stmt_reset(prepared_stmt);
238.      mysql_stmt_close(prepared_stmt);
239.      return tappaSupport;
240.
241.  }
242.  static void aggiungi_localita(MYSQL *conn,struct localita *localita)
243.  {
244.      MYSQL_STMT *prepared_stmt;
245.      MYSQL_BIND param[2];
246.
247.      if(!setup_prepared_stmt(&prepared_stmt, "call aggiungi_localita(?, ?)", conn)) {
248.          finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize aggiungi_localita
    statement\n", false);
249.      }
250.
251.      set_binding_param(¶m[0],MYSQL_TYPE_VAR_STRING,localita->nome_località,strlen(localita-
    >nome_località));
252.      set_binding_param(¶m[1],MYSQL_TYPE_VAR_STRING,localita->stato,strlen(localita->stato));
```

```
253.
254.      if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
255.          finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters for
    aggiungi_localita\n", true);
256.      }
257.
258.      if (mysql_stmt_execute(prepared_stmt) != 0) {
259.          print_stmt_error (prepared_stmt, "Si è verificato un errore nell'aggiungere la
    località\n");
260.      } else {
261.          printf("Località correttamente aggiunta! \n");
262.      }
263.
264.      mysql_stmt_close(prepared_stmt);
265.  }
266.
267.  static void aggiungi_albergo(MYSQL *conn, struct albergo*albergo)
268.  {
269.      MYSQL_STMT *prepared_stmt;
270.      MYSQL_BIND param[9];
271.
272.      if(!setup_prepared_stmt(&prepared_stmt, "call
    aggiungi_albergo(?, ?, ?, ?, ?, ?, ?, ?, ?)", conn)) {
273.          finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize aggiungi_albergo
    statement\n", false);
274.      }
275.
276.      set_binding_param(¶m[0],MYSQL_TYPE_VAR_STRING,albergo->indirizzo,strlen(albergo-
    >indirizzo));
277.      set_binding_param(¶m[1],MYSQL_TYPE_VAR_STRING,albergo->nome_località,strlen(albergo-
    >nome_località));
278.      set_binding_param(¶m[2],MYSQL_TYPE_DOUBLE,&(albergo->costo_per_notte),sizeof(albergo-
    >costo_per_notte));
279.      set_binding_param(¶m[3],MYSQL_TYPE_VAR_STRING,albergo->referente,strlen(albergo-
    >referente));
280.      set_binding_param(¶m[4],MYSQL_TYPE_VAR_STRING,albergo->nome_albergo,strlen(albergo-
    >nome_albergo));
281.      set_binding_param(¶m[5],MYSQL_TYPE_VAR_STRING,albergo->fax,strlen(albergo->fax));
282.      set_binding_param(¶m[6],MYSQL_TYPE_VAR_STRING,albergo->email,strlen(albergo->email));
283.      set_binding_param(¶m[7],MYSQL_TYPE_VAR_STRING,albergo->telefono,strlen(albergo-
    >telefono));
284.      set_binding_param(¶m[8],MYSQL_TYPE_LONG,&(albergo->capienza_max),sizeof(albergo-
    >capienza_max));
285.
286.      if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
287.          finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters for
    aggiungi_albergo\n", true);
288.      }
289.
290.      if (mysql_stmt_execute(prepared_stmt) != 0) {
291.          print_stmt_error(prepared_stmt, "Si è verificato un errore nell'aggiunta
    dell'albergo\n");
292.          goto out;
293.      } else {
294.          printf("Nuovo albergo correttamento aggiunto! \n");
295.      }
296.      out:
297.      mysql_stmt_close(prepared_stmt);
298.  }
299.  static struct viaggio_support * list_viaggi_by_status(MYSQL *conn, struct viaggio*viaggio){
300.      int status;
301.      size_t row = 0;
302.
303.      MYSQL_STMT *prepared_stmt;
304.      MYSQL_BIND param[5];
305.      MYSQL_TIME giorno_partenza;
306.      MYSQL_TIME giorno_rientro;
307.      double costo;
308.      int num_partecipanti;
309.      int id_programma;
```

```
310.
311.      struct viaggio_support *viaggioSupport = NULL;
312.
313.      if(!setup_prepared_stmt(&prepared_stmt, "call lista_viaggi_input_stato(?)", conn)) {
314.          finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize
    lista_viaggi_input_stato statement\n", false);
315.      }
316.      set_binding_param(¶m[0],MYSQL_TYPE_VAR_STRING,&(viaggio->stato_viaggio),sizeof (viaggio-
    >stato_viaggio));
317.      if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
318.          finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters for
    lista_viaggi_input_stato\n", true);
319.      }
320.      if (mysql_stmt_execute(prepared_stmt) != 0) {
321.          finish_with_stmt_error(conn, prepared_stmt, "Non è possibile restituire la lista dei
    viaggi\n", true);
322.      }
323.
324.      mysql_stmt_store_result(prepared_stmt);
325.
326.      viaggioSupport= malloc(sizeof(*viaggioSupport) + sizeof(struct viaggio) *
    mysql_stmt_num_rows(prepared_stmt));
327.
328.      if(viaggioSupport == NULL)
329.          goto out;
330.
331.      memset(viaggioSupport, 0, sizeof(*viaggioSupport) + sizeof(struct viaggio) *
    mysql_stmt_num_rows(prepared_stmt));
332.
333.      viaggioSupport->num_entries = mysql_stmt_num_rows(prepared_stmt);
334.
335.      set_binding_param(¶m[0], MYSQL_TYPE_DATE, &giorno_partenza, sizeof(giorno_partenza));
336.      set_binding_param(¶m[1], MYSQL_TYPE_DATE, &giorno_rientro, sizeof (giorno_rientro));
337.      set_binding_param(¶m[2], MYSQL_TYPE_DOUBLE, &costo, sizeof(costo));
338.      set_binding_param(¶m[3], MYSQL_TYPE_LONG, &num_partecipanti, sizeof(num_partecipanti));
339.      set_binding_param(¶m[4], MYSQL_TYPE_LONG, &id_programma, sizeof(id_programma));
340.
341.      if(mysql_stmt_bind_result(prepared_stmt, param)) {
342.          print_stmt_error(prepared_stmt, "Unable to bind output parameters for
    lista_viaggi_input_stato\n");
343.          free(viaggioSupport);
344.          viaggioSupport = NULL;
345.          goto out;
346.      }
347.
348.      while (true) {
349.          status = mysql_stmt_fetch(prepared_stmt);
350.
351.          if (status == 1 || status == MYSQL_NO_DATA)
352.              break;
353.          mysql_date_to_string(&giorno_partenza,viaggioSupport->viaggi[row].data_partenza);
354.          mysql_date_to_string(&giorno_rientro,viaggioSupport->viaggi[row].data_rientro);
355.          viaggioSupport->viaggi[row].costo = costo;
356.          viaggioSupport->viaggi[row].id_programma = id_programma;
357.          viaggioSupport->viaggi[row].num_partecipanti_confermati = num_partecipanti;
358.          row++;
359.      }
360.      out:
361.      while(mysql_stmt_next_result(prepared_stmt) != -1) {}
362.      mysql_stmt_free_result(prepared_stmt);
363.      mysql_stmt_reset(prepared_stmt);
364.      mysql_stmt_close(prepared_stmt);
365.      return viaggioSupport;
366.
367.  }
368.  static struct guida_support * lista_assegnamenti_guide(MYSQL *conn){
369.      int status;
370.      size_t row = 0;
371.
372.      MYSQL_STMT *prepared_stmt;
```

```
373.      MYSQL_BIND param[3];
374.      char cf[CF_LEN];
375.      MYSQL_TIME datapartenza;
376.      MYSQL_TIME datarientro;
377.
378.      struct guida_support *guidaSupport = NULL;
379.
380.      if(!setup_prepared_stmt(&prepared_stmt, "call lista_assegnamenti_guide()", conn)) {
381.          finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize
    lista_assegnamenti_guide statement\n", false);
382.      }
383.
384.      if (mysql_stmt_execute(prepared_stmt) != 0) {
385.          finish_with_stmt_error(conn, prepared_stmt, "Non è possibile  restituire gli
    assegnamenti delle guide\n", true);
386.      }
387.
388.      mysql_stmt_store_result(prepared_stmt);
389.
390.      guidaSupport= malloc(sizeof(*guidaSupport) + sizeof(struct guida) *
    mysql_stmt_num_rows(prepared_stmt));
391.
392.      if(guidaSupport == NULL)
393.          goto out;
394.
395.      memset(guidaSupport, 0, sizeof(*guidaSupport) + sizeof(struct guida) *
    mysql_stmt_num_rows(prepared_stmt));
396.
397.      guidaSupport->num_entries = mysql_stmt_num_rows(prepared_stmt);
398.
399.      set_binding_param(¶m[0], MYSQL_TYPE_VAR_STRING, &cf, MAX_LEN);
400.      set_binding_param(¶m[1], MYSQL_TYPE_DATE, &datapartenza, sizeof (datapartenza));
401.      set_binding_param(¶m[2], MYSQL_TYPE_DATE, &datarientro, sizeof (datarientro));
402.
403.
404.      if(mysql_stmt_bind_result(prepared_stmt, param)) {
405.          print_stmt_error(prepared_stmt, "Unable to bind output parameters for
    lista_assegnamenti_guide\n");
406.          free(guidaSupport);
407.          guidaSupport = NULL;
408.          goto out;
409.      }
410.
411.      while (true) {
412.          status = mysql_stmt_fetch(prepared_stmt);
413.
414.          if (status == 1 || status == MYSQL_NO_DATA)
415.              break;
416.          strcpy(guidaSupport->guide[row].cf,cf);
417.          mysql_date_to_string(&datapartenza,guidaSupport->guide[row].dataP);
418.          mysql_date_to_string(&datarientro,guidaSupport->guide[row].dataR);
419.          row++;
420.      }
421.      out:
422.      while(mysql_stmt_next_result(prepared_stmt) != -1) {}
423.      mysql_stmt_free_result(prepared_stmt);
424.      mysql_stmt_reset(prepared_stmt);
425.      mysql_stmt_close(prepared_stmt);
426.      return guidaSupport;
427.
428.  }
429.  static struct autobus_support * lista_assegnamenti_autobus(MYSQL *conn){
430.      int status;
431.      size_t row = 0;
432.
433.      MYSQL_STMT *prepared_stmt;
434.      MYSQL_BIND param[3];
435.      char targa[TARGA_LEN];
436.      MYSQL_TIME datapartenza;
437.      MYSQL_TIME datarientro;
```

```
438.
439.      struct autobus_support *autobusSupport = NULL;
440.
441.      if(!setup_prepared_stmt(&prepared_stmt, "call lista_assegnamenti_autobus()", conn)) {
442.          finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize
    lista_assegnamenti_autobus statement\n", false);
443.      }
444.
445.      if (mysql_stmt_execute(prepared_stmt) != 0) {
446.          finish_with_stmt_error(conn, prepared_stmt, "Non è possibile resitituire gli
    assegnamenti degli autobus\n", true);
447.      }
448.
449.      mysql_stmt_store_result(prepared_stmt);
450.
451.      autobusSupport= malloc(sizeof(*autobusSupport) + sizeof(struct autobus_privato) *
    mysql_stmt_num_rows(prepared_stmt));
452.
453.      if(autobusSupport == NULL)
454.          goto out;
455.
456.      memset(autobusSupport, 0, sizeof(*autobusSupport) + sizeof(struct autobus_privato) *
    mysql_stmt_num_rows(prepared_stmt));
457.
458.      autobusSupport->num_entries = mysql_stmt_num_rows(prepared_stmt);
459.
460.      set_binding_param(¶m[0], MYSQL_TYPE_VAR_STRING, &targa, MAX_LEN);
461.      set_binding_param(¶m[1], MYSQL_TYPE_DATE, &datapartenza, sizeof (datapartenza));
462.      set_binding_param(¶m[2], MYSQL_TYPE_DATE, &datarientro, sizeof (datarientro));
463.
464.
465.      if(mysql_stmt_bind_result(prepared_stmt, param)) {
466.          print_stmt_error(prepared_stmt, "Unable to bind output parameters for
    lista_assegnamenti_autobus\n");
467.          free(autobusSupport);
468.          autobusSupport = NULL;
469.          goto out;
470.      }
471.
472.      while (true) {
473.          status = mysql_stmt_fetch(prepared_stmt);
474.
475.          if (status == 1 || status == MYSQL_NO_DATA)
476.              break;
477.          strcpy(autobusSupport->autobusPrivato[row].targa,targa);
478.          mysql_date_to_string(&datapartenza,autobusSupport->autobusPrivato[row].dataP);
479.          mysql_date_to_string(&datarientro,autobusSupport->autobusPrivato[row].dataR);
480.          row++;
481.      }
482.      out:
483.      while(mysql_stmt_next_result(prepared_stmt) != -1) {}
484.      mysql_stmt_free_result(prepared_stmt);
485.      mysql_stmt_reset(prepared_stmt);
486.      mysql_stmt_close(prepared_stmt);
487.      return autobusSupport;
488.
489.  }
490.  static struct guida_support * list_info_guide(MYSQL *conn){
491.      int status;
492.      size_t row = 0;
493.
494.      MYSQL_STMT *prepared_stmt;
495.      MYSQL_BIND param[5];
496.      char cf[CF_LEN];
497.      char nome[MAX_LEN];
498.      char cognome[MAX_LEN];
499.      char email[MAX_LEN];
500.      MYSQL_TIME datanascita;
501.
502.      struct guida_support *guidaSupport = NULL;
```

```
503.
504.      if(!setup_prepared_stmt(&prepared_stmt, "call lista_guide_all()", conn)) {
505.          finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize lista_guide_all
    statement\n", false);
506.      }
507.      if (mysql_stmt_execute(prepared_stmt) != 0) {
508.          finish_with_stmt_error(conn, prepared_stmt, "Non è possibile restituire la lista delle
    guide\n", true);
509.      }
510.
511.      mysql_stmt_store_result(prepared_stmt);
512.
513.      guidaSupport= malloc(sizeof(*guidaSupport) + sizeof(struct guida) *
    mysql_stmt_num_rows(prepared_stmt));
514.
515.      if(guidaSupport == NULL)
516.          goto out;
517.
518.      memset(guidaSupport, 0, sizeof(*guidaSupport) + sizeof(struct guida) *
    mysql_stmt_num_rows(prepared_stmt));
519.
520.      guidaSupport->num_entries = mysql_stmt_num_rows(prepared_stmt);
521.
522.      set_binding_param(¶m[0], MYSQL_TYPE_VAR_STRING, &cf, MAX_LEN);
523.      set_binding_param(¶m[1], MYSQL_TYPE_VAR_STRING, &nome, MAX_LEN);
524.      set_binding_param(¶m[2], MYSQL_TYPE_VAR_STRING, &cognome, MAX_LEN);
525.      set_binding_param(¶m[3], MYSQL_TYPE_VAR_STRING, &email, MAX_LEN);
526.      set_binding_param(¶m[4], MYSQL_TYPE_DATE, &datanascita, sizeof (datanascita));
527.
528.      if(mysql_stmt_bind_result(prepared_stmt, param)) {
529.          print_stmt_error(prepared_stmt, "Unable to bind output parameters for
    lista_guide_all\n");
530.          free(guidaSupport);
531.          guidaSupport = NULL;
532.          goto out;
533.      }
534.
535.      while (true) {
536.          status = mysql_stmt_fetch(prepared_stmt);
537.
538.          if (status == 1 || status == MYSQL_NO_DATA)
539.              break;
540.          strcpy(guidaSupport->guide[row].nome,nome);
541.          strcpy(guidaSupport->guide[row].cf,cf);
542.          strcpy(guidaSupport->guide[row].cognome,cognome);
543.          strcpy(guidaSupport->guide[row].email,email);
544.          mysql_date_to_string(&datanascita,guidaSupport->guide[row].data_nascita);
545.          row++;
546.      }
547.      out:
548.      while(mysql_stmt_next_result(prepared_stmt) != -1) {}
549.      mysql_stmt_free_result(prepared_stmt);
550.      mysql_stmt_reset(prepared_stmt);
551.      mysql_stmt_close(prepared_stmt);
552.      return guidaSupport;
553.
554.  }
555.  static struct localita_support * list_info_localita(MYSQL *conn){
556.      int status;
557.      size_t row = 0;
558.
559.      MYSQL_STMT *prepared_stmt;
560.      MYSQL_BIND param[2];
561.      char nome[MAX_LEN];
562.      char stato[MAX_LEN];
563.
564.      struct localita_support *localitaSupport = NULL;
565.
566.      if(!setup_prepared_stmt(&prepared_stmt, "call lista_localita_all()", conn)) {
```

```
567.          finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize lista_localita_all
    statement\n", false);
568.      }
569.
570.      if (mysql_stmt_execute(prepared_stmt) != 0) {
571.          finish_with_stmt_error(conn, prepared_stmt, "Non è possibile restituire la lista delle
    guide\n", true);
572.      }
573.
574.      mysql_stmt_store_result(prepared_stmt);
575.
576.      localitaSupport= malloc(sizeof(*localitaSupport) + sizeof(struct localita) *
    mysql_stmt_num_rows(prepared_stmt));
577.
578.      if(localitaSupport == NULL)
579.          goto out;
580.
581.      memset(localitaSupport, 0, sizeof(*localitaSupport) + sizeof(struct localita) *
    mysql_stmt_num_rows(prepared_stmt));
582.
583.      localitaSupport->num_entries = mysql_stmt_num_rows(prepared_stmt);
584.
585.      set_binding_param(¶m[0], MYSQL_TYPE_VAR_STRING, &nome, MAX_LEN);
586.      set_binding_param(¶m[1], MYSQL_TYPE_VAR_STRING, &stato, MAX_LEN);
587.
588.      if(mysql_stmt_bind_result(prepared_stmt, param)) {
589.          print_stmt_error(prepared_stmt, "Unable to bind output parameters for
    lista_localita_all\n");
590.          free(localitaSupport);
591.          localitaSupport = NULL;
592.          goto out;
593.      }
594.
595.      while (true) {
596.          status = mysql_stmt_fetch(prepared_stmt);
597.
598.          if (status == 1 || status == MYSQL_NO_DATA)
599.              break;
600.          strcpy(localitaSupport->localita[row].nome_località,nome);
601.          strcpy(localitaSupport->localita[row].stato,stato);
602.          row++;
603.      }
604.      out:
605.      while(mysql_stmt_next_result(prepared_stmt) != -1) {}
606.      mysql_stmt_free_result(prepared_stmt);
607.      mysql_stmt_reset(prepared_stmt);
608.      mysql_stmt_close(prepared_stmt);
609.      return localitaSupport;
610.
611.  }
612.  static struct autobus_support * list_info_autobus(MYSQL *conn){
613.      int status;
614.      size_t row = 0;
615.
616.      MYSQL_STMT *prepared_stmt;
617.      MYSQL_BIND param[4];
618.      char targa[TARGA_LEN];
619.      char nomedep[MAX_LEN];
620.      int capienza;
621.      double costo;
622.
623.      struct autobus_support *autobusSupport = NULL;
624.
625.      if(!setup_prepared_stmt(&prepared_stmt, "call lista_autobus_all()", conn)) {
626.          finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize lista_autobus_all
    statement\n", false);
627.      }
628.
629.      if (mysql_stmt_execute(prepared_stmt) != 0) {
```

```
630.          finish_with_stmt_error(conn, prepared_stmt, "Non è possibile restituire la lista degli
    autobus\n", true);
631.      }
632.
633.      mysql_stmt_store_result(prepared_stmt);
634.
635.      autobusSupport= malloc(sizeof(*autobusSupport) + sizeof(struct autobus_privato) *
    mysql_stmt_num_rows(prepared_stmt));
636.
637.      if(autobusSupport == NULL)
638.          goto out;
639.
640.      memset(autobusSupport, 0, sizeof(*autobusSupport) + sizeof(struct autobus_privato) *
    mysql_stmt_num_rows(prepared_stmt));
641.
642.      autobusSupport->num_entries = mysql_stmt_num_rows(prepared_stmt);
643.
644.      set_binding_param(¶m[0], MYSQL_TYPE_VAR_STRING, &targa, TARGA_LEN);
645.      set_binding_param(¶m[1], MYSQL_TYPE_VAR_STRING, &nomedep, MAX_LEN);
646.      set_binding_param(¶m[2], MYSQL_TYPE_LONG, &capienza, sizeof(int));
647.      set_binding_param(¶m[3], MYSQL_TYPE_DOUBLE, &costo, sizeof(double));
648.
649.      if(mysql_stmt_bind_result(prepared_stmt, param)) {
650.          print_stmt_error(prepared_stmt, "Unable to bind output parameters for
    lista_autobus_all\n");
651.          free(autobusSupport);
652.          autobusSupport = NULL;
653.          goto out;
654.      }
655.
656.      while (true) {
657.          status = mysql_stmt_fetch(prepared_stmt);
658.
659.          if (status == 1 || status == MYSQL_NO_DATA)
660.              break;
661.          strcpy(autobusSupport->autobusPrivato[row].targa,targa);
662.          strcpy(autobusSupport->autobusPrivato[row].nome_deposito,nomedep);
663.          autobusSupport->autobusPrivato[row].capienza_max=capienza;
664.          autobusSupport->autobusPrivato[row].costo_giornaliero=costo;
665.          row++;
666.      }
667.      out:
668.      while(mysql_stmt_next_result(prepared_stmt) != -1) {}
669.      mysql_stmt_free_result(prepared_stmt);
670.      mysql_stmt_reset(prepared_stmt);
671.      mysql_stmt_close(prepared_stmt);
672.      return autobusSupport;
673.
674.  }
675.  static struct albergo_support * list_info_alberghi(MYSQL *conn){
676.      int status;
677.      size_t row = 0;
678.
679.      MYSQL_STMT *prepared_stmt;
680.      MYSQL_BIND param[9];
681.      char indirizzo[MAX_LEN];
682.      char localita[MAX_LEN];
683.      double costo;
684.      char referente[MAX_LEN];
685.      char nome[MAX_LEN];
686.      char fax[MAX_LEN];
687.      char email[MAX_LEN];
688.      char telefono[MAX_LEN];
689.      int capienza;
690.
691.      struct albergo_support *albergoSupport = NULL;
692.
693.      if(!setup_prepared_stmt(&prepared_stmt, "call lista_alberghi_all()", conn)) {
694.          finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize lista_alberghi_all
    statement\n", false);
```

```
695.      }
696.
697.      if (mysql_stmt_execute(prepared_stmt) != 0) {
698.          finish_with_stmt_error(conn, prepared_stmt, "Non è possibile restituire la lista degli
    alberghi\n", true);
699.      }
700.
701.      mysql_stmt_store_result(prepared_stmt);
702.
703.      albergoSupport= malloc(sizeof(*albergoSupport) + sizeof(struct albergo) *
    mysql_stmt_num_rows(prepared_stmt));
704.
705.      if(albergoSupport == NULL)
706.          goto out;
707.
708.      memset(albergoSupport, 0, sizeof(*albergoSupport) + sizeof(struct albergo) *
    mysql_stmt_num_rows(prepared_stmt));
709.
710.      albergoSupport->num_entries = mysql_stmt_num_rows(prepared_stmt);
711.
712.      set_binding_param(¶m[0], MYSQL_TYPE_VAR_STRING, &indirizzo, MAX_LEN);
713.      set_binding_param(¶m[1], MYSQL_TYPE_VAR_STRING, &localita, MAX_LEN);
714.      set_binding_param(¶m[2], MYSQL_TYPE_DOUBLE, &costo, sizeof(double));
715.      set_binding_param(¶m[3], MYSQL_TYPE_VAR_STRING, &referente, MAX_LEN);
716.      set_binding_param(¶m[4], MYSQL_TYPE_VAR_STRING, &nome, MAX_LEN);
717.      set_binding_param(¶m[5], MYSQL_TYPE_VAR_STRING, &fax,MAX_LEN);
718.      set_binding_param(¶m[6], MYSQL_TYPE_VAR_STRING, &email, MAX_LEN);
719.      set_binding_param(¶m[7], MYSQL_TYPE_VAR_STRING, &telefono, MAX_LEN);
720.      set_binding_param(¶m[8], MYSQL_TYPE_LONG, &capienza, sizeof(int));
721.
722.      if(mysql_stmt_bind_result(prepared_stmt, param)) {
723.          print_stmt_error(prepared_stmt, "Unable to bind output parameters for
    lista_alberghi_all\n");
724.          free(albergoSupport);
725.          albergoSupport = NULL;
726.          goto out;
727.      }
728.
729.      while (true) {
730.          status = mysql_stmt_fetch(prepared_stmt);
731.
732.          if (status == 1 || status == MYSQL_NO_DATA)
733.              break;
734.          strcpy(albergoSupport->alberghi[row].indirizzo,indirizzo);
735.          strcpy(albergoSupport->alberghi[row].nome_località,localita);
736.          strcpy(albergoSupport->alberghi[row].referente,referente);
737.          strcpy(albergoSupport->alberghi[row].nome_albergo,nome);
738.          strcpy(albergoSupport->alberghi[row].fax,fax);
739.          strcpy(albergoSupport->alberghi[row].telefono,telefono);
740.          albergoSupport->alberghi[row].capienza_max=capienza;
741.          albergoSupport->alberghi[row].costo_per_notte=costo;
742.          row++;
743.      }
744.      out:
745.      while(mysql_stmt_next_result(prepared_stmt) != -1) {}
746.      mysql_stmt_free_result(prepared_stmt);
747.      mysql_stmt_reset(prepared_stmt);
748.      mysql_stmt_close(prepared_stmt);
749.      return albergoSupport;
750.
751.  }
752.  static int aggiungi_tappa(MYSQL *conn,struct tappa*tappa)
753.  {
754.      int ret=0;
755.      MYSQL_STMT *prepared_stmt;
756.      MYSQL_BIND param[7];
757.      MYSQL_TIME oraPart;
758.      MYSQL_TIME oraArr;
759.
```

```
760.      if(!setup_prepared_stmt(&prepared_stmt, "call aggiungi_tappa(?, ?, ?, ?, ?, ?, ?)", conn))
    {
761.          finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize aggiungi_tappa
    statement\n", false);
762.      }
763.
764.      time_to_mysql_time(tappa->ora_partenza,&oraPart);
765.      time_to_mysql_time(tappa->ora_arrivo,&oraArr);
766.
767.      set_binding_param(¶m[0],MYSQL_TYPE_LONG,&(tappa->giorno_arrivo),sizeof (tappa-
    >giorno_arrivo));
768.      set_binding_param(¶m[1],MYSQL_TYPE_TIME,&(oraArr),sizeof (oraArr));
769.      set_binding_param(¶m[2],MYSQL_TYPE_LONG,&(tappa->id_programma),sizeof (tappa-
    >id_programma));
770.      set_binding_param(¶m[3],MYSQL_TYPE_LONG,&(tappa->giorno_partenza),sizeof (tappa-
    >giorno_partenza));
771.      set_binding_param(¶m[4],MYSQL_TYPE_TIME,&(oraPart),sizeof (oraPart));
772.      set_binding_param(¶m[5],MYSQL_TYPE_VAR_STRING,&(tappa->trattamento),strlen(tappa-
    >trattamento));
773.      set_binding_param(¶m[6],MYSQL_TYPE_VAR_STRING,&(tappa->nome_località),strlen(tappa-
    >nome_località));
774.
775.      if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
776.          finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters for
    aggiungi_tappa\n", true);
777.      }
778.
779.      if (mysql_stmt_execute(prepared_stmt) != 0) {
780.          print_stmt_error (prepared_stmt, "\n Non è stato possibile aggiungere questa tappa,
    riprova!\n ");
781.      } else {
782.          ret=1;
783.          printf("Tappa correctly added!\n");
784.      }
785.
786.      mysql_stmt_close(prepared_stmt);
787.      return ret;
788.  }
789.
790.  static void aggiungi_programma_di_viaggio(MYSQL *conn,struct
    programma_di_viaggio*programmaDiViaggio)
791.  {
792.      MYSQL_STMT *prepared_stmt;
793.      MYSQL_BIND param[3];
794.
795.      if(!setup_prepared_stmt(&prepared_stmt, "call aggiungi_programma_di_viaggio(? ,?, ?)",
    conn)) {
796.          finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize
    aggiungi_programma_di_viaggio statement\n", false);
797.      }
798.
799.      set_binding_param(¶m[0],MYSQL_TYPE_VAR_STRING,&(programmaDiViaggio->nome_programma),strlen
    (programmaDiViaggio->nome_programma));
800.      set_binding_param(¶m[1],MYSQL_TYPE_LONG,&(programmaDiViaggio->num_giorni),sizeof
    (programmaDiViaggio->num_giorni));
801.      set_binding_param(¶m[2],MYSQL_TYPE_LONG,&(programmaDiViaggio->id_programma),sizeof
    (programmaDiViaggio->id_programma));
802.
803.      if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
804.          finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters for
    aggiungi_programma_di_viaggio\n", true);
805.      }
806.
807.      if (mysql_stmt_execute(prepared_stmt) != 0) {
808.          print_stmt_error (prepared_stmt, "Si è verificato un errore nell'aggiungere il
    programma di viaggio\n");
809.      } else {
810.          printf("Programma di viaggio correctly registered...\n");
811.      }
```

```
812.      set_binding_param(¶m[0],MYSQL_TYPE_LONG,&(programmaDiViaggio->id_programma),sizeof
    (programmaDiViaggio->id_programma));
813.
814.      if(mysql_stmt_bind_result(prepared_stmt, param)) {
815.          print_stmt_error(prepared_stmt, "Could not retrieve output parameter for
    aggiungi_programma_di_viaggio");
816.          goto out;
817.      }
818.      if(mysql_stmt_fetch(prepared_stmt)) {
819.          print_stmt_error(prepared_stmt, "Could not buffer results for
    aggiungi_programma_di_viaggio");
820.          goto out;
821.      }else{
822.          printf("\nCreato programma di viaggio con successo! [ID: %d]\n",programmaDiViaggio-
    >id_programma);
823.      }
824.      out:
825.      mysql_stmt_free_result(prepared_stmt);
826.      while(mysql_stmt_next_result(prepared_stmt) != -1) {}
827.      mysql_stmt_free_result(prepared_stmt);
828.      mysql_stmt_reset(prepared_stmt);
829.
830.      mysql_stmt_close(prepared_stmt);
831.      return;
832.  }
833.  static struct report_support * extract_partecipanti_info(MYSQL_STMT*prepared_stmt){
834.      int status;
835.      size_t row = 0;
836.
837.      MYSQL_BIND param[5];
838.
839.      char nome_cliente[MAX_LEN];
840.      char cognome_cliente[MAX_LEN];
841.      MYSQL_TIME datanascita;
842.      char email[MAX_LEN];
843.      char nazionalità[MAX_LEN];
844.      struct report_support *reportSupport = NULL;
845.
846.      set_binding_param(¶m[0], MYSQL_TYPE_VAR_STRING, &nome_cliente,MAX_LEN);
847.      set_binding_param(¶m[1], MYSQL_TYPE_VAR_STRING, &cognome_cliente, MAX_LEN);
848.      set_binding_param(¶m[2], MYSQL_TYPE_DATE, &datanascita, sizeof(datanascita));
849.      set_binding_param(¶m[3], MYSQL_TYPE_VAR_STRING, &email, MAX_LEN);
850.      set_binding_param(¶m[4], MYSQL_TYPE_VAR_STRING, &nazionalità, MAX_LEN);
851.      if(mysql_stmt_bind_result(prepared_stmt, param)) {
852.          print_stmt_error(prepared_stmt, "Unable to bind column parameters for
    extract_partecipanti_info \n");
853.          free(reportSupport);
854.          reportSupport = NULL;
855.          goto out;
856.      }
857.
858.      if (mysql_stmt_store_result(prepared_stmt)) {
859.          print_stmt_error(prepared_stmt, "Unable to store partecipanti information result
    set.");
860.          goto out;
861.      }
862.      reportSupport= malloc(sizeof(*reportSupport)+  sizeof(struct partecipante) *
    mysql_stmt_num_rows(prepared_stmt));
863.      if(reportSupport == NULL)
864.          goto out;
865.      memset(reportSupport, 0, sizeof(*reportSupport) + sizeof(struct partecipante) *
    mysql_stmt_num_rows(prepared_stmt));
866.      reportSupport->num_entries = mysql_stmt_num_rows(prepared_stmt);
867.
868.      while (true) {
869.          status = mysql_stmt_fetch(prepared_stmt);
870.          if (status == 1 || status == MYSQL_NO_DATA)
871.              break;
872.          mysql_date_to_string(&datanascita,reportSupport->partecipanti[row].data_nascita);
873.          strcpy(reportSupport->partecipanti[row].email,email);
```

```
874.          strcpy(reportSupport->partecipanti[row].nazionalità,nazionalità);
875.          strcpy(reportSupport->partecipanti[row].nome,nome_cliente);
876.          strcpy(reportSupport->partecipanti[row].cognome,cognome_cliente);
877.          row++;
878.      }
879.      out:
880.      return reportSupport;
881.  }
882.  static void extract_costi_information(MYSQL_STMT*prepared_stmt,struct
    report_support*reportSupport){
883.      MYSQL_BIND param[5];
884.      double costo_per_persona;
885.      double costo_tot_autobus;
886.      double costo_tot_alberghi;
887.      double profitto;
888.      int num_partecipanti;
889.      set_binding_param(¶m[0], MYSQL_TYPE_DOUBLE, &costo_per_persona, sizeof
    (costo_per_persona));
890.      set_binding_param(¶m[1], MYSQL_TYPE_DOUBLE, &costo_tot_autobus, sizeof
    (costo_tot_autobus));
891.      set_binding_param(¶m[2], MYSQL_TYPE_DOUBLE, &costo_tot_alberghi, sizeof
    (costo_tot_alberghi));
892.      set_binding_param(¶m[3], MYSQL_TYPE_DOUBLE, &profitto, sizeof (profitto));
893.      set_binding_param(¶m[4], MYSQL_TYPE_LONG, &num_partecipanti, sizeof (num_partecipanti));
894.      if(mysql_stmt_bind_result(prepared_stmt, param)) {
895.          print_stmt_error(prepared_stmt, "Unable to bind column parameters for
    extract_costi_information \n");
896.          return;
897.      }
898.      if (mysql_stmt_store_result(prepared_stmt)) {
899.          print_stmt_error(prepared_stmt, "Unable to store general prenotazione information
    result set.\n");
900.          return;
901.      }
902.      mysql_stmt_fetch(prepared_stmt);
903.      reportSupport->num_partecipanti=num_partecipanti;
904.      reportSupport->costo_tot_alberghi=costo_tot_alberghi;
905.      reportSupport->costo_tot_autobus=costo_tot_autobus;
906.      reportSupport->profitto=profitto;
907.      reportSupport->costo_per_persona=costo_per_persona;
908.      return;
909.  }
910.  static struct report_support *do_report(MYSQL *conn,struct viaggio*viaggio)
911.  {
912.      struct report_support*reportSupport=NULL;
913.      int status;
914.      MYSQL_STMT *prepared_stmt;
915.      MYSQL_BIND param[2];
916.      MYSQL_TIME datapartenza;
917.
918.      date_to_mysql_time(viaggio->data_partenza, &datapartenza);
919.
920.      if(!setup_prepared_stmt(&prepared_stmt, "call report_viaggio(? ,?)", conn)) {
921.          finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize
    report_viaggiostatement\n", false);
922.      }
923.
924.      set_binding_param(¶m[0],MYSQL_TYPE_LONG,&(viaggio->id_programma),sizeof (viaggio-
    >id_programma));
925.      set_binding_param(¶m[1],MYSQL_TYPE_DATE,&(datapartenza),sizeof(datapartenza));
926.      if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
927.          finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters for
    report_viaggio\n", true);
928.          goto out;
929.      }
930.      if (mysql_stmt_execute(prepared_stmt) != 0) {
931.          print_stmt_error (prepared_stmt, "Si è verificato un errore nella generazione del
    Report\n");
932.          goto out;
933.      }
```

```
934.      reportSupport=extract_partecipanti_info(prepared_stmt);
935.      if(reportSupport==NULL)
936.          goto out;
937.      mysql_stmt_free_result(prepared_stmt);
938.      status = mysql_stmt_next_result(prepared_stmt);
939.      if (status > 0)
940.          finish_with_stmt_error(conn, prepared_stmt, "Unexpected condition", false);
941.      else if(!status) extract_costi_information(prepared_stmt,reportSupport);
942.      out:
943.      mysql_stmt_free_result(prepared_stmt);
944.      while(mysql_stmt_next_result(prepared_stmt) != -1) {}
945.      mysql_stmt_close(prepared_stmt);
946.      mysql_stmt_reset(prepared_stmt);
947.      return reportSupport;
948.  }
949.
950.  static void aggiungi_autobus_privato(MYSQL *conn,struct autobus_privato*autobusPrivato)
951.  {
952.      MYSQL_STMT *prepared_stmt;
953.      MYSQL_BIND param[4];
954.
955.      if(!setup_prepared_stmt(&prepared_stmt, "call aggiungi_autobus_privato(? ,?, ?, ?)",
    conn)) {
956.          finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize
    aggiungi_autobus_privato statement\n", false);
957.      }
958.
959.      set_binding_param(¶m[0],MYSQL_TYPE_VAR_STRING,autobusPrivato->targa,strlen
    (autobusPrivato->targa));
960.      set_binding_param(¶m[1],MYSQL_TYPE_VAR_STRING,autobusPrivato->nome_deposito,strlen
    (autobusPrivato->nome_deposito));
961.      set_binding_param(¶m[2],MYSQL_TYPE_LONG,&(autobusPrivato->capienza_max),sizeof
    (autobusPrivato->capienza_max));
962.      set_binding_param(¶m[3],MYSQL_TYPE_DOUBLE,&(autobusPrivato->costo_giornaliero),sizeof
    (autobusPrivato->costo_giornaliero));
963.
964.      if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
965.          finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters for
    aggiungi_autobus_privaton", true);
966.      }
967.
968.      if (mysql_stmt_execute(prepared_stmt) != 0) {
969.          print_stmt_error (prepared_stmt, "Si è verificato un errore nell'aggiungere l'autobus
    privato\n");
970.      } else {
971.          printf("Autobus privato correttamente registrato.\n");
972.      }
973.      mysql_stmt_close(prepared_stmt);
974.  }
975.
976.  static void aggiungi_viaggio(MYSQL *conn,struct viaggio*viaggio)
977.  {
978.      MYSQL_STMT *prepared_stmt;
979.      MYSQL_BIND param[4];
980.      MYSQL_TIME datapartenza;
981.      MYSQL_TIME datarientro;
982.
983.      if(!setup_prepared_stmt(&prepared_stmt, "call aggiungi_viaggio(?,?,?,?)", conn)) {
984.          finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize aggiungi_viaggio
    statement\n", false);
985.      }
986.
987.      date_to_mysql_time(viaggio->data_rientro, &datarientro);
988.      date_to_mysql_time(viaggio->data_partenza, &datapartenza);
989.
990.      set_binding_param(¶m[0],MYSQL_TYPE_DATE,&datapartenza,sizeof(datapartenza));
991.      set_binding_param(¶m[1],MYSQL_TYPE_LONG,&(viaggio->id_programma),sizeof (viaggio-
    >id_programma));
992.      set_binding_param(¶m[2],MYSQL_TYPE_DATE,&datarientro,sizeof(datarientro));
993.      set_binding_param(¶m[3],MYSQL_TYPE_DOUBLE,&(viaggio->costo),sizeof (viaggio->costo));
```

```
994.
995.      if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
996.          finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters for
    aggiungi_viaggio\n", true);
997.      }
998.
999.      if (mysql_stmt_execute(prepared_stmt) != 0) {
1000.         print_stmt_error (prepared_stmt, "Si è verificato un erroe nell'aggiungere  il
    viaggio\n");
1001.     } else {
1002.         printf("Viaggio correttamente aggiunto\n");
1003.     }
1004.     mysql_stmt_close(prepared_stmt);
1005. }
1006.
```

dao\segretariodao.h

```
1.  #include <stdio.h>
2.  #include <stdlib.h>
3.  #include <string.h>
4.  #include "../defines.h"
5.  #include "../model/model.h"
6.  static struct programma_di_viaggio_support * list_programmi_di_viaggio_segretari(MYSQL *conn);
7.  static void aggiungi_guida(MYSQL *conn,struct guida * guida);
8.  static void assegna_autobus(MYSQL *conn,struct autobus_privato * autobusPrivato,struct viaggio *
    viaggio);
9.  static struct tappa_support *  list_tappe_for_a_program(MYSQL *conn,struct
    programma_di_viaggio*programmaDiViaggio);
10. static void aggiungi_localita(MYSQL *conn,struct localita *localita);
11. static void aggiungi_albergo(MYSQL *conn, struct albergo*albergo);
12. static struct viaggio_support * list_viaggi_by_status(MYSQL *conn,struct viaggio*viaggio);
13. static struct guida_support * lista_assegnamenti_guide(MYSQL *conn);
14. static struct autobus_support * lista_assegnamenti_autobus(MYSQL *conn);
15. static struct guida_support * list_info_guide(MYSQL *conn);
16. static struct autobus_support * list_info_autobus(MYSQL *conn);
17. static struct albergo_support * list_info_alberghi(MYSQL *conn);
18. static int aggiungi_tappa(MYSQL *conn,struct tappa*tappa);
19. static void aggiungi_programma_di_viaggio(MYSQL *conn,struct
    programma_di_viaggio*programmaDiViaggio);
20. static struct report_support * extract_partecipanti_info(MYSQL_STMT*prepared_stmt);
21. static void extract_costi_information(MYSQL_STMT*prepared_stmt,struct
    report_support*reportSupport);
22. static struct report_support *do_report(MYSQL *conn,struct viaggio*viaggio);
23. static void aggiungi_autobus_privato(MYSQL *conn,struct autobus_privato*autobusPrivato);
24. static void aggiungi_viaggio(MYSQL *conn,struct viaggio*viaggio);
25.
```

model\model.h

```
1.  #pragma once
2.  #include <stdbool.h>
3.  #include <stdlib.h>
4.  #define TIME_LEN 6
5.  #define DATE_LEN 11
6.  #define MAX_LEN 45
7.  #define CF_LEN 17
8.  #define TARGA_LEN 8
9.  #define USERNAME_LEN 45
10. #define PASSWORD_LEN 45
11. #define MAX_LEN_STATO 13
12. #define TRATTAMENTO_LEN 3
13. #define SIZE_INPUT_NUMBER 11
14. #define CODE_LEN 8
15.
```

```
16. typedef enum {
17.     FAILED_LOGIN=0,
18.     SEGRETARIO,
19.     CLIENTE
20. } role_t;
21.
22. struct viaggio {
23.     char data_partenza[DATE_LEN];
24.     int id_programma;
25.     char data_rientro[DATE_LEN];
26.     double costo;
27.     char stato_viaggio[MAX_LEN_STATO];
28.     int num_partecipanti_confermati;
29. };
30. struct viaggio_support {
31.     unsigned num_entries;
32.     struct viaggio viaggi[];
33. };
34.
35. struct credentials {
36.   char username[USERNAME_LEN];
37.   char password[PASSWORD_LEN];
38. };
39.
40. struct autobus_privato{
41.     char targa[TARGA_LEN];
42.     char nome_deposito[MAX_LEN];
43.     int capienza_max;
44.     double costo_giornaliero;
45.     char dataP[DATE_LEN];
46.     char dataR[DATE_LEN];
47. };
48. struct autobus_support{
49.     unsigned num_entries;
50.     struct autobus_privato autobusPrivato[];
51. };
52.
53. struct guida{
54.     char cf[CF_LEN];
55.     char nome[MAX_LEN];
56.     char cognome[MAX_LEN];
57.     char email[MAX_LEN];
58.     char data_nascita[DATE_LEN];
59.     char dataP[DATE_LEN];
60.     char dataR[DATE_LEN];
61. };
62. struct partecipante{
63.     char cf[CF_LEN];
64.     char nome[MAX_LEN];
65.     char cognome[MAX_LEN];
66.     char email[MAX_LEN];
67.     char data_nascita[DATE_LEN];
68.     char dataP[DATE_LEN];
69.     char dataR[DATE_LEN];
70.     char nazionalità[MAX_LEN];
71. };
72.
73. struct report_support{
74.     unsigned num_entries;
75.     double costo_per_persona;
76.     double costo_tot_autobus;
77.     double costo_tot_alberghi;
78.     double profitto;
79.     int num_partecipanti;
80.     struct partecipante partecipanti[];
81. };
82.
83. struct guida_support{
84.     unsigned num_entries;
85.     struct guida guide[];
```

```
86. };
87. struct programma_di_viaggio {
88.     int id_programma;
89.     char nome_programma[MAX_LEN];
90.     int num_giorni;
91.     double min_price;
92. };
93. struct programma_di_viaggio_support{
94.     unsigned num_entries;
95.     struct programma_di_viaggio programmiDiViaggio[];
96. };
97. struct tappa{
98.     int giorno_arrivo;
99.     char ora_arrivo[TIME_LEN];
100.      int id_programma;
101.      int giorno_partenza;
102.      char ora_partenza[TIME_LEN];
103.      char trattamento[TRATTAMENTO_LEN];
104.      char nome_località[MAX_LEN];
105.  };
106.  struct tappa_support {
107.      unsigned num_entries;
108.      struct tappa tappe[];
109.  };
110.  struct prenotazione{
111.      int id_prenotazione;
112.      char codice_per_gestire[MAX_LEN];
113.      char data_partenza[DATE_LEN];
114.      int id_programma;
115.      int num_partecipanti;
116.      double prezzo_tot_prenotazione;
117.  };
118.  struct prenotazione_support {
119.      unsigned num_entries;
120.      char data_prenotazione[DATE_LEN];
121.      char nome_programma[MAX_LEN];
122.      char data_partenza[MAX_LEN];
123.      double prezzo_per_persona;
124.      struct partecipante partecipanti[];
125.  };
126.  struct albergo {
127.      char indirizzo[MAX_LEN];
128.      char nome_località[MAX_LEN];
129.      double costo_per_notte;
130.      char referente[MAX_LEN];
131.      char nome_albergo[MAX_LEN];
132.      char fax[MAX_LEN];
133.      char email[MAX_LEN];
134.      char telefono[MAX_LEN];
135.      int capienza_max;
136.      int num_notti;
137.  };
138.  struct albergo_support {
139.      unsigned num_entries;
140.      struct albergo alberghi[];
141.  };
142.  struct localita {
143.      char nome_località[MAX_LEN];
144.      char stato[MAX_LEN];
145.  };
146.
147.  struct localita_support {
148.      unsigned num_entries;
149.      struct localita localita[];
150.  };
151.
```

users\cliente.json

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

users\login.json

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

users\segretario.json

```
{
  "host": "localhost",
  "username": "segretarioTA",
  "password": "segretario",
  "port": 3306,
  "database": "agenzia_viaggi"
}
```

utils\db.c

```
#include <mysql.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "db.h"
#include "../utils/db.h"
void set_binding_param(MYSQL_BIND *param, enum enum_field_types type, void *buffer, unsigned long len)
{
    memset(param, 0, sizeof(*param));

    param->buffer_type = type;
    param->buffer = buffer;
    param->buffer_length = len;
}
void time_to_mysql_time(char *str, MYSQL_TIME *time)
{
    memset(time, 0, sizeof(*time));
    sscanf(str, "%02d:%02d", &time->hour, &time->minute);
    time->time_type = MYSQL_TIMESTAMP_TIME;
}
void date_to_mysql_time(char *str, MYSQL_TIME *time)
{
    memset(time, 0, sizeof(*time));
    sscanf(str, "%4d-%2d-%2d", &time->year, &time->month, &time->day);
    time->time_type = MYSQL_TIMESTAMP_DATE;
}
void mysql_date_to_string(MYSQL_TIME *date, char *str)
{
    snprintf(str, DATE_LEN, "%4d-%02d-%02d", date->year, date->month, date->day);
}
void mysql_time_to_string(MYSQL_TIME *time, char *str)
```

```
32. {
33.     snprintf(str,TIME_LEN, "%02d:%02d", time->hour, time->minute);
34. }
35.
```

utils\db.h

```
1.  #include <stdbool.h>
2.  #include <mysql.h>
3.  extern void set_binding_param(MYSQL_BIND *param, enum enum_field_types type, void *buffer,
    unsigned long len);
4.  extern void time_to_mysql_time(char *str, MYSQL_TIME *time);
5.  extern void date_to_mysql_time(char *str, MYSQL_TIME *time);
6.  extern void mysql_date_to_string(MYSQL_TIME *date, char *str);
7.  extern void mysql_time_to_string(MYSQL_TIME *time, char *str);
```

view\clienteview.c

```
1.  #include<stdio.h>
2.  #include <stdlib.h>
3.  #include <string.h>
4.  #include "../defines.h"
5.  #include "../model/model.h"
6.  #include "clienteview.h"
7.  char cliente_get_action(){
8.      char options[6] = {'1','2', '3', '4', '5','q'};
9.      char op;
10.     printf("\033[2J\033[H");
11.     printf("*** WELCOME TO CLIENTE DASHBOARD ***\n\n");
12.     printf("1) Vedi i programmi di viaggio disponibili\n");
13.     printf("2) Vedi i viaggi disponibili per un certo programma\n");
14.     printf("3) Prenota un Viaggio\n");
15.     printf("4) Gestisci una Prenotazione\n");
16.     printf("5) Visualizza una Prenotazione\n");
17.     printf("q) Quit\n");
18.     op = multiChoice("Select an option", options, 6);
19.     return op;
20. }
21. char prenotazione_get_action(){
22.     char options[3] = {'1','2', 'b'};
23.     char op;
24.     printf("\033[2J\033[H");
25.     printf("*** Cosa vuoi fare ***\n\n");
26.     printf("1) Aggiungi Partecipante\n");
27.     printf("2) Disdici Prenotazione\n");
28.     printf("b) Back\n");
29.     op = multiChoice("Select an option", options, 3);
30.     return op;
31. }
32. static void get_partecipante_info(struct partecipante *partecipante){
33.     printf("\n**Inserisci per favore i dati del nuovo partecipante**\n");
34.     printf("\nInserisci il nome: ");
35.     getInput(MAX_LEN, partecipante->nome, false);
36.     printf("\nInserisci il cognome: ");
37.     getInput(MAX_LEN, partecipante->cognome, false);
38.     printf("\nInserisci il codice fiscale: ");
39.     getInput(CF_LEN, partecipante->cf, false);
40.     printf("\nInserisci la email: ");
41.     getInput(MAX_LEN, partecipante->email, false);
42.     printf("\nInserisci il la data di nascita [YYYY-MM-DD]: ");
43.     getInput(DATE_LEN, partecipante->data_nascita, false);
44.     printf("\nInserisci la nazionalità : ");
45.     getInput(MAX_LEN, partecipante->nazionalità, false);
46. }
47. static void get_prenotazione_info(struct prenotazione *prenotazione){
```

```
48.     char support[SIZE_INPUT_NUMBER];
49.     printf("\nInserisci l'id del programma di viaggio a cui il viaggio si riferisce: ");
50.     getInput(SIZE_INPUT_NUMBER, support, false);
51.     prenotazione->id_programma=atoi(support);
52.     printf("\nInserisci la data di partenza del viaggio [YYYY-MM-DD] : ");
53.     getInput(DATE_LEN, prenotazione->data_partenza, false);
54.     printf("\nInserisci il numero di persone per cui è  prevista  la prenotazione: ");
55.     getInput(SIZE_INPUT_NUMBER, support, false);
56.     prenotazione->num_partecipanti=atoi(support);
57. }
58. static void get_prenotazione_to_manage_info(struct prenotazione *prenotazione){
59.     char support[SIZE_INPUT_NUMBER];
60.     printf("\nInserisci l'id della prenotazione: ");
61.     getInput(SIZE_INPUT_NUMBER, support, false);
62.     prenotazione->id_prenotazione=atoi(support);
63.     printf("\nInserisci il codice che ti era stato rilasciato per gestire la prenotazione: ");
64.     getInput(MAX_LEN, prenotazione->codice_per_gestire, false);
65. }
66. static void get_id_info_for_tappe(struct programma_di_viaggio* programmaDiViaggio){
67.     char support[SIZE_INPUT_NUMBER];
68.     printf("\nInserisci l'id del prorgramma di viaggio che vuoi vedere in dettaglio: ");
69.     getInput(SIZE_INPUT_NUMBER, support, false);
70.     programmaDiViaggio->id_programma=atoi(support);
71. }
72. static void get_id_info_for_viaggi(struct programma_di_viaggio* programmaDiViaggio){
73.     char support[SIZE_INPUT_NUMBER];
74.     printf("\nInserisci l'id del programma di viaggio di cui vuoi vedere i viaggi in programma:
    ");
75.     getInput(SIZE_INPUT_NUMBER, support, false);
76.     programmaDiViaggio->id_programma=atoi(support);
77. }
78. static void print_viaggi_available(struct viaggio_support *viaggioSupport)
79. {
80.     printf("\n[** Edizioni Viaggi Disponibili **]\n\n");
81.
82.     for(size_t i = 0; i < viaggioSupport->num_entries; i++) {
83.         printf("[%ld]  Data Partenza: %s , Data Rientro: %s , Costo: %02.2f$ \n",
84.                i+1,
85.                viaggioSupport->viaggi[i].data_partenza,
86.                viaggioSupport->viaggi[i].data_rientro,
87.                viaggioSupport->viaggi[i].costo);
88.     }
89. }
90. static void print_programmi_available(struct programma_di_viaggio_support
    *programmaDiViaggioSupport)
91. {
92.     printf("[** Programmi di Viaggi Disponibili **]\n\n");
93.     for(size_t i = 0; i < programmaDiViaggioSupport->num_entries; i++) {
94.         printf("[ID: %d] Nome Programma: %s , Numero Giorni %d , Costo a partire
    da: %02.2f$ \n",
95.                programmaDiViaggioSupport->programmiDiViaggio[i].id_programma,
96.                programmaDiViaggioSupport->programmiDiViaggio[i].nome_programma,
97.                programmaDiViaggioSupport->programmiDiViaggio[i].num_giorni,
98.                programmaDiViaggioSupport->programmiDiViaggio[i].min_price
99.                );
100.      }
101.  }
102.  static void print_itinerario(struct tappa_support *tappaSupport)
103.  {
104.      printf("\n[** Itinerario **]\n\n");
105.
106.      for(size_t i = 0; i < tappaSupport->num_entries; i++) {
107.          printf("[Tappa %ld] %s \n ->Arrivo in giorno %d alle ore %s \n ->Partenza in giorno %d
    alle ore %s\n ->Trattamento %s\n",
108.                 i+1,
109.                 tappaSupport->tappe[i].nome_località,
110.                 tappaSupport->tappe[i].giorno_arrivo,
111.                 tappaSupport->tappe[i].ora_arrivo,
112.                 tappaSupport->tappe[i].giorno_partenza,
113.                 tappaSupport->tappe[i].ora_partenza,
```

```
                tappaSupport->tappe[i].trattamento
                );
        }

    }
    static void print_info_prenotazione(struct prenotazione_support *prenotazioneSupport)
    {
        printf("\n\n[**Informazioni Prenotazione**]");
        printf("\nRiferita a: %s , Data Partenza: %s\nAvvenuta in data: %s\nCosto per
    persona %02.2f$\n",
                prenotazioneSupport->nome_programma,
                prenotazioneSupport->data_partenza,
                prenotazioneSupport->data_prenotazione,
                prenotazioneSupport->prezzo_per_persona
        );
        if(!prenotazioneSupport->num_entries){
            printf("\nAncora nessun partecipante aggiunto\n");
        }else {
            for (size_t i = 0; i < prenotazioneSupport->num_entries; i++) {
                printf("[Partecipante %ld]: %s %s\n",
                       i + 1,
                       prenotazioneSupport->partecipanti[i].cognome,
                       prenotazioneSupport->partecipanti[i].nome
                );
            }
        }
    }
```

view\clienteview.h

```
#include<stdio.h>
#include <stdlib.h>
#include <string.h>
#include "../defines.h"
#include "../model/model.h"
char cliente_get_action();
char prenotazione_get_action();
static void get_partecipante_info(struct partecipante *partecipante);
static void get_prenotazione_info(struct prenotazione *prenotazione);
static void get_prenotazione_to_manage_info(struct prenotazione *prenotazione);
static void get_id_info_for_tappe(struct programma_di_viaggio* programmaDiViaggio);
static void get_id_info_for_viaggi(struct programma_di_viaggio* programmaDiViaggio);
static void print_viaggi_available(struct viaggio_support *viaggioSupport);
static void print_programmi_available(struct programma_di_viaggio_support
    *programmaDiViaggioSupport);
static void print_itinerario(struct tappa_support *tappaSupport);
static void print_info_prenotazione(struct prenotazione_support *prenotazioneSupport);
```

view\segretarioview.c

```
#include<stdio.h>
#include <stdlib.h>
#include <string.h>
#include "../defines.h"
#include "../model/model.h"
#include "segretarioview.h"
char segretario_get_action(){
    char options[8] = {'1','2', '3', '4', '5','6','7','q'};
    char op;
    printf("\033[2J\033[H");
    printf("\n[*** WELCOME TO SEGRETERIA DASHBOARD ***]\n\n");
    printf("1) Gestisci Programmi di Viaggio\n");
    printf("2) Gestisci Viaggi\n");
    printf("3) Gestisci Località\n");
    printf("4) Gestisci Guide\n");
```

```
    printf("5) Gestisci Autobus Privati\n");
    printf("6) Gestisci Alberghi\n");
    printf("7) Genera Report di un viaggio\n");
    printf("q) Quit\n");
    op = multiChoice("Select an option", options, 8);
    return op;
}
char nested_get_action_1(){
    char options[4] = {'1','2', '3', 'b'};
    char op;
    printf("\033[2J\033[H");
    printf("******\n\n");
    printf("1) Lista Programmi di Viaggio\n");
    printf("2) Inserisci nuovo Programma di Viaggio\n");
    printf("3) Aggiungi tappe a un programma di viaggio\n");
    printf("b) Back to previous menu\n");
    op = multiChoice("Select an option", options, 4);
    return op;
}
char nested_get_action_2(){
    char options[6] = {'1','2', '3','4','5','b'};
    char op;
    printf("\033[2J\033[H");
    printf("******\n\n");
    printf("1) Lista Viaggi\n");
    printf("2) Inserisci nuovo Viaggio\n");
    printf("3) Assegna una guida a un Viaggio\n");
    printf("4) Assegna un autobus privato a un Viaggio\n");
    printf("5) Assegna un albergo a un Viaggio\n");
    printf("b) Back to previous menu\n");
    op = multiChoice("Select an option", options, 6);
    return op;
}
char nested_get_action_3(){
    char options[3] = {'1','2','b'};
    char op;
    printf("\033[2J\033[H");
    printf("******\n\n");
    printf("1) Lista Guide\n");
    printf("2) Aggiungi Guida\n");
    printf("b) Back to previous menu\n");
    op = multiChoice("Select an option", options, 3);
    return op;
}
char nested_get_action_4(){
    char options[3] = {'1','2','b'};
    char op;
    printf("\033[2J\033[H");
    printf("******\n\n");
    printf("1) Lista Autobus Privati\n");
    printf("2) Aggiungi Autobus Privato\n");
    printf("b) Back to previous menu\n");
    op = multiChoice("Select an option", options, 3);
    return op;
}
char nested_get_action_5(){
    char options[4] = {'1','2', '3','b'};
    char op;
    printf("\033[2J\033[H");
    printf("******\n\n");
    printf("1) Lista Alberghi\n");
    printf("2) Aggiungi Albergo\n");
    printf("3) Rimuovi Albergo\n");
    printf("b) Back to previous menu\n");
    op = multiChoice("Select an option", options, 4);
    return op;
}
char nested_get_action_6(){
    char options[3] = {'1','2','b'};
    char op;
```

```
86.     printf("\033[2J\033[H");
87.     printf("******\n\n");
88.     printf("1) Lista Località\n");
89.     printf("2) Aggiungi nuova Località\n");
90.     printf("b) Back to previous menu\n");
91.     op = multiChoice("Select an option", options, 3);
92.     return op;
93. }
94. static void print_viaggi_confermati(struct viaggio_support *viaggioSupport,struct
    viaggio*viaggio)
95. {
96.     printf("\n[Viaggi nello stato '%s']\n\n",viaggio->stato_viaggio);
97.    if(!viaggioSupport->num_entries){
98.        printf("La lista è vuota!\n");
99.    }else {
100.         for (size_t i = 0; i < viaggioSupport->num_entries; i++) {
101.             printf("[ID Programma %d] Data Partenza: %s , Data Rientro: %s , Costo: %02.2f$",
102.                    viaggioSupport->viaggi[i].id_programma,
103.                    viaggioSupport->viaggi[i].data_partenza,
104.                    viaggioSupport->viaggi[i].data_rientro,
105.                    viaggioSupport->viaggi[i].costo
106.             );
107.             if (strcmp(viaggio->stato_viaggio, "prenotabile") != 0) {
108.                 printf(", Numero Partecipanti: %d", viaggioSupport-
    >viaggi[i].num_partecipanti_confermati);
109.             }
110.             puts("");
111.         }
112.     }
113. }
114.
115. static void print_guide_assign(struct guida_support *guidaSupport)
116. {
117.     printf("\n[** Assegnamenti Guide **]\n\n");
118.
119.     for(size_t i = 0; i < guidaSupport->num_entries; i++) {
120.         printf("[%s] Data Partenza: %s , Data Rientro: %s\n",
121.                guidaSupport->guide[i].cf,
122.                guidaSupport->guide[i].dataP,
123.                guidaSupport->guide[i].dataR
124.         );
125.     }
126. }
127. static void print_itinerario(struct tappa_support *tappaSupport)
128. {
129.     if(!tappaSupport->num_entries){
130.         printf("\nItinerario Vuoto: ancora nessuna tappa aggiunta!\n");
131.     }else {
132.         printf("\n[** Itinerario **]\n\n");
133.         for (size_t i = 0; i < tappaSupport->num_entries; i++) {
134.             printf("[Tappa %ld] %s \n ->Arrivo in giorno %d alle ore %s \n ->Partenza in
    giorno %d alle ore %s\n ->Trattamento %s\n",
135.                    i + 1,
136.                    tappaSupport->tappe[i].nome_località,
137.                    tappaSupport->tappe[i].giorno_arrivo,
138.                    tappaSupport->tappe[i].ora_arrivo,
139.                    tappaSupport->tappe[i].giorno_partenza,
140.                    tappaSupport->tappe[i].ora_partenza,
141.                    tappaSupport->tappe[i].trattamento
142.             );
143.         }
144.     }
145.
146. }
147. static void print_autobus_assign(struct autobus_support *autobusSupport)
148. {
149.     printf("\n[** Assegnamenti Autobus **]\n\n");
150.
151.     for(size_t i = 0; i < autobusSupport->num_entries; i++) {
152.         printf("[%s] Data Partenza: %s , Data Rientro: %s\n",
```

```
153.                 autobusSupport->autobusPrivato[i].targa,
154.                 autobusSupport->autobusPrivato[i].dataP,
155.                 autobusSupport->autobusPrivato[i].dataR
156.             );
157.         }
158.     }
159.
160.     static void print_info_guide(struct guida_support *guidaSupport)
161.     {
162.         printf("\n[** Informazioni Guide **]\n\n");
163.
164.         for(size_t i = 0; i < guidaSupport->num_entries; i++) {
165.             printf("[CF: %s] Nome: %s , Cognome: %s , Contatto: %s , Data Nascita: %s\n",
166.                 guidaSupport->guide[i].cf,
167.                 guidaSupport->guide[i].nome,
168.                 guidaSupport->guide[i].cognome,
169.                 guidaSupport->guide[i].email,
170.                 guidaSupport->guide[i].data_nascita
171.             );
172.         }
173.     }
174.     static void print_info_localita(struct localita_support *localitaSupport)
175.     {
176.         printf("\n[** Informazioni Località **]\n\n");
177.
178.         for(size_t i = 0; i < localitaSupport->num_entries; i++) {
179.             printf("%ld) Nome: %s , Stato: %s \n",
180.                 i+1,
181.                 localitaSupport->localita[i].nome_località,
182.                 localitaSupport->localita[i].stato
183.             );
184.         }
185.     }
186.
187.     static void print_info_alberghi(struct albergo_support *albergoSupport)
188.     {
189.         printf("\n[** Informazioni Alberghi **]\n\n");
190.
191.         for(size_t i = 0; i < albergoSupport->num_entries; i++) {
192.             printf("[Località: %s , Indirizzo: %s ] Nome: %s , Costo: %.2f $, Capienza: %d
    \n[Contatti] Referente: %s, Fax : %s , E-mail : %s, Telefono: %s \n\n",
193.                 albergoSupport->alberghi[i].nome_località,
194.                 albergoSupport->alberghi[i].indirizzo,
195.                 albergoSupport->alberghi[i].nome_albergo,
196.                 albergoSupport->alberghi[i].costo_per_notte,
197.                 albergoSupport->alberghi[i].capienza_max,
198.                 albergoSupport->alberghi[i].referente,
199.                 albergoSupport->alberghi[i].fax,
200.                 albergoSupport->alberghi[i].email,
201.                 albergoSupport->alberghi[i].telefono
202.             );
203.         }
204.     }
205.
206.     static void print_report(struct report_support *reportSupport)
207.     {
208.         printf("[** Report **]\n\n");
209.         printf("Numero Partecipanti: %d\nCosto Per Persona: %.2f $\nCosto Totale Autobus: %.2f
    $\nCosto Totale Alberghi: %.2f $\nProfitto: %.2f $\n",
210.             reportSupport->num_partecipanti,
211.             reportSupport->costo_per_persona,
212.             reportSupport->costo_tot_autobus,
213.             reportSupport->costo_tot_alberghi,
214.             reportSupport->profitto);
215.         printf("\n[** Dati sui partecipanti **]\n\n");
216.         for(size_t i = 0; i < reportSupport->num_entries; i++) {
217.             printf("[Partecipante %ld]  Nome: %s , Cognome: %s , Email: %s , Nazionalità: %s, Data
    Nascita: %s\n",
218.                 i+1,
219.                 reportSupport->partecipanti[i].nome,
```

```
                reportSupport->partecipanti[i].cognome,
                reportSupport->partecipanti[i].email,
                reportSupport->partecipanti[i].nazionalità,
                reportSupport->partecipanti[i].data_nascita
        );
    }
}

static void print_info_autobus(struct autobus_support *autobusSupport)
{
    printf("\n[** Informazioni Autobus Privati **]\n\n");

    for(size_t i = 0; i < autobusSupport->num_entries; i++) {
        printf("[Targa: %s]  Nome Deposito: %s , Costo Giornaliero: %.2f , Capienza: %d\n",
                autobusSupport->autobusPrivato[i].targa,
                autobusSupport->autobusPrivato[i].nome_deposito,
                autobusSupport->autobusPrivato[i].costo_giornaliero,
                autobusSupport->autobusPrivato[i].capienza_max
        );
    }
}

static void get_programma_di_viaggio_info(struct programma_di_viaggio *programmaDiViaggio){
    char giorni[SIZE_INPUT_NUMBER];
    printf("\nInserisci il nome del programma di viaggio: ");
    getInput(MAX_LEN, programmaDiViaggio->nome_programma, false);
    printf("\nInserisci il numero di giorni del programma di viaggio: ");
    getInput(SIZE_INPUT_NUMBER, giorni, false);
    programmaDiViaggio->num_giorni = atoi(giorni);
}
static void get_autobus_privato_info(struct autobus_privato *autobusPrivato){
    char support[sizeof(int)];
    printf("\nInserisci la targa dell'autobus privato: ");
    getInput(TARGA_LEN, autobusPrivato->targa, false);
    printf("\nInserisci il nome del suo deposito: ");
    getInput(MAX_LEN, autobusPrivato->nome_deposito, false);
    printf("\nInserisci la capienza massima: ");
    getInput(SIZE_INPUT_NUMBER, support, false);
    autobusPrivato->capienza_max= atoi(support);
    printf("\nInserisci il costo giornaliero: ");
    getInput(SIZE_INPUT_NUMBER, support, false);
    autobusPrivato->costo_giornaliero= strtof(support,NULL);
}
static void get_viaggio_info(struct viaggio *viaggio){
    char support[SIZE_INPUT_NUMBER];
    printf("\nInserisci l'id del programma di viaggio a cui si riferisce: ");
    getInput(SIZE_INPUT_NUMBER, support, false);
    viaggio->id_programma=atoi(support);
    printf("\nInserisci la data di partenza del viaggio [YYYY-MM-DD] : ");
    getInput(DATE_LEN, viaggio->data_partenza, false);
    printf("\nInserisci la data di rientro del viaggio [YYYY-MM-DD] : ");
    getInput(DATE_LEN, viaggio->data_rientro, false);
    printf("\nInserisci il costo del viaggio: ");
    getInput(SIZE_INPUT_NUMBER, support, false);
    viaggio->costo= atoi(support);
}
static void get_viaggio_info_for_report(struct viaggio *viaggio){
    char support[SIZE_INPUT_NUMBER];
    printf("\nInserisci l'id del programma di viaggio a cui si riferisce: ");
    getInput(SIZE_INPUT_NUMBER, support, false);
    viaggio->id_programma=atoi(support);
    printf("\nInserisci la data di partenza del viaggio [YYYY-MM-DD] : ");
    getInput(DATE_LEN, viaggio->data_partenza, false);
}
static void get_guida_info(struct guida *guida){
    printf("\nInserisci il nome della guida: ");
    getInput(MAX_LEN, guida->nome, false);
    printf("\nInserisci il cognome della guida: ");
    getInput(MAX_LEN, guida->cognome, false);
    printf("\nInserisci il codice fiscale della guida: ");
```

```
290.      getInput(CF_LEN, guida->cf, false);
291.      printf("\nInserisci il email della guida: ");
292.      getInput(MAX_LEN, guida->email, false);
293.      printf("\nInserisci il la data di nascita della guida [YYYY-MM-DD]: ");
294.      getInput(DATE_LEN, guida->data_nascita, false);
295.  }
296.  static void print_programmi_available_segretari(struct programma_di_viaggio_support
    *programmaDiViaggioSupport)
297.  {
298.      printf("\n\n[** Programmi Di Viaggio Disponibili **]\n\n");
299.      for(size_t i = 0; i < programmaDiViaggioSupport->num_entries; i++) {
300.          printf("[ID %d] Nome Programma: %s , Numero Giorni %d \n",
301.                 programmaDiViaggioSupport->programmiDiViaggio[i].id_programma,
302.                 programmaDiViaggioSupport->programmiDiViaggio[i].nome_programma,
303.                 programmaDiViaggioSupport->programmiDiViaggio[i].num_giorni
304.          );
305.      }
306.  }
307.  static void get_info_for_assign_guida(struct guida *guida,struct viaggio *viaggio){
308.      char support[SIZE_INPUT_NUMBER];
309.      printf("\nInserisci il codice fiscale della guida da assegnare: ");
310.      getInput(CF_LEN, guida->cf, false);
311.      printf("\nInserisci l'id del programma di viaggio a cui il viaggio si riferisce: ");
312.      getInput(SIZE_INPUT_NUMBER, support, false);
313.      viaggio->id_programma=atoi(support);
314.      printf("\nInserisci la data di partenza del viaggio [YYYY-MM-DD] : ");
315.      getInput(DATE_LEN, viaggio->data_partenza, false);
316.  }
317.  static void get_info_for_assign_autobus(struct autobus_privato *autobusPrivato,struct viaggio
    *viaggio){
318.      char support[sizeof(int)];
319.      printf("\nInserisci la targa dell'autobus da assegnare: ");
320.      getInput(TARGA_LEN, autobusPrivato->targa, false);
321.      printf("\nInserisci l'id del programma di viaggio a cui il viaggio si riferisce: ");
322.      getInput(SIZE_INPUT_NUMBER, support, false);
323.      viaggio->id_programma=atoi(support);
324.      printf("\nInserisci la data di partenza del viaggio [YYYY-MM-DD] : ");
325.      getInput(DATE_LEN, viaggio->data_partenza, false);
326.  }
327.  static void get_info_for_assign_albergo(struct albergo *albergo,struct viaggio *viaggio){
328.      char support[SIZE_INPUT_NUMBER];
329.      printf("\nInserisci il numero di notti per cui è previsto il pernottamento: ");
330.      getInput(SIZE_INPUT_NUMBER, support, false);
331.      albergo->num_notti=atoi(support);
332.      printf("\nInserisci la località dell'albergo da assegnare: ");
333.      getInput(MAX_LEN, albergo->nome_località, false);
334.      printf("\nInserisci l'indirizzo dell'albergo da assegnare: ");
335.      getInput(MAX_LEN, albergo->indirizzo, false);
336.      printf("\nInserisci l'id del programma di viaggio a cui il viaggio si riferisce: ");
337.      getInput(SIZE_INPUT_NUMBER, support, false);
338.      viaggio->id_programma=atoi(support);
339.      printf("\nInserisci la data di partenza del viaggio [YYYY-MM-DD] : ");
340.      getInput(DATE_LEN, viaggio->data_partenza, false);
341.  }
342.  static void get_idprogramma_info(struct  programma_di_viaggio*programmaDiViaggio){
343.      char input[SIZE_INPUT_NUMBER];
344.      printf("\nInserisci l'ID del programma di viaggio a cui vuoi aggiungere una tappa: ");
345.      getInput(SIZE_INPUT_NUMBER, input, false);
346.      programmaDiViaggio->id_programma=atoi(input);
347.  }
348.  static int ask_for_another(){
349.      int ret=0;
350.      char options[2] = {'Y','N'};
351.      int r = multiChoice("\nVuoi inserire un altra tappa?\n"
352.                          "Inserire 'Y' per continuare oppure 'N' per concludere ", options, 2);
353.
354.      switch(r) {
355.          case 'Y':
356.              ret=1;
357.              break;
```

```
358.          case 'N':
359.              break;
360.          default:
361.              fprintf(stderr, "Invalid condition at %s:%d\n", __FILE__, __LINE__);
362.              abort();
363.      }
364.      return ret;
365.
366.  }
367.  static char ask_for_stato_richiesto(){
368.      char ret;
369.      char options[4]={'1','2','3','4'};
370.      printf("\nSeleziona lo stato dei viaggi da visualizzare: \n");
371.      printf("1) Viaggi in programma\n");
372.      printf("2) Viaggi attivi\n");
373.      printf("3) Viaggi terminati\n");
374.      printf("4) Viaggi prenotabili\n");
375.      ret = multiChoice("\nSeziona una possibilità", options, 4);
376.      return ret;
377.
378.  }
379.  static void get_tappa_info(struct tappa *tappa){
380.      char options[5] = {'1','2', '3','4','5'};
381.      char r;
382.      char giornoarrivo[10];
383.      char giornopartenza[10];
384.      // Get the required info
385.      printf("\nInsert località  nuova della tappa: ");
386.      getInput(MAX_LEN, tappa->nome_località, false);
387.      printf("\nInsert giorno di arrivo: ");
388.      getInput(SIZE_INPUT_NUMBER, giornoarrivo, false);
389.      tappa->giorno_arrivo= atoi(giornoarrivo);
390.
391.      printf("\nInsert ora arrivo [HH:MM] : ");
392.      getInput(TIME_LEN, tappa->ora_arrivo, false);
393.
394.      printf("\nInsert giorno di partenza: ");
395.      getInput(SIZE_INPUT_NUMBER, giornopartenza, false);
396.      tappa->giorno_partenza= atoi(giornopartenza);
397.
398.      printf("\nInsert ora partenza [HH:MM] : ");
399.      getInput(TIME_LEN, tappa->ora_partenza, false);
400.      printf("\nAssign a possible trattamento: \n");
401.      printf("\t1) FB\n");
402.      printf("\t2) HB\n");
403.      printf("\t3) BB\n");
404.      printf("\t4) OB\n");
405.      printf("\t5) NB\n");
406.      r = multiChoice("\nSelect trattamento", options, 5);
407.
408.      // Convert role into enum value
409.      switch(r) {
410.          case '1':
411.              strcpy(tappa->trattamento, "FB");
412.              break;
413.          case '2':
414.              strcpy(tappa->trattamento, "HB");
415.              break;
416.          case '3':
417.              strcpy(tappa->trattamento, "BB");
418.              break;
419.          case '4':
420.              strcpy(tappa->trattamento, "OB");
421.              break;
422.          case '5':
423.              strcpy(tappa->trattamento, "NB");
424.              break;
425.          default:
426.              fprintf(stderr, "Invalid condition at %s:%d\n", __FILE__, __LINE__);
427.              abort();
```

```
428.      }
429.  }
430.
431.  static void get_albergo_info(struct albergo *albergo){
432.      char capienza[MAX_LEN];
433.      char costo_pn_char[MAX_LEN];
434.      printf("\nInserisci indirizzo: ");
435.      getInput(MAX_LEN, albergo->indirizzo, false);
436.      printf("\nInserisci località indirizzo: ");
437.      getInput(MAX_LEN, albergo->nome_località, false);
438.
439.      redo:
440.      printf("\nInserisci il costo per notte per persona: "); //deve essere un float
441.      getInput(SIZE_INPUT_NUMBER, costo_pn_char, false);
442.      if(!strtof(costo_pn_char,NULL)){
443.          printf("\nInput non valido, per favore ripeti");
444.          goto redo;
445.      }else{
446.          albergo->costo_per_notte= strtof(costo_pn_char,NULL);
447.      }
448.
449.      printf("\nInserisci il nome di un referente: ");
450.      getInput(MAX_LEN, albergo->referente, false);
451.      printf("\nInserisci il nome dell'albergo: ");
452.      getInput(MAX_LEN, albergo->nome_albergo, false);
453.      printf("\nInserisci il fax: ");
454.      getInput(MAX_LEN, albergo->fax, false);
455.      printf("\nInserisci l'email: ");
456.      getInput(MAX_LEN, albergo->email, false);
457.      printf("\nInserisci un telefono: ");
458.      getInput(MAX_LEN, albergo->telefono, false);
459.
460.      redo2:
461.      printf("\nInserisci capienza massima di persone: "); //deve essere un intero!
462.      getInput(SIZE_INPUT_NUMBER, capienza, false);
463.      if(!atoi(capienza)){
464.          printf("\nInput non valido, per favore ripeti");
465.          goto redo2;
466.      }else{
467.          albergo->capienza_max =atoi(capienza);
468.      }
469.
470.  }
471.  static void get_localita_info(struct localita *localita){
472.      printf("\nNome località: ");
473.      getInput(MAX_LEN, localita->nome_località, false);
474.      printf("Stato località: ");
475.      getInput(MAX_LEN, localita->stato, false);
476.  }
```

view\segretarioview.h

```
1.  #include<stdio.h>
2.  #include <stdio.h>
3.  #include <string.h>
4.  #include "../defines.h"
5.  #include "../model/model.h"
6.  char segretario_get_action();
7.  static void get_programma_di_viaggio_info(struct programma_di_viaggio *programmaDiViaggio);
8.  static void get_autobus_privato_info(struct autobus_privato *autobusPrivato);
9.  static void get_viaggio_info(struct viaggio *viaggio);
10. static void get_guida_info(struct guida *guida);
11. static void get_albergo_info(struct albergo *albergo);
12. static void get_localita_info(struct localita *localita);
```

defines.h

```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 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 run_as_cliente(MYSQL *conn);
extern void run_as_professor(MYSQL *conn);
extern void run_as_segretario(MYSQL *conn);

```

inout.c

```c
#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"

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;

  sigaction_t sa, savealrm, saveint, savehup, savequit, saveterm;
  sigaction_t savetstp, savettin, savettou;
  struct termios term, oterm;

  if(hide) {
            (void) fflush(stdout);

            sigemptyset(&sa.sa_mask);
            sa.sa_flags = SA_INTERRUPT;
            sa.sa_handler = handler;
```

```
33.             (void) sigaction(SIGALRM, &sa, &savealrm);
34.             (void) sigaction(SIGINT, &sa, &saveint);
35.             (void) sigaction(SIGHUP, &sa, &savehup);
36.             (void) sigaction(SIGQUIT, &sa, &savequit);
37.             (void) sigaction(SIGTERM, &sa, &saveterm);
38.             (void) sigaction(SIGTSTP, &sa, &savetstp);
39.             (void) sigaction(SIGTTIN, &sa, &savettin);
40.             (void) sigaction(SIGTTOU, &sa, &savettou);
41.
42.             if (tcgetattr(fileno(stdin), &oterm) == 0) {
43.                     (void) memcpy(&term, &oterm, sizeof(struct termios));
44.                     term.c_lflag &= ~(ECHO|ECHONL);
45.                     (void) tcsetattr(fileno(stdin), TCSAFLUSH, &term);
46.             } else {
47.                     (void) memset(&term, 0, sizeof(struct termios));
48.                     (void) memset(&oterm, 0, sizeof(struct termios));
49.             }
50.     }
51.
52.     for(i = 0; i < lung; i++) {
53.             (void) fread(&c, sizeof(char), 1, stdin);
54.             if(c == '\n') {
55.                     stringa[i] = '\0';
56.                     break;
57.             } else
58.                     stringa[i] = c;
59.
60.             if(hide) {
61.                     if(c == '\b')
62.                             (void) write(fileno(stdout), &c, sizeof(char));
63.                     else
64.                             (void) write(fileno(stdout), "*", sizeof(char));
65.             }
66.     }
67.
68.     if(i == lung - 1)
69.             stringa[i] = '\0';
70.
71.     if(strlen(stringa) >= lung) {
72.             do {
73.                     c = getchar();
74.             } while (c != '\n');
75.     }
76.
77.     if(hide) {
78.             (void) write(fileno(stdout), "\n", 1);
79.
80.             (void) tcsetattr(fileno(stdin), TCSAFLUSH, &oterm);
81.
82.             (void) sigaction(SIGALRM, &savealrm, NULL);
83.             (void) sigaction(SIGINT, &saveint, NULL);
84.             (void) sigaction(SIGHUP, &savehup, NULL);
85.             (void) sigaction(SIGQUIT, &savequit, NULL);
86.             (void) sigaction(SIGTERM, &saveterm, NULL);
87.             (void) sigaction(SIGTSTP, &savetstp, NULL);
88.             (void) sigaction(SIGTTIN, &savettin, NULL);
89.             (void) sigaction(SIGTTOU, &savettou, NULL);
90.
91.             if(signo)
92.                     (void) raise(signo);
93.     }
94.
95.     return stringa;
96. }
97.
98. static void handler(int s) {
99.     signo = s;
100.    }
101.
102.    char multiChoice(char *domanda, char choices[], int num)
```

```
103.  {
104.
105.          char *possib = malloc(2 * num * sizeof(char));
106.          int i, j = 0;
107.          for(i = 0; i < num; i++) {
108.                  possib[j++] = choices[i];
109.                  possib[j++] = '/';
110.          }
111.          possib[j-1] = '\0';
112.
113.          while(true) {
114.                  printf("%s [%s]: ", domanda, possib);
115.
116.                  char c;
117.                  getInput(1, &c, false);
118.
119.                  for(i = 0; i < num; i++) {
120.                          if(c == choices[i])
121.                                  return c;
122.                  }
123.          }
124.  }
125.
```

parse.c

```
1.  #include <stddef.h>
2.  #include <stdio.h>
3.  #include <stdlib.h>
4.  #include <string.h>
5.
6.  #include "defines.h"
7.
8.  #define BUFF_SIZE 4096
9.
10. // The final config struct will point into this
11. static char config[BUFF_SIZE];
12.
13. /**
14.  * JSON type identifier. Basic types are:
15.  *          o Object
16.  *          o Array
17.  *          o String
18.  *          o Other primitive: number, boolean (true/false) or null
19.  */
20. typedef enum {
21.   JSMN_UNDEFINED = 0,
22.   JSMN_OBJECT = 1,
23.   JSMN_ARRAY = 2,
24.   JSMN_STRING = 3,
25.   JSMN_PRIMITIVE = 4
26. } jsmntype_t;
27.
28. enum jsmnerr {
29.   /* Not enough tokens were provided */
30.   JSMN_ERROR_NOMEM = -1,
31.   /* Invalid character inside JSON string */
32.   JSMN_ERROR_INVAL = -2,
33.   /* The string is not a full JSON packet, more bytes expected */
34.   JSMN_ERROR_PART = -3
35. };
36.
37. /**
38.  * JSON token description.
39.  * type              type (object, array, string etc.)
40.  * start    start position in JSON data string
41.  * end               end position in JSON data string
```

```
42.  */
43. typedef struct {
44.    jsmntype_t type;
45.    int start;
46.    int end;
47.    int size;
48. #ifdef JSMN_PARENT_LINKS
49.    int parent;
50. #endif
51. } jsmntok_t;
52.
53. /**
54.  * JSON parser. Contains an array of token blocks available. Also stores
55.  * the string being parsed now and current position in that string
56.  */
57. typedef struct {
58.    unsigned int pos; /* offset in the JSON string */
59.    unsigned int toknext; /* next token to allocate */
60.    int toksuper; /* superior token node, e.g parent object or array */
61. } jsmn_parser;
62.
63. /**
64.  * Allocates a fresh unused token from the token pool.
65.  */
66. static jsmntok_t *jsmn_alloc_token(jsmn_parser *parser, jsmntok_t *tokens, size_t num_tokens) {
67.    jsmntok_t *tok;
68.    if (parser->toknext >= num_tokens) {
69.            return NULL;
70.    }
71.    tok = &tokens[parser->toknext++];
72.    tok->start = tok->end = -1;
73.    tok->size = 0;
74. #ifdef JSMN_PARENT_LINKS
75.    tok->parent = -1;
76. #endif
77.    return tok;
78. }
79.
80. /**
81.  * Fills token type and boundaries.
82.  */
83. static void jsmn_fill_token(jsmntok_t *token, jsmntype_t type,
84.                              int start, int end) {
85.    token->type = type;
86.    token->start = start;
87.    token->end = end;
88.    token->size = 0;
89. }
90.
91. /**
92.  * Fills next available token with JSON primitive.
93.  */
94. static int jsmn_parse_primitive(jsmn_parser *parser, const char *js,
95.            size_t len, jsmntok_t *tokens, size_t num_tokens) {
96.    jsmntok_t *token;
97.    int start;
98.
99.    start = parser->pos;
100.
101.           for (; parser->pos < len && js[parser->pos] != '\0'; parser->pos++) {
102.                   switch (js[parser->pos]) {
103.   #ifndef JSMN_STRICT
104.                           /* In strict mode primitive must be followed by "," or "}" or "]"
    */
105.                           case ':':
106.   #endif
107.                           case '\t' : case '\r' : case '\n' : case ' ' :
108.                           case ','  : case ']'  : case '}' :
109.                                   goto found;
110.                   }
```

```
111.                    if (js[parser->pos] < 32 || js[parser->pos] >= 127) {
112.                            parser->pos = start;
113.                            return JSMN_ERROR_INVAL;
114.                    }
115.            }
116.    #ifdef JSMN_STRICT
117.            /* In strict mode primitive must be followed by a comma/object/array */
118.            parser->pos = start;
119.            return JSMN_ERROR_PART;
120.    #endif
121.
122.    found:
123.            if (tokens == NULL) {
124.                    parser->pos--;
125.                    return 0;
126.            }
127.            token = jsmn_alloc_token(parser, tokens, num_tokens);
128.            if (token == NULL) {
129.                    parser->pos = start;
130.                    return JSMN_ERROR_NOMEM;
131.            }
132.            jsmn_fill_token(token, JSMN_PRIMITIVE, start, parser->pos);
133.    #ifdef JSMN_PARENT_LINKS
134.            token->parent = parser->toksuper;
135.    #endif
136.            parser->pos--;
137.            return 0;
138.    }
139.
140.    /**
141.     * Fills next token with JSON string.
142.     */
143.    static int jsmn_parse_string(jsmn_parser *parser, const char *js,
144.                    size_t len, jsmntok_t *tokens, size_t num_tokens) {
145.            jsmntok_t *token;
146.
147.            int start = parser->pos;
148.
149.            parser->pos++;
150.
151.            /* Skip starting quote */
152.            for (; parser->pos < len && js[parser->pos] != '\0'; parser->pos++) {
153.                    char c = js[parser->pos];
154.
155.                    /* Quote: end of string */
156.                    if (c == '\"') {
157.                            if (tokens == NULL) {
158.                                    return 0;
159.                            }
160.                            token = jsmn_alloc_token(parser, tokens, num_tokens);
161.                            if (token == NULL) {
162.                                    parser->pos = start;
163.                                    return JSMN_ERROR_NOMEM;
164.                            }
165.                            jsmn_fill_token(token, JSMN_STRING, start+1, parser->pos);
166.    #ifdef JSMN_PARENT_LINKS
167.                            token->parent = parser->toksuper;
168.    #endif
169.                            return 0;
170.                    }
171.
172.                    /* Backslash: Quoted symbol expected */
173.                    if (c == '\\' && parser->pos + 1 < len) {
174.                            int i;
175.                            parser->pos++;
176.                            switch (js[parser->pos]) {
177.                                    /* Allowed escaped symbols */
178.                                    case '\"': case '/' : case '\\' : case 'b' :
179.                                    case 'f' : case 'r' : case 'n'  : case 't' :
180.                                            break;
```

```
181.                                    /* Allows escaped symbol \uXXXX */
182.                                    case 'u':
183.                                            parser->pos++;
184.                                            for(i = 0; i < 4 && parser->pos < len &&
    js[parser->pos] != '\0'; i++) {
185.                                                    /* If it isn't a hex character we
    have an error */
186.                                                    if(!((js[parser->pos] >= 48 &&
    js[parser->pos] <= 57) || /* 0-9 */
187.
      (js[parser->pos] >= 65 && js[parser->pos] <= 70) || /* A-F */
188.
      (js[parser->pos] >= 97 && js[parser->pos] <= 102))) { /* a-f */
189.                                                            parser->pos = start;
190.                                                            return JSMN_ERROR_INVAL;
191.                                                    }
192.                                                    parser->pos++;
193.                                            }
194.                                            parser->pos--;
195.                                            break;
196.                                    /* Unexpected symbol */
197.                                    default:
198.                                            parser->pos = start;
199.                                            return JSMN_ERROR_INVAL;
200.                                }
201.                        }
202.                }
203.                parser->pos = start;
204.                return JSMN_ERROR_PART;
205.    }
206.
207.    /**
208.     * Parse JSON string and fill tokens.
209.     */
210.    static int jsmn_parse(jsmn_parser *parser, const char *js, size_t len, jsmntok_t *tokens,
    unsigned int num_tokens) {
211.                int r;
212.                int i;
213.                jsmntok_t *token;
214.                int count = parser->toknext;
215.
216.                for (; parser->pos < len && js[parser->pos] != '\0'; parser->pos++) {
217.                        char c;
218.                        jsmntype_t type;
219.
220.                        c = js[parser->pos];
221.                        switch (c) {
222.                                case '{': case '[':
223.                                        count++;
224.                                        if (tokens == NULL) {
225.                                                break;
226.                                        }
227.                                        token = jsmn_alloc_token(parser, tokens, num_tokens);
228.                                        if (token == NULL)
229.                                                return JSMN_ERROR_NOMEM;
230.                                        if (parser->toksuper != -1) {
231.                                                tokens[parser->toksuper].size++;
232.    #ifdef JSMN_PARENT_LINKS
233.                                                token->parent = parser->toksuper;
234.    #endif
235.                                        }
236.                                        token->type = (c == '{' ? JSMN_OBJECT : JSMN_ARRAY);
237.                                        token->start = parser->pos;
238.                                        parser->toksuper = parser->toknext - 1;
239.                                        break;
240.                                case '}': case ']':
241.                                        if (tokens == NULL)
242.                                                break;
243.                                        type = (c == '}' ? JSMN_OBJECT : JSMN_ARRAY);
244.    #ifdef JSMN_PARENT_LINKS
```

```
245.                                    if (parser->toknext < 1) {
246.                                            return JSMN_ERROR_INVAL;
247.                                    }
248.                                    token = &tokens[parser->toknext - 1];
249.                                    for (;;) {
250.                                            if (token->start != -1 && token->end == -1) {
251.                                                    if (token->type != type) {
252.                                                            return JSMN_ERROR_INVAL;
253.                                                    }
254.                                                    token->end = parser->pos + 1;
255.                                                    parser->toksuper = token->parent;
256.                                                    break;
257.                                            }
258.                                            if (token->parent == -1) {
259.                                                    if(token->type != type || parser-
    >toksuper == -1) {
260.                                                            return JSMN_ERROR_INVAL;
261.                                                    }
262.                                                    break;
263.                                            }
264.                                            token = &tokens[token->parent];
265.                                    }
266.  #else
267.                                    for (i = parser->toknext - 1; i >= 0; i--) {
268.                                            token = &tokens[i];
269.                                            if (token->start != -1 && token->end == -1) {
270.                                                    if (token->type != type) {
271.                                                            return JSMN_ERROR_INVAL;
272.                                                    }
273.                                                    parser->toksuper = -1;
274.                                                    token->end = parser->pos + 1;
275.                                                    break;
276.                                            }
277.                                    }
278.                                    /* Error if unmatched closing bracket */
279.                                    if (i == -1) return JSMN_ERROR_INVAL;
280.                                    for (; i >= 0; i--) {
281.                                            token = &tokens[i];
282.                                            if (token->start != -1 && token->end == -1) {
283.                                                    parser->toksuper = i;
284.                                                    break;
285.                                            }
286.                                    }
287.  #endif
288.                                    break;
289.                            case '\"':
290.                                    r = jsmn_parse_string(parser, js, len, tokens,
    num_tokens);
291.                                    if (r < 0) return r;
292.                                    count++;
293.                                    if (parser->toksuper != -1 && tokens != NULL)
294.                                            tokens[parser->toksuper].size++;
295.                                    break;
296.                            case '\t' : case '\r' : case '\n' : case ' ':
297.                                    break;
298.                            case ':':
299.                                    parser->toksuper = parser->toknext - 1;
300.                                    break;
301.                            case ',':
302.                                    if (tokens != NULL && parser->toksuper != -1 &&
303.                                            tokens[parser->toksuper].type !=
    JSMN_ARRAY &&
304.                                            tokens[parser->toksuper].type !=
    JSMN_OBJECT) {
305.  #ifdef JSMN_PARENT_LINKS
306.                                            parser->toksuper = tokens[parser-
    >toksuper].parent;
307.  #else
308.                                            for (i = parser->toknext - 1; i >= 0; i--) {
```

```
309.                                                                    if (tokens[i].type == JSMN_ARRAY ||
    tokens[i].type == JSMN_OBJECT) {
310.                                                                            if (tokens[i].start != -1
    && tokens[i].end == -1) {
311.                                                                                    parser->toksuper
    = i;
312.                                                                                    break;
313.                                                                            }
314.                                                                    }
315.                                                            }
316.  #endif
317.                                                    }
318.                                                    break;
319.  #ifdef JSMN_STRICT
320.                                    /* In strict mode primitives are: numbers and booleans */
321.                                    case '-': case '0': case '1' : case '2': case '3' : case '4':
322.                                    case '5': case '6': case '7' : case '8': case '9':
323.                                    case 't': case 'f': case 'n' :
324.                                            /* And they must not be keys of the object */
325.                                            if (tokens != NULL && parser->toksuper != -1) {
326.                                                    jsmntok_t *t = &tokens[parser->toksuper];
327.                                                    if (t->type == JSMN_OBJECT ||
328.                                                            (t->type == JSMN_STRING
    && t->size != 0)) {
329.                                                            return JSMN_ERROR_INVAL;
330.                                                    }
331.                                            }
332.  #else
333.                                    /* In non-strict mode every unquoted value is a primitive */
334.                                    default:
335.  #endif
336.                                            r = jsmn_parse_primitive(parser, js, len, tokens,
    num_tokens);
337.                                            if (r < 0) return r;
338.                                            count++;
339.                                            if (parser->toksuper != -1 && tokens != NULL)
340.                                                    tokens[parser->toksuper].size++;
341.                                            break;
342.
343.  #ifdef JSMN_STRICT
344.                                    /* Unexpected char in strict mode */
345.                                    default:
346.                                            return JSMN_ERROR_INVAL;
347.  #endif
348.                            }
349.                    }
350.
351.                    if (tokens != NULL) {
352.                            for (i = parser->toknext - 1; i >= 0; i--) {
353.                                    /* Unmatched opened object or array */
354.                                    if (tokens[i].start != -1 && tokens[i].end == -1) {
355.                                            return JSMN_ERROR_PART;
356.                                    }
357.                            }
358.                    }
359.
360.                    return count;
361.  }
362.
363.  /**
364.   * Creates a new parser based over a given  buffer with an array of tokens
365.   * available.
366.   */
367.  static void jsmn_init(jsmn_parser *parser) {
368.                    parser->pos = 0;
369.                    parser->toknext = 0;
370.                    parser->toksuper = -1;
371.  }
372.
373.  static int jsoneq(const char *json, jsmntok_t *tok, const char *s)
```

```
374.  {
375.            if (tok->type == JSMN_STRING
376.                && (int) strlen(s) == tok->end - tok->start
377.                && strncmp(json + tok->start, s, tok->end - tok->start) == 0) {
378.                      return 0;
379.            }
380.            return -1;
381.  }
382.
383.  static size_t load_file(char *filename)
384.  {
385.            FILE *f = fopen(filename, "rb");
386.            if(f == NULL) {
387.                      fprintf(stderr, "Unable to open file %s\n", filename);
388.                      exit(1);
389.            }
390.
391.            fseek(f, 0, SEEK_END);
392.            size_t fsize = ftell(f);
393.            fseek(f, 0, SEEK_SET);  //same as rewind(f);
394.
395.            if(fsize >= BUFF_SIZE) {
396.                      fprintf(stderr, "Configuration file too large\n");
397.                      abort();
398.            }
399.
400.            fread(config, fsize, 1, f);
401.            fclose(f);
402.
403.            config[fsize] = 0;
404.            return fsize;
405.  }
406.
407.  int parse_config(char *path, struct configuration *conf)
408.  {
409.            int i;
410.            int r;
411.            jsmn_parser p;
412.            jsmntok_t t[128]; /* We expect no more than 128 tokens */
413.
414.            load_file(path);
415.
416.            jsmn_init(&p);
417.            r = jsmn_parse(&p, config, strlen(config), t, sizeof(t)/sizeof(t[0]));
418.            if (r < 0) {
419.                      printf("Failed to parse JSON: %d\n", r);
420.                      return 0;
421.            }
422.
423.            /* Assume the top-level element is an object */
424.            if (r < 1 || t[0].type != JSMN_OBJECT) {
425.                      printf("Object expected\n");
426.                      return 0;
427.            }
428.
429.            /* Loop over all keys of the root object */
430.            for (i = 1; i < r; i++) {
431.                      if (jsoneq(config, &t[i], "host") == 0) {
432.                                /* We may use strndup() to fetch string value */
433.                                conf->host = strndup(config + t[i+1].start, t[i+1].end-
    t[i+1].start);
434.                                i++;
435.                      } else if (jsoneq(config, &t[i], "username") == 0) {
436.                                conf->db_username = strndup(config + t[i+1].start, t[i+1].end-
    t[i+1].start);
437.                                i++;
438.                      } else if (jsoneq(config, &t[i], "password") == 0) {
439.                                conf->db_password = strndup(config + t[i+1].start, t[i+1].end-
    t[i+1].start);
440.                                i++;
```

```
441.                    } else if (jsoneq(config, &t[i], "port") == 0) {
442.                            conf->port = strtol(config + t[i+1].start, NULL, 10);
443.                            i++;
444.                    } else if (jsoneq(config, &t[i], "database") == 0) {
445.                            conf->database = strndup(config + t[i+1].start, t[i+1].end-
    t[i+1].start);
446.                            i++;
447.                    } else {
448.                            printf("Unexpected key: %.*s\n", t[i].end-t[i].start, config +
    t[i].start);
449.                    }
450.            }
451.            return 1;
452.    }
453.
```

utils.c

```
1.  #include <stdio.h>
2.  #include <stdlib.h>
3.  #include <string.h>
4.
5.  #include "defines.h"
6.
7.  void print_stmt_error (MYSQL_STMT *stmt, char *message)
8.  {
9.    fprintf (stderr, "%s\n", message);
10.   if (stmt != NULL) {
11.             fprintf (stderr, "Error %u (%s): %s\n",
12.                      mysql_stmt_errno (stmt),
13.                      mysql_stmt_sqlstate(stmt),
14.                      mysql_stmt_error (stmt));
15.   }
16. }
17.
18.
19. void print_error(MYSQL *conn, char *message)
20. {
21.   fprintf (stderr, "%s\n", message);
22.   if (conn != NULL) {
23.             #if MYSQL_VERSION_ID >= 40101
24.             fprintf (stderr, "Error %u (%s): %s\n",
25.             mysql_errno (conn), mysql_sqlstate(conn), mysql_error (conn));
26.             #else
27.             fprintf (stderr, "Error %u: %s\n",
28.             mysql_errno (conn), mysql_error (conn));
29.             #endif
30.   }
31. }
32.
33. bool setup_prepared_stmt(MYSQL_STMT **stmt, char *statement, MYSQL *conn)
34. {
35.   _Bool update_length = true;
36.
37.   *stmt = mysql_stmt_init(conn);
38.   if (*stmt == NULL)
39.   {
40.             print_error(conn, "Could not initialize statement handler");
41.             return false;
42.   }
43.
44.   if (mysql_stmt_prepare (*stmt, statement, strlen(statement)) != 0) {
45.             print_stmt_error(*stmt, "Could not prepare statement");
46.             return false;
47.   }
48.
49.   mysql_stmt_attr_set(*stmt, STMT_ATTR_UPDATE_MAX_LENGTH, &update_length);
```

```
50.
51.   return true;
52. }
53.
54. void finish_with_error(MYSQL *conn, char *message)
55. {
56.   print_error(conn, message);
57.   mysql_close(conn);
58.   exit(EXIT_FAILURE);
59. }
60.
61. void finish_with_stmt_error(MYSQL *conn, MYSQL_STMT *stmt, char *message, bool close_stmt)
62. {
63.   print_stmt_error(stmt, message);
64.   if(close_stmt)      mysql_stmt_close(stmt);
65.   mysql_close(conn);
66.   exit(EXIT_FAILURE);
67. }
```