Basi di Dati e Conoscenza

Progetto A.A. 2020/2021

Progetto di una base di dati per un’agenzia di viaggi

Matricola:

0270638

Ossama El oukili

Indice

1. Descrizione del Minimondo

L'agenzia di viaggi per cui si vuole realizzare il sistema informativo gestionale svolge la sua attività principalmente nel settore dei tour di breve/media durata (da un singolo giorno, fino alla durata di una settimana). L'agenzia utilizza pullman Gran Turismo, e raggiunge con i suoi viaggi mete turistiche italiane ed europee.
L'agenzia, oltre ad occuparsi della formulazione delle offerte di viaggio ai clienti e della gestione delle prenotazioni, dispone di un proprio parco automezzi e di un'officina/deposito dove eseguire le revisioni sui pullman. Alle dipendenze dell'agenzia sono quindi anche un certo numero di meccanici e di autisti.
Gli autisti devono condurre i turisti attraverso le varie località in cui il loro tour prevede fermate. Queste possono aver luogo o presso alberghi, quando la durata del soggiorno nella relativa località supera la giornata, o presso beni artistici/ambientali. Le fermate, di entrambi i tipi, sono caratterizzate dal nome del bene o dell'albergo, dall'indirizzo e dall'eventuale numero di telefono. Il conducente vuole poter conoscere i dati temporali relativi a tutte le fermate del tour. In particolare per i beni da visitare è d'interesse la giornata e l'ora d'inizio della visita, mentre per gli alloggi si vogliono sapere le giornate e le ore di arrivo e di partenza. Ogni tour può essere replicato in più date. Altri dati dei tour sono la data di arrivo e il numero totale di Km. Gli autisti desiderano avere informazioni circa i tour cui sono stati assegnati e i mezzi scelti per i viaggi. Inoltre necessitano dei dati (nome, regione, stato) non solo dei luoghi in cui il tour si fermerà, ma di tutti quelli utili a tracciare con precisione il percorso del viaggio. A tal fine è utile poter disporre delle distanze tra le principali località come pure di cartine con diversi livelli di dettaglio (e indicazione della zona laddove la mappa sia relativa solo a parte di una città). Gli autisti, e i mezzi loro assegnati, hanno come unità minima di lavoro il giornaliero e a loro disposizione è un cellulare fornito dall'agenzia. Infine, per quanto riguarda i pullman, si dovranno memorizzare le seguenti informazioni: numero di posti, ingombri del veicolo, sua autonomia, targa e il valore del contatore di km (aggiornato dal conducente tra un viaggio e l'altro).
I meccanici si occupano di effettuare le revisioni sulle macchine dell'agenzia. Ogni meccanico è rintracciabile in officina tramite un cordless personale e ha competenze tecniche su almeno uno dei modelli di pullman dell'agenzia. Viceversa è necessario che, per ogni tipo di macchina, ci siano almeno due addetti con competenze specifiche. Di ogni modello deve essere noto il costruttore e un elenco di dati tecnici. I meccanici desiderano avere di ogni pullman il valore aggiornato del conta-km, nonché la data di immatricolazione e quella relativa alla prossima revisione presso il locale Uff. Motorizzazione. I controlli che i meccanici eseguono in officina possono essere di due diversi tipi: controlli periodici (detti "Ordinari") e revisioni occasionali (dette "Straordinarie") di cui deve essere registrato il motivo. Ogni revisione ha una data di inizio e una di fine e può comportare la sostituzione di un certo numero di componenti con altrettanti pezzi di ricambio. Per ridurre i tempi morti legati alle revisioni, si mantiene in officina anche un magazzino ricambi: ogni ricambio ha associati una descrizione, un costo unitario e i dati relativi a esistenza, scorta minima e quantità di riordino. Per pianificare le revisioni ordinarie, i meccanici mantengono aggiornati, per ogni mezzo, contatori chilometrici relativi agli ultimi controlli eseguiti. Infatti le revisioni ordinarie si suddividono in vari tipi, ciascuno caratterizzato da controlli specifici ed eseguito a intervalli chilometrici differenti. Naturalmente in una stessa revisione ordinaria su una macchina possono essere anche accorpati tagliandi di più tipi, se le relative validità sono state tutte superate.

I clienti possono accedere ai programmi di viaggio dell'agenzia attraverso un'interfaccia dedicata. Ogni programma può prevedere più partenze in date diverse, ma non cambiano, al variare della data di partenza, i diritti di iscrizione che il cliente è tenuto a versare e il numero minimo di partecipanti. Variano invece, con la data di partenza, il costo del viaggio (differenziato per adulti e bambini, camera singola con supplemento), per tener conto principalmente delle variazioni stagionali di prezzo degli alberghi, e il numero dei posti ancora disponibili. Delle mete incluse in un programma (e localizzate tramite nome, regione e stato), il cliente è interessato a conoscere le giornate e le ore di arrivo e partenza (nonché l'eventuale trattamento alberghiero, se la meta lo prevede). Un dato pacchetto comprende, in una certa località, un insieme di visite di cui i passeggeri vogliono conoscere: il nome del bene turistico oggetto della visita, se questa è compresa nella quota di viaggio, se si tratta, infine, di una visita guidata. La giornata di svolgimento, e l'ora d'inizio, non dipendono soltanto dalla particolare visita, ma sono soggette a variazioni in base al tipo di programma in cui il bene è inserito. Oltre a consentire la consultazione dei viaggi in programmazione, il sito dell'agenzia riporta una copiosa documentazione fotografica della maggior parte delle mete turistiche inserite nei programmi. Ogni foto, con relativa descrizione, è associata a una sola località, ma una stessa località può avere più foto. Esiste inoltre una sezione dedicata alla descrizione dei Pullman G. T. che costituiscono la flotta dell'agenzia: dei veicoli, oltre al numero di posti e alla casa costruttrice, si fornisce l'elenco dei comfort presenti a bordo (ogni comfort ha collegata una descrizione e può essere in dotazione a più pullman).

Le hostess hanno la responsabilità di gestire le principali attività connesse ai viaggi organizzati dall'agenzia. Ogni viaggio è relativo a un solo programma: ogni programma è contraddistinto da un nome univoco, da una quota d'iscrizione (che copre le spese di assicurazione medica e del bagaglio, e include garanzia di annullamento viaggio) e, infine, da un numero minimo di partecipanti. Uno stesso programma di viaggio può essere ripetuto in più date: i viaggi relativi differiscono tra loro (a parte le date di partenza/ritorno) per il costo (soggetto a variazioni stagionali, distinto tra adulti e bambini – considerati tali se di età inferiore ai 12 anni – e che non include il supplemento necessario per la sistemazione in camera singola) e per la disponibilità di posti sul Pullman. In caso di non superamento della soglia di partecipazione al viaggio, occorre registrare la data di annullamento del medesimo. A ciascun viaggio in programmazione, è associato il numero corrente di prenotazioni. I clienti (di cui si vogliono memorizzare i nominativi, gli indirizzi, i recapiti telefonici – compresi eventuale fax e email – e l'ultima data di invio documentazione sui tour dell'agenzia) possono effettuare le loro prenotazioni esclusivamente presso la sede, indicando alle hostess (oltre al nome del tour, alla data e ai numeri di posto scelti, tra quelli ancora disponibili per il viaggio) i nomi dei passeggeri, cui i posti vanno intestati, e le loro età. Sarà altresì compito delle hostess associare a ogni posto prenotato la data in cui la prenotazione è stata fatta e successivamente, e solo al superamento della soglia per la convalida del viaggio, quelle di avvenuta conferma del posto al cliente e di saldo del medesimo. Infine, molti tour prevedono la presenza a bordo del pullman di una delle hostess dell'agenzia: ad ogni modo l'associazione dell'eventuale hostess al viaggio viene fatta solo al superamento del numero minimo di partecipanti, inoltre l'impegno minimo per una accompagnatrice è dato dal tour giornaliero. Per quanto riguarda invece l'organizzazione dei Programmi di viaggio, bisogna dire che, tra tutte le località turistiche presenti nell'archivio dell'agenzia, ogni programma ne include un certo numero, relativamente alle quali occorre sapere le giornate/ore di arrivo/partenza e, se la località prevede pernottamento, anche una descrizione del trattamento garantito dall'albergo. A ogni sito presente in archivio, oltre al nome, regione e stato di appartenenza, possono essere collegati elenchi relativi agli alberghi e ai beni turistici

presenti nella località. Degli alberghi si vuole conoscere: nome, categoria, indirizzo, telefono (insieme a fax ed email, se ne possiede). In aggiunta, si desidera tener traccia dei vari servizi offerti dagli alberghi, per ognuno dei quali si riporta una breve descrizione. Delle attrazioni turistiche, invece, si mantengono il nome e, laddove abbiano significato, anche l'orario di apertura, l'indirizzo e il telefono. Una località, inclusa in un programma di viaggio, comprende almeno una visita (a un bene presente nella località) di cui si riportano: giornata e ora d'inizio, se guidata, se compresa nel costo del viaggio. Per concludere, le hostess hanno anche il compito di registrare tutte le prenotazioni fatte presso i vari alberghi associati a un certo viaggio: per ogni tipologia di camera (doppia, doppia con 3°/4° letto per bambini, singola, ...), si conservano il costo unitario e il numero di camere già prenotate.

2. Analisi dei Requisiti

Identificazione dei termini ambigui e correzioni possibili

Linea	Termine	Nuovo termine	Motivo correzione
9	tour	programma	Il termine tour viene usato sia per indicare programma, sia per indicare viaggio. Essendo queste ultime due entità distinte ho deciso di eliminare il termine tour dalla specifica. Essendo interessato solo alle fermate questo sono già definite nel viaggio (la specifica non fa riferimento alla possibilità che in un viaggio ci siano modifiche rispetto alla organizzazione imposta dal programma)
14	tour	viaggio	Il termine tour viene usato sia per indicare programma, sia per indicare viaggio. Essendo queste ultime due entità distinte ho deciso di eliminare il termine tour dalla specifica. In questo caso chiede i dati temporali relativi alle fermate. Tutti i dati temporali sono legati al viaggio e non al programma, non essendo legato a nessuna informazione temporale
16(1)	tour	programma	Il termine tour viene usato sia per indicare programma, sia per indicare viaggio. Essendo queste ultime due entità distinte ho deciso di eliminare il termine tour dalla specifica. In questo caso chiede i dati temporali relativi alle fermate. In questo si riferisce al fatto che il viaggio è una istanza di programma e che le varie istanze variano tra di loro anche per la data di partenza (ma ad esempio non nella durata)
16(2)	tour	viaggio	Il termine tour viene usato sia per indicare programma, sia per indicare viaggio. Essendo queste ultime due entità distinte ho deciso di eliminare il termine tour dalla specifica. In questo caso chiede i dati temporali relativi alle fermate. Tutti i dati temporali sono legati al viaggio e non al programma, non essendo legato a nessuna informazione temporale
18	tour	viaggio	Il termine tour viene usato sia per indicare programma, sia per indicare viaggio. Essendo queste ultime due entità distinte ho deciso di eliminare il termine tour dalla specifica. Gli autisti ed i pullman non sono associati perfroza sempre ad uno stesso

			programma. Sono invece assegnati ad un solo viaggio per volta.
19	tour	viaggio	Il termine tour viene usato sia per indicare programma, sia per indicare viaggio. Essendo queste ultime due entità distinte ho deciso di eliminare il termine tour dalla specifica. Si sarebbe potuto mettere anche programma. Ma essendo l'autista legato al viaggio potrebbe tornare utile mantenere viaggio.
34	revisione	Controllo ordinario	Possiamo considerare revisione come sinonimo di controllo. In questo caso però si fa rifarimento ad una revisione programmata. Gli unici controlli ad avere una organizzazione sono quelli ordianari.
44	controlli	operazioni	I vari tipi di controlli ordinari variano per le operazioni che i meccanici andranno ad eseguire. Il termine "controlli" si riferisce già ad un'altra entità
82	tour	viaggi	Il termine tour viene usato sia per indicare programma, sia per indicare viaggio. Essendo queste ultime due entità distinte ho deciso di eliminare il termine tour dalla specifica. I clienti inviano documentazione per un determinato viaggio in fase di organizzazione.
83	tour	programma	Il termine tour viene usato sia per indicare programma, sia per indicare viaggio. Essendo queste ultime due entità distinte ho deciso di eliminare il termine tour dalla specifica. Nella specifica non si accenna al fatto che un determinato viaggio sia caratterizzato da un nome, bensi da uno e un solo programma a cui fa riferimento. Suppongo quindi che il nome sia una caratterisitica esclusiva del programma.
88	tour	programma	Il termine tour viene usato sia per indicare programma, sia per indicare viaggio. Essendo queste ultime due entità distinte ho deciso di eliminare il termine tour dalla specifica. Suppongo che un determinato programma preveda sempre la presenza di un numero di hostess definito. Ho quindi escluso che due viaggi relativi ad uno stesso programma prevedano un numero diverso di hostess accompagnatrici.
91	tour	viaggio	Il termine tour viene usato sia per indicare programma, sia per indicare viaggio. Essendo queste ultime due entità distinte ho deciso di eliminare il termine tour dalla specifica. Le hostess sono assegnate al viaggio e non al programma.

Specifica disambiguata

L'agenzia di viaggi per cui si vuole realizzare il sistema informativo gestionale svolge la sua attività principalmente nel settore dei tour di breve/media durata (da un singolo giorno, fino alla durata di una settimana). L'agenzia utilizza pullman Gran Turismo, e raggiunge con i suoi viaggi mete turistiche(sinonimo) italiane ed europee.
L'agenzia, oltre ad occuparsi della formulazione delle offerte di viaggio ai clienti e della gestione delle prenotazioni, dispone di un proprio parco automezzi e di un'officina/deposito dove eseguire le revisioni sui pullman. Alle dipendenze dell'agenzia sono quindi anche un certo numero di meccanici e di autisti.
Gli autisti devono condurre i turisti attraverso le varie località in cui il loro programma prevede fermate. Queste possono aver luogo o presso alberghi, quando la durata del soggiorno nella relativa località supera la giornata, o presso beni artistici/ambientali (sinonimo). Le fermate, di entrambi i tipi, sono caratterizzate dal nome del bene (sinonimo) o dell'albergo, dall'indirizzo e dall'eventuale numero di telefono. Il conducente vuole poter conoscere i dati temporali relativi a tutte le fermate del viaggio. In particolare per i beni(sinonimo) da visitare è d'interesse la giornata e l'ora d'inizio della visita, mentre per gli alloggi si vogliono sapere le giornate e le ore di arrivo e di partenza. Ogni programma può essere replicato in più date. Altri dati dei viaggio sono la data di arrivo e il numero totale di Km(anche del programma). Gli autisti desiderano avere informazioni circa i viaggi cui sono stati assegnati e i mezzi scelti per i viaggi. Inoltre necessitano dei dati (nome, regione, stato) non solo dei luoghi(sinonimo) in cui il viaggio si fermerà, ma di tutti quelli utili a tracciare con precisione il percorso del viaggio. A tal fine è utile poter disporre delle distanze tra le principali località come pure di cartine con diversi livelli di dettaglio (e indicazione della zona laddove la mappa sia relativa solo a parte di una città). Gli autisti, e i mezzi loro assegnati, hanno come unità minima di lavoro il giornaliero e a loro disposizione è un cellulare fornito dall'agenzia. Infine, per quanto riguarda i pullman, si dovranno memorizzare le seguenti informazioni: numero di posti, ingombri del veicolo(sinonimo), sua autonomia, targa e il valore del contatore di km (aggiornato dal conducente tra un viaggio e l'altro).
I meccanici si occupano di effettuare le revisioni sulle macchine dell'agenzia. Ogni meccanico è rintracciabile in officina tramite un cordless personale e ha competenze tecniche su almeno uno dei modelli di pullman dell'agenzia. Viceversa è necessario che, per ogni tipo di macchina(sinonimo), ci siano almeno due addetti(sinonimo) con competenze specifiche. Di ogni modello deve essere noto il costruttore e un elenco di dati tecnici. I meccanici desiderano avere di ogni pullman il valore aggiornato del conta-km, nonché la data di immatricolazione e quella relativa al prossimo controllo ordinario presso il locale Uff. Motorizzazione. I controlli che i meccanici eseguono in officina possono essere di due diversi tipi: controlli periodici (detti "Ordinari") e revisioni(sinonimo) occasionali (dette "Straordinarie") di cui deve essere registrato il motivo. Ogni revisione(sinonimo) ha una data di inizio e una di fine e può comportare la sostituzione di un certo numero di componenti con altrettanti pezzi di ricambio. Per ridurre i tempi morti legati alle revisioni, si mantiene in officina anche un magazzino ricambi: ogni ricambio(sinonimo) ha associati una descrizione, un costo unitario e i dati relativi a esistenza, scorta minima e quantità di riordino. Per pianificare le revisioni ordinarie, i meccanici mantengono aggiornati, per ogni mezzo, contatori chilometrici relativi agli ultimi controlli eseguiti. Infatti le revisioni ordinarie si suddividono in vari tipi, ciascuno caratterizzato da operazioni specifiche ed eseguito a intervalli chilometrici differenti. Naturalmente in una stessa revisione ordinaria su una macchina possono essere anche accorpati tagliandi(sinonimo) di più tipi, se le relative validità(sinonimo) sono state tutte superate.
I clienti possono accedere ai programmi di viaggio dell'agenzia attraverso un'interfaccia dedicata. Ogni programma può prevedere più partenze in date diverse, ma non cambiano, al variare della data di partenza, i diritti di iscrizione che il cliente è tenuto a versare e il numero minimo di partecipanti. Variano invece, con la data di partenza, il costo del viaggio (differenziato per adulti e bambini,

camera singola con supplemento), per tener conto principalmente delle variazioni stagionali di prezzo degli alberghi, e il numero dei posti ancora disponibili. Delle mete (sinonimo) incluse in un programma (e localizzate tramite nome, regione e stato), il cliente è interessato a conoscere le giornate e le ore di arrivo e partenza (nonché l'eventuale trattamento alberghiero, se la meta lo prevede). Un dato pacchetto (sinonimo) comprende, in una certa località, un insieme di visite di cui i passeggeri vogliono conoscere: il nome del bene turistico(sinonimo) oggetto della visita, se questa è compresa nella quota di viaggio, se si tratta, infine, di una visita guidata. La giornata di svolgimento, e l'ora d'inizio, non dipendono soltanto dalla particolare visita, ma sono soggette a variazioni in base al tipo di programma in cui il bene(sinonimo) è inserito. Oltre a consentire la consultazione dei viaggi in programmazione, il sito dell'agenzia riporta una copiosa documentazione fotografica della maggior parte delle mete turistiche (sinonimo) inserite nei programmi. Ogni foto, con relativa descrizione, è associata a una sola località, ma una stessa località può avere più foto. Esiste inoltre una sezione dedicata alla descrizione dei Pullman G. T. che costituiscono la flotta dell'agenzia: dei veicoli, oltre al numero di posti e alla casa costruttrice, si fornisce l'elenco dei comfort presenti a bordo (ogni comfort ha collegata una descrizione e può essere in dotazione a più pullman).
Le hostess hanno la responsabilità di gestire le principali attività connesse ai viaggi organizzati dall'agenzia. Ogni viaggio è relativo a un solo programma: ogni programma è contraddistinto da un nome univoco, da una quota d'iscrizione (che copre le spese di assicurazione medica e del bagaglio, e include garanzia di annullamento viaggio) e, infine, da un numero minimo di partecipanti. Uno stesso programma di viaggio (sinonimo) può essere ripetuto in più date: i viaggi relativi differiscono tra loro (a parte le date di partenza/ritorno) per il costo (soggetto a variazioni stagionali, distinto tra adulti e bambini – considerati tali se di età inferiore ai 12 anni – e che non include il supplemento necessario per la sistemazione in camera singola) e per la disponibilità di posti sul Pullman(associato al viaggio). In caso di non superamento della soglia di partecipazione al viaggio, occorre registrare la data di annullamento del medesimo. A ciascun viaggio in programmazione, è associato il numero corrente di prenotazioni. I clienti (di cui si vogliono memorizzare i nominativi, gli indirizzi, i recapiti telefonici – compresi eventuale fax e email – e l'ultima data di invio documentazione sui viaggi dell'agenzia) possono effettuare le loro prenotazioni esclusivamente presso la sede, indicando alle hostess (oltre al nome del programma, alla data e ai numeri di posto scelti, tra quelli ancora disponibili per il viaggio) i nomi dei passeggeri, cui i posti vanno intestati, e le loro età. Sarà altresì compito delle hostess associare a ogni posto prenotato la data in cui la prenotazione è stata fatta e successivamente, e solo al superamento della soglia per la convalida del viaggio, quelle di avvenuta conferma del posto al cliente e di saldo del medesimo. Infine, molti programmi prevedono la presenza a bordo del pullman di una delle hostess dell'agenzia: ad ogni modo l'associazione dell'eventuale hostess al viaggio viene fatta solo al superamento del numero minimo di partecipanti, inoltre l'impegno minimo per una accompagnatrice è dato dal viaggio giornaliero. Per quanto riguarda invece l'organizzazione dei Programmi di viaggio, bisogna dire che, tra tutte le località turistiche(sinonimo) presenti nell'archivio dell'agenzia, ogni programma ne include un certo numero, relativamente alle quali occorre sapere le giornate/ore di arrivo/partenza e, se la località prevede pernottamento, anche una descrizione del trattamento garantito dall'albergo. A ogni sito(sinonimo) presente in archivio, oltre al nome, regione e stato di appartenenza, possono essere collegati elenchi relativi agli alberghi e ai beni turistici(sinonimo) presenti nella località. Degli alberghi si vuole conoscere: nome, categoria, indirizzo, telefono (insieme a fax ed email, se ne possiede). In aggiunta, si desidera tener traccia dei vari servizi offerti dagli alberghi, per ognuno dei quali si riporta una breve descrizione. Delle attrazioni turistiche, invece, si mantengono il nome e, laddove abbiano significato, anche l'orario di apertura, l'indirizzo e il telefono. Una località, inclusa in un programma di viaggio, comprende almeno una visita (a un bene(sinonimo) presente nella località) di cui si riportano: giornata e ora d'inizio, se guidata, se compresa nel costo del viaggio. Per concludere, le hostess hanno anche il compito di registrare tutte le prenotazioni fatte presso i vari alberghi associati

a un certo viaggio: per ogni tipologia di camera (doppia, doppia con 3°/4° letto per bambini, singola, ...), si conservano il costo unitario e il numero di camere già prenotate.	

Glossario dei Termini

Termine	Descrizione	Sinonimi	Collegamenti
Viaggio	È la cocncretizzazione del programma		Hostess, pullman
Programma	È il modello sul quale si basa un certo viaggio.	Pacchetto, programma di viaggio	Cartina, Località turistica
Modello pullman		modello	meccanico
Pullman	È il pullman fisico, con una propria matricola etc	Mezzo, veicolo, macchina, pullman G.T.	
Comfort offerto dal pullman			Modello pullman
Località turistica	È una città o in generale un luogo in cui ci si troverà durante il viaggio. All'interno della località sono presenti I beni e gli alberghi	Meta turistica, meta, località, città, sito	Programma, foto
cartina			Località turistica
Meccanico		addetto	
Autista			
Fermata	Luogo in cui ci si ferma col pullman. Puo essere o l'albergo o un bene. Non puo essere la località		

visita	Visita ad un bene turistico		
Albergo			
Attrazione turistica		Bene artistico, bene ambientale, bene turistico, bene	
controllo		Revision, tagliando	
Controllo ordinario	Controllo programmato dopo un certo numero di km percorsi dal mezzo	Controllo periodico, revision ordinaria	
Controllo occasionale	Controllo non programmato legato ad un malfunzionamento (inaspettato)	Revisione occasionale, revisione straordinaria	
Pezzo di ricambio		Ricambio	
foto			Località turistica
hostess			Viaggio,
cliente	Persona che si reca alla sede per prenotare il viaggio, possibilimente anche per alter persone e non necessariamente per se stesso		viaggio
passeggero	Persona che ha prenotato un viaggio		viaggio
Servizio offerto dall'albergo			albergo
Tipo di stanza	Tipo di camera prenotato per un certo viaggio in un certo albergo.		Viaggio, albergo

Raggruppamento dei requisiti in insiemi omogenei

Frasi relative a Viaggio
- Ogni viaggio è relativo a un solo programma

- Variano tra loro per le date di partenza/ritorno, per il costo e per la disponibilità di posti sul Pullman. - A ciascun viaggio in programmazione, è associato il numero corrente di prenotazioni. - Il costo è distinto tra adulti e bambini, considerati tali se di età inferiore ai 12 anni, e che non include il supplemento necessario per la sistemazione in camera singola - molti tour (programma)prevedono la presenza a bordo del pullman di una delle hostess. l'associazione dell'eventuale hostess al viaggio viene fatta solo al superamento del numero minimo di partecipanti - il sito dell'agenzia consente la consultazione dei viaggi in programmazione
Frasi relative a Programma
- ogni programma è contraddistinto da un nome univoco, da una quota d'iscrizione e da un numero minimo di partecipanti. - Ogni tour (programma) può essere replicato in più date. - è utile poter disporre delle distanze tra le principali località come pure di cartine - ogni programma include un certo numero di localita turistiche, relativamente alle quali occorre sapere le giornate/ore di arrivo/partenza e, se la località prevede pernottamento, anche una descrizione del trattamento garantito dall'albergo. - molti tour (programma)prevedono la presenza a bordo del pullman di una delle hostess
Frasi relative a Modello pullman
- per ogni modello di pullman ci sono almeno due addetti con competenze specifiche. - per ogni modello deve essere noto il costruttore e un elenco di dati tecnici. - si memorizza il numero di posti, ingombri del veicolo, sua autonomia. - Esiste una sezione del sito dedicata alla descrizione dei Pullman G. T. che costituiscono la flotta dell'agenzia: dei veicoli, oltre al numero di posti e alla casa costruttrice, si fornisce l'elenco dei comfort presenti a bordo
Frasi relative a Pullman
- si memorizza la targa, il valore del contatore di km, data di immatricolazione e data del prossimo controllo oridinario
Frasi relative a Comfort offerto dal pullman
- ogni comfort ha collegata una descrizione e può essere in dotazione a più pullman
Frasi relative a Località turistica
- le mete (località turistica) incluse in un programma sono localizzate tramite nome, regione e stato e possono essere italiane ed europee. - possono essere collegati elenchi relativi agli alberghi e ai beni turistici (attrazioni turistiche) presenti nella località (località turistica).

- comprende almeno una visita ad un bene presente nella località stessa - una stessa località può avere più foto
Frasi relative a cartina
- posso avere diversi livelli di dettaglio - è indicate la zona laddove la mappa sia relativa solo a parte di una città
Frasi relative a Meccanico
- Ogni meccanico è rintracciabile in officina tramite un cordless personale. - Ogni meccanico ha competenze tecniche su almeno uno dei modelli di pullman dell'agenzia. -i meccanici mantengono aggiornati, per ogni mezzo, contatori chilometrici relativi agli ultimi controlli eseguiti.
Frasi relative a Autista
- hanno come unità minima di lavoro il giornaliero - è a loro disposizione un cellulare fornito dall'agenzia - desiderano avere informazioni circa viaggi cui sono stati assegnati e i mezzi scelti per i viaggi. Inoltre necessitano dei dati (nome, regione, stato) non solo delle località in cui il programma si fermerà, ma di tutti quelli utili a tracciare con precisione il percorso del viaggio. In particolare vuole poter conoscere la giornata e l'ora d'inizio della visita, per gli alloggi le giornate e le ore di arrivo e di partenza.
Frasi relative a Fermata
Le fermate sono caratterizzate dal nome del bene (attrazione turistica) o dell'albergo, dall'indirizzo e dall'eventuale numero di telefono. Le fermate possono aver luogo o presso alberghi o presso beni artistici/ambientali (attrazioni turistiche).possono aver luogo presso gli alberghi solo quando la durata del soggiorno nella relativa località (località turistica) supera la giornata,
Frasi relative a Albergo
- Degli alberghi si vuole conoscere nome, categoria, indirizzo, telefono (insieme a fax ed email, se ne possiede). - si desidera tener traccia dei vari servizi offerti dagli alberghi, per ognuno dei quali si riporta una breve descrizione. - è presente una descrizione del trattamento offerto
Frasi relative a Attrazione turistica
- per ogni bene sappiamo il nome del bene turistico (attrazione turistica) e, laddove abbiano significato, anche l'orario di apertura, l'indirizzo e il telefono.

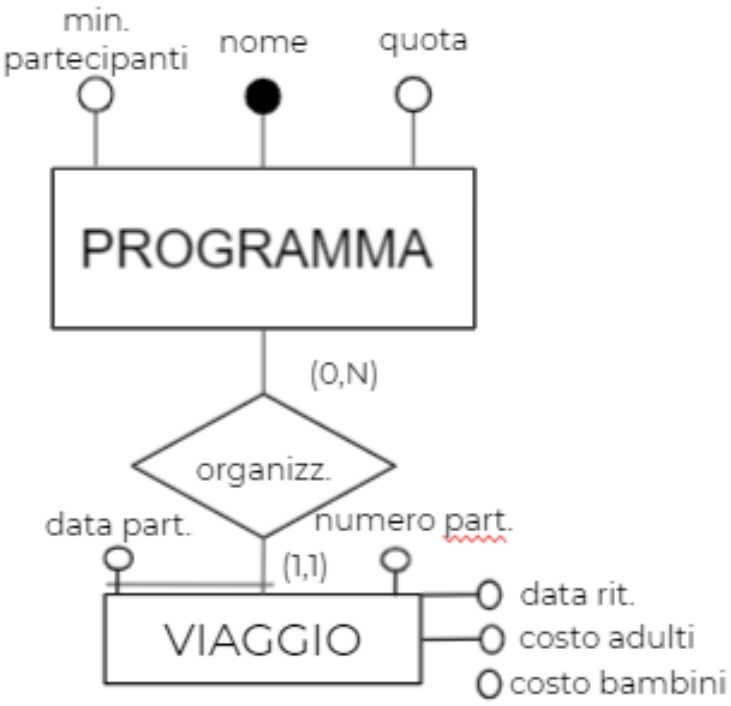
Frasi relative a controllo
- I controlli che i meccanici eseguono in officina possono essere di due diversi tipi: controlli periodici (detti “Ordinari”) e revisioni occasionali (dette “Straordinarie”) - Ogni controllo ha una data di inizio e una di fine - ogni controllo può comportare la sostituzione di un certo numero di componenti con altrettanti pezzi di ricambio
Frasi relative a Controllo ordinario
- si suddividono in vari tipi, ciascuno caratterizzato da controlli specifici ed eseguito a intervalli chilometrici differenti. - in una stessa sessione di controllo ordinaria su una macchina possono essere anche accorpati controlli ordinari di più tipi, se le relative validità sono state tutte superate.
Frasi relative a Controllo occasionale
- deve essere registrato il motivo
Frasi relative a Pezzo di ricambio
- ogni ricambio ha associati una descrizione, un costo unitario e i dati relativi a esistenza, scorta minima e quantità di riordino.
Frasi relative a foto
- Ogni foto, con relativa descrizione, è associata a una sola località (località turistica), ma una stessa località può avere più foto - il sito dell’agenzia consente la consultazione delle foto
Frasi relative a hostess
- compito delle hostess associare a ogni posto prenotato la data in cui la prenotazione è stata fatta e successivamente, e solo al superamento della soglia per la convalida del viaggio, quelle di avvenuta conferma del posto al cliente e di saldo del medesimo. -molti tour (programma)prevedono la presenza a bordo del pullman di una delle hostess. l’associazione dell’eventuale hostess al viaggio viene fatta solo al superamento del numero minimo di partecipanti - le hostess hanno anche il compito di registrare tutte le prenotazioni fatte presso i vari alberghi associati a un certo viaggio
Frasi relative a cliente
- si vogliono memorizzare i nominativi, gli indirizzi, i recapiti telefonici ,compresi eventuale fax e email e l’ultima data di invio documentazione sui viaggi - effettua le prenotazioni indicando nome del programma, data e i numeri di posto scelti, tra quelli ancora disponibili per il viaggio, i nomi dei passeggeri, cui i posti vanno intestati, e le loro età. - il cliente è interessato a conoscere le giornate e le ore di arrivo e partenza del viaggio
Frasi relative a passeggero

- i passeggeri vogliono conoscere: il nome del bene turistico (attrazione turistica) oggetto della visita, se questa è compresa nella quota di viaggio, se si tratta, infine, di una visita guidata. - si memorizza nome ,posto e età
Frasi relative a Servizio offerto dall'albergo
- per ogni servizio si riporta una breve descrizione
Frasi relative a tipo di stanza
- per ogni tipologia di camera (doppia, doppia con 3°/4° letto per bambini, singola, ...), si conservano il costo unitario e il numero di camere già prenotate.
Frasi relative a visita
- si riporta il nome del bene turistico, giornata e ora d'inizio, se guidata, se compresa nel costo del viaggio.

3. Progettazione concettuale

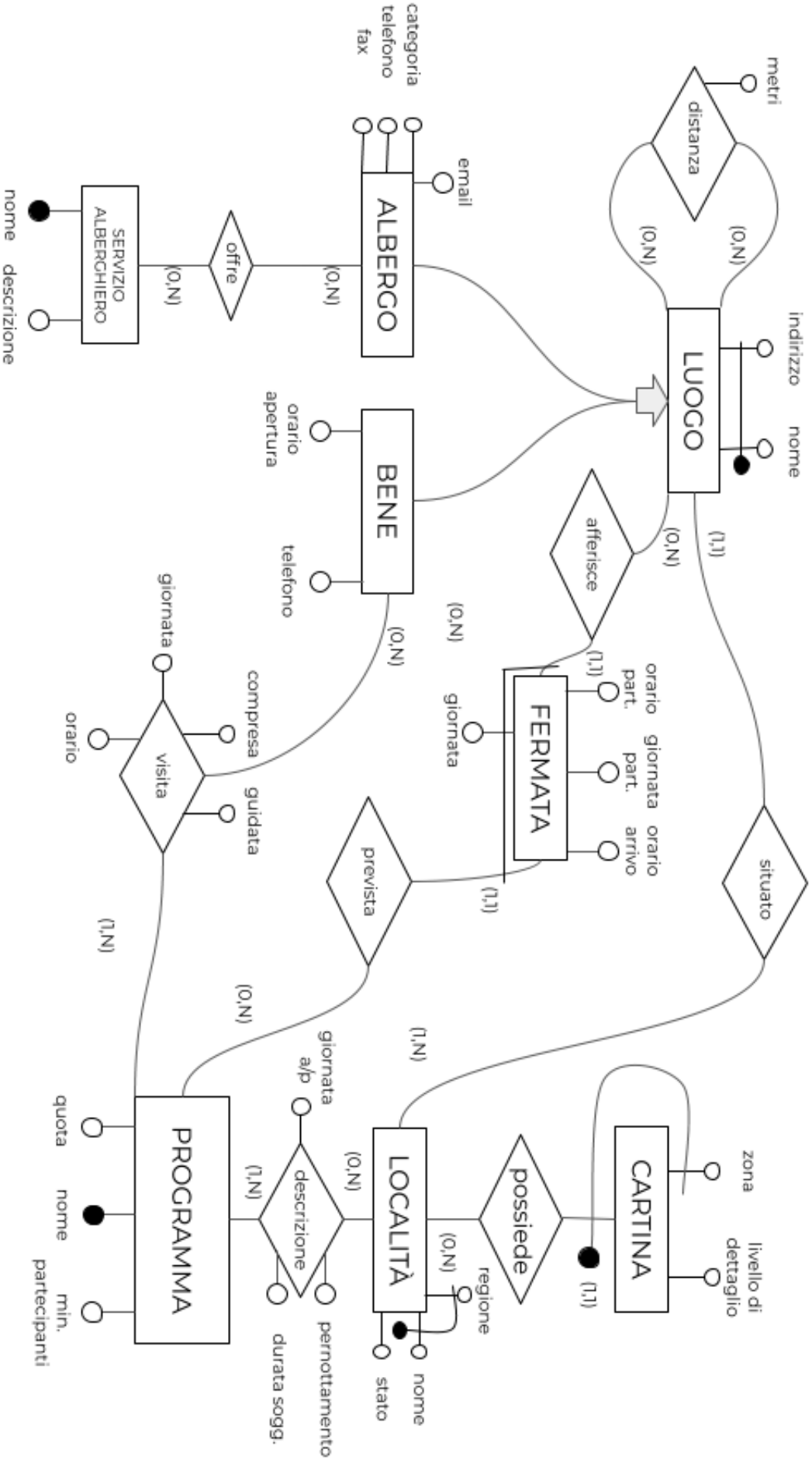
Costruzione dello schema E-R

Sono partito strutturando il nucleo del mia specifica:



Da qui ho lavorato intorno suddividendo nei vari temi.

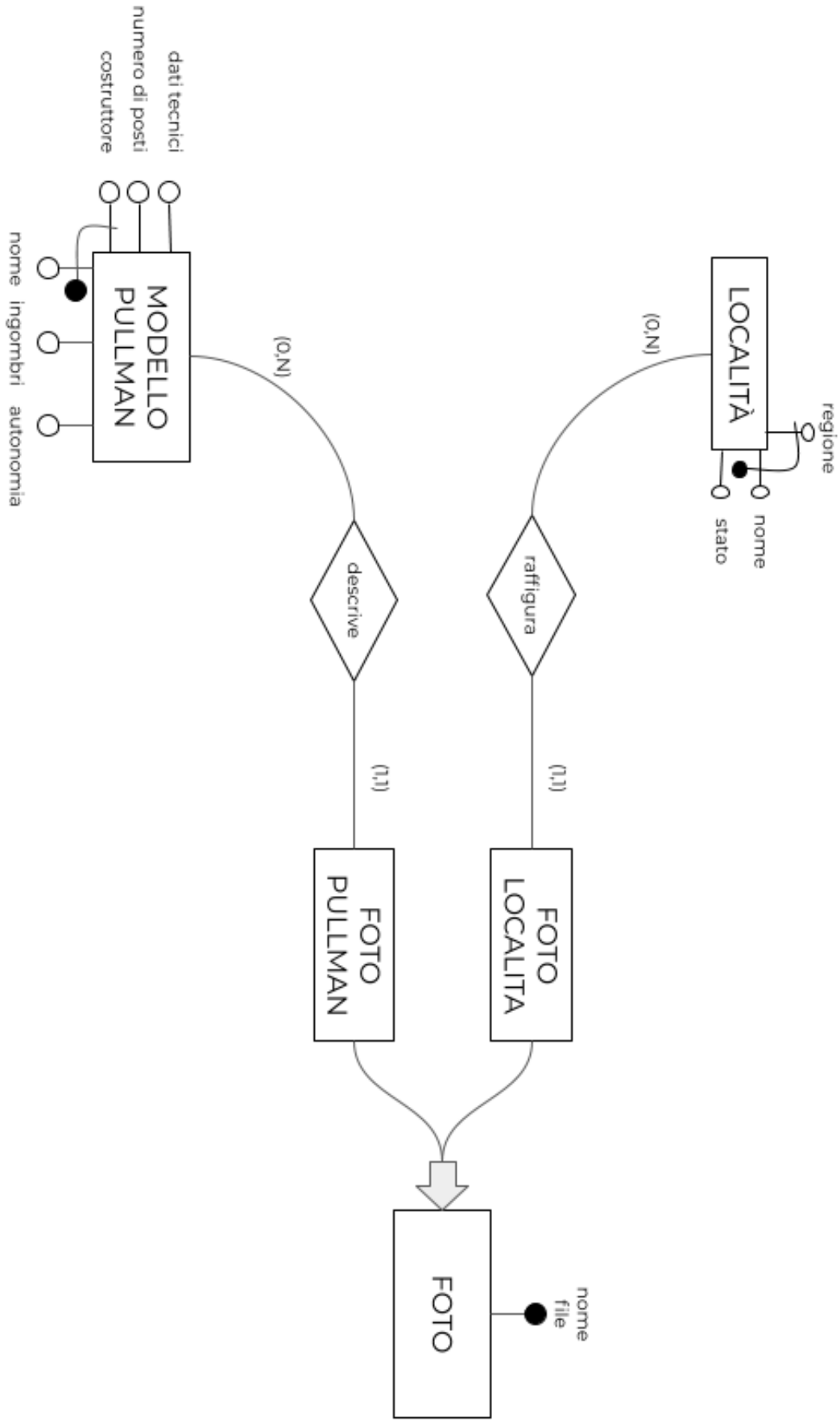
Ho schematizzato in primis tutto ciò che riguardava le località:


LUOGO
indirizzo
nome
distanza
metri
(0,N)
(0,N)
ALBERGO
categoria
telefono
fax
email
offre
(0,N)
(0,N)
SERVIZIO ALBERGHIERO
nome
descrizione
BENE
orario apertura
telefono
afferisce
(0,N)
(1,1)
FERMATA
orario part.
giornata part.
orario arrivo
giornata
(1,1)
(0,N)
situato
(1,1)
CARTINA
zona
livello di dettaglio
possiede
(1,1)
(0,N)
regione
LOCALITÀ
nome
stato
pernottamento
(1,N)
(0,N)
prevista
(0,N)
(1,N)
visita
giornata
orario
compresa
guidata
(1,N)
descrizione
giornata a/p
durata sogg.
(1,N)
PROGRAMMA
quota
nome
min. partecipanti

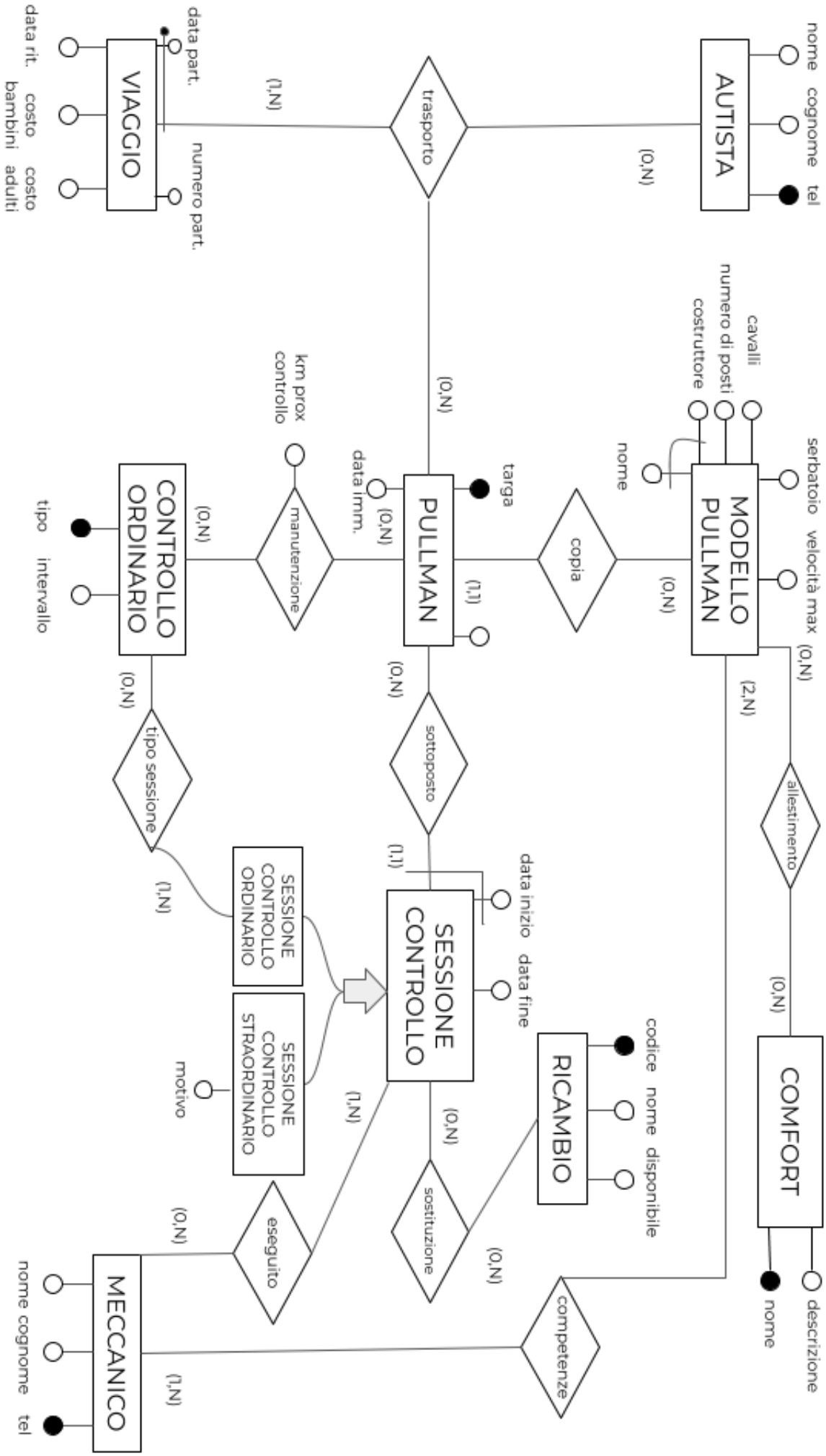
Sono poi passato a schematizzare tutto ciò che ruota intorno al viaggio:

ALBERGO
categoria
telefono
fax
email
(0,N)
TIPO STANZA
tipo
(0,N)
costo
prenotazione stanza
n° stanze prenotate
(1,N)
PROGRAMMA
min. partecipanti
nome
quota
(0,N)
organizz.
(1,1)
data part.
numero part.
VIAGGIO
data rit.
costo adulti
costo bambini
(0,N)
partecipa
data invio doc.
data prenot.
data invio doc.
PASSEGGERO
nome
età
posto
(1,1)
veci
(0,N)
CLIENTE
email
fax
indirizzo
nome
tel
VIAGGIO CANC.
data canc.
VIAGGIO CONF.
(1,1)
visita
(0,N)
HOSTESS
nome
cognome
tel.

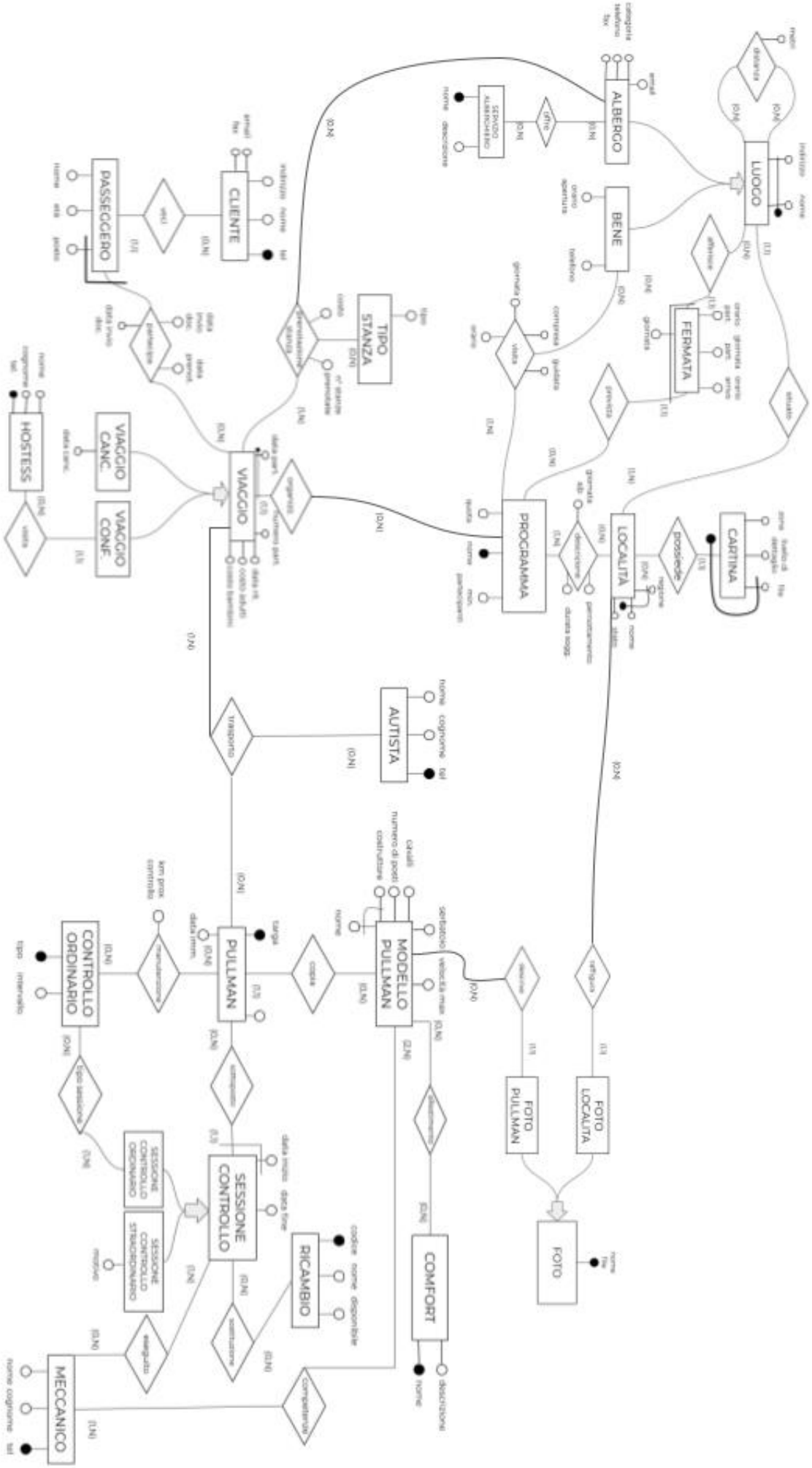
La parte che riguarda le foto:


LOCALITÀ
regione
nome
stato
(0,N)
raffigura
(1,1)
FOTO LOCALITA
MODELLO PULLMAN
dati tecnici
numero di posti
costruttore
nome ingombri autonomia
(0,N)
descrive
(1,1)
FOTO PULLMAN
FOTO
nome file

La parte che riguarda i pullman:


AUTISTA
nome
cognome
tel
VIAGGIO
data part.
data rit.
costo bambini
costo adulti
numero part.
trasporto
(1,N)
(0,N)
PULLMAN
targa
data imm.
km prox controllo
(0,N)
(1,1)
MODELLO PULLMAN
cavalli
numero di posti
costruttore
nome
serbatoio
velocità max
(0,N)
(0,N)
copia
manutenzione
CONTROLLO ORDINARIO
tipo
intervallo
(0,N)
(0,N)
COMFORT
nome
descrizione
(2,N)
(0,N)
allestimento
competenze
(0,N)
(1,N)
MECCANICO
nome
cognome
tel
sottoposto
(0,N)
(1,1)
tipo sessione
(0,N)
(1,N)
SESSIONE CONTROLLO
data inizio
data fine
(1,1)
(1,N)
(0,N)
SESSIONE CONTROLLO ORDINARIO
SESSIONE CONTROLLO STRAORDINARIO
motivo
eseguito
(0,N)
(1,N)
sostituzione
(0,N)
RICAMBIO
codice
nome
disponibile

Lo schema complessivo:


LUOGO
ALBERGO
BENE
FERMATA
CARTINA
LOCALITÀ
PROGRAMMA
VIAGGIO
TIPO STANZA
CLIENTE
PASSEGGERO
HOSTESS
VIAGGIO CANC.
VIAGGIO CONF.
AUTISTA
PULLMAN
MODELLO PULLMAN
CONTROLLO ORDINARIO
SESSIONE CONTROLLO
SESSIONE CONTROLLO ORDINARIO
SESSIONE CONTROLLO STRAORDINARIO
RICAMBIO
COMFORT
MECCANICO
FOTO
FOTO PULLMAN
FOTO LOCALITÀ

Integrazione finale

Regole aziendali

RV1 – Le fermate possono aver luogo presso alberghi solo se la durata del soggiorno nella relativa località supera la giornata.

RV2 - Gli autisti, e i mezzi loro assegnati, hanno come unità minima di lavoro il giornaliero.

RV3 - I controlli ordinari possono essere effettuati solo se le relative validità sono state superate.

RV4 - I bambini sono considerati tali se di età inferiore ai 12 anni.

RV5 - In caso di non superamento della soglia di partecipazione il viaggio viene annullato.

RV6 - L'impegno minimo per una accompagnatrice è dato dal viaggio giornaliero.

RV7 – la giornata di visita deve essere compresa tra giornata di arrivo e giornata di partenza

RV8 – Le località possono essere o italiane o europee.

RV9 – Dato un programma e una delle località comprese nel pacchetto deve esserci almeno una visita programma in uno dei beni presenti nella località.

RV10 – Ad un certo viaggio devono afferire lo stesso numero di autisti e di pullman.

RV11 – Una hostess puo essere associata ad un solo viaggio per volta.

RV12 – Una certa istanza di viaggio non puo essere associata sia a viaggio conf. Che a viaggio canc.

RV13– una foto deve avere una e una sola associazione, o con un modello di pullman o con una località

RV14– un autista e un mezzo posso essere assegnati ad un viaggio per volta, un autista puo essere assegnato ad un solo pullman per volta , un pullman ad un solo autista per volta (ipotizzando che non ci siano autisti che si danno il cambio)

RV15 – l'entità sessione di controllo ha il campo motivo null se è ordinario, diverso da null se è straordinario

RV16 – i pullman hanno targa europea

Dizionario dei dati

Entità	Descrizione	Attributi	Identificatori
Localita		nome, regione, stato	nome, regione
Programma	Struttura organizzativa di un viaggio che viene riproposta. Un'agenzia ne ha un numero limitato che ripropone	Nome, quota di iscrizione, numero minimo partecipanti,	Nome
Viaggio	Viaggio che viene organizzato dall'agenzia sulla base di un programma pre strutturato.	Data di partenza, data di ritorno, costo adulti, costo bambini	Programma, Data di partenza
Viaggio cancellato	Viaggio che non raggiunge la soglia minima di partecipanti	Data cancellazione	
Viaggio confermato	Viaggio che raggiunge la soglia minima di partecipanti		
Luogo	Luogo che si trova all'interno di una certa località	Nome, indirizzo	Nome, indirizzo
Albergo		categoria, telefono, fax, email, servizi	
Tipo stanza			
Servizio alberghiero		Nome, descrizione	nome
Bene	Bene turistico	l'orario di apertura, il telefono	
Fermata	Tutte le tappe che compiono i pullman secondo il programma	Orario partenza, orario arrive, giornata partenza, giornata	Giornata, luogo, programma
Cliente	Persona che per contro di pontenziali passeggeri acquista i bliglietti	nome, indirizzo, telefono, fax, email, l'ultima	Telefono, viaggio

		data di invio documentazione	
Passeggero	Persona per cui c'è una prenotazzione attiva per un viaggio	Nome, posto, età	viaggio
Foto		nome file	Nome file
Foto pullman			
Foto località			
Modello pullman		Nome, Dati tecnici, costruttore, numero di posti, ingombri del veicolo, autonomia,	Nome, costruttore
Pullman	Vettura concreta prodotta sulla base di un certo modello.	targa, valore del contatore di km, data di immatricolazione, data prossimo controllo oridinario	targa
Autista		Nome, cellulare	cellulare
Meccanico	Addetto che si occupa della manutenzione delle vetture dell'agenzia	Nome, cordless	cordless
Sessione di controllo	Sessione di controllo che puo essere sia straordinaria che ordinaria	data di inizio, data di fine	data di inizio, pullman
Sessione di controllo ordinario			
Sessione di controllo straordinario		motivo	
Hostess		nome	nome
Comfort pullman		Nome, descrizione	Nome
Cartina	Una cartina a supposrto degli autisti, riferita ad una localit a o ha parte di essa	Livello di dettaglio, zona	Località, zona

Ricambio		Codice, disponibile	codice
Tipo stanza		tipo	tipo

4. Progettazione logica

Volume dei dati

Concetto nello schema	Tipo[1]	Volume atteso
Viaggi	E	250
Localita	E	160
Programma	E	80
Viaggio	E	240
Viaggio cancellato	E	10
Viaggio confermato	E	230
Luogo	E	4800
Albergo	E	160
Tipo di stanza	E	8
Prenotazione stanza	R	2000
Servizio alberghiero	E	15
Bene	E	3000
Fermata	E	2500
Cliente	E	2300
Passeggero	E	7000
Foto	E	3040
Foto pullman	E	40
Foto località	E	3000
Modello pullman	E	4
Pullman	E	12
Autista	E	25

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

Meccanico	E	8
Sessione di controllo	E	75
Controllo ordinario	E	8
Sessione Controllo straordinario	E	25
Sessione di controllo ordinario	E	50
Hostess	E	30
Comfort pullman	E	20
Cartina	E	300
Visite	R	3000

Tavola delle operazioni

Cod.	Descrizione	Frequenza attesa
OP1	RICERCA DATI TEMPORALI FERMATE DEL GIORNO	(2500 fermate/anno)*(2/fermata) = 5000/anno
OP2	RICERCA VIAGGIO E MEZZO ASSEGNATO	(250 viaggi/anno)*(1.5/viaggi) = 375/anno
OP3	INFO VISITE DI UN VIAGGIO (autista)	(250 viaggi/anno)*(3/viaggi) = 750/anno
OP4	DISTANZA TRA LUOGHI	(250 viaggi/anno)*(10/viaggi) = 2500/anno
OP5	INFO ALLOGGI VIAGGIO	(250 viaggi/anno)*(3/viaggi) = 750/anno
OP6	CARTINE DISPONIBILI	(250 viaggi/anno)*(5/viaggi) = 1250/anno
OP7	RICERCA VIAGGI ASSEGNATI	(250 viaggi/anno)*(2/viaggio) = 500/anno
OP8	INSERIMENTO CLIENTE	2300/anno
OP9	INSERIMENTO PASSEGGERO	7000/anno
OP10	CONFERMA VIAGGIO	240/ anno
OP11	PRENOTAZIONE STANZA	2000/anno

OP12	INSERIMENTO COSTO STANZA	(160 alberghi)*(5 stanze) = 1000 alberghi* stanze
OP13	RICERCA PROSSIMA REVISIONE ORDINARIA PULLMAN	(8 tipi)*(12 pullman)*(5/tipi*pullman*anno) =480/anno
OP14	EFFETTUA REVISIONE	75/anno
OP15	LEGGI CONTATORE PULLMAN	40/anno
OP16	UTILIZZA PEZZO DI RICAMBIO	75 revisioni/anno*0.5 revisione = 40/anno
OP17	ORDINA RICAMBIO	15/anno
OP18	VISUALIZZARE STATO VIAGGIO	7000 passeggeri/anno * 3/passeggero = 21000 /anno
OP19	INFO VISITE DI UN VIAGGIO (passeggero)	7000 passeggeri/anno * 10 passeggero = 70000/anno
OP20	VISUALIZZA VIAGGI PRENOTABILI	100000/anno
OP21	REVISIONI DA EFFETTUARE	1 al giorno

Costo delle operazioni

OP1 – 1 L ‘luogo’ + 1 L ‘fermata’ = 2

OP2 – 1 L ‘autista’ + 1 L ‘trasporto’ = 2

OP3 – 1 L ‘luogo’ * 12 visite + 1 L ‘visita’ = 13

OP4 – 1 L ‘distanza’ = 1

OP5 – (1 L ‘luogo’ + 1 L ‘fermata’) * 2 alberghi + 1 L ‘albergo’ = 5

OP6 – 1 L ‘località’ + 1 L ‘cartina’ * 2 località = 3

OP7 – 1 L ‘viaggio’ = 1

OP8 – 1 S ‘cliente’ = 2

OP9 – 1 S ‘passeggero’ + 1 S ‘viaggio’ = 4

OP10 – 1 S ‘viaggio’ = 2

OP11 – 1 L ‘viaggio’ + 1 S ‘prenotazione stanza’ = 3

OP12 – 1 L ‘viaggio’ + 1 S ‘prenotazione stanza’ = 3

OP13 – 1 L ‘manutenzione’ = 1

OP14 – 1 S ‘sessione controllo’ + 1 S ‘eseguito' + 4*(1 S ‘tipo sessione') = 12

OP15 – 1 L ‘pullman’ = 1

OP16 – 1 L ‘ricambio’ + 1 S ‘sostituzione’ = 3

OP17 – 1 S ‘ricambio’ = 2

OP18 – 1 L ‘viaggio’ = 1

OP19 – 1 L ‘visite’ + 12 * (1 L ‘luogo’) = 13

OP20 – 1 L ‘viaggio’ = 1

OP21 – 1 L pullman + 12 * (1 L ‘manutenzione’) = 13

Ristrutturazione dello schema E-R

1) Analisi delle ridondanze

- Il numero di partecipanti ad un viaggio può essere ricavato vedendo quanti passeggeri sono legati a quel viaggio. Tengo però l’attributo per rendere l’operazione di conferma meno dispendiosa.
- Il pernottamento in una località può essere dedotto dalla giornata di arrivo e da quella di partenza. Elimino quindi l’attributo pernottamento.
- Anche la durata del soggiorno può essere dedotta dalla giornata di arrivo e da quella di partenza . Elimino quindi l’attributo durata soggiorno.
- Le operazioni riguardanti le visite hanno un costo elevato, data la necessita di dover accedere per ogni bene al luogo corrispettivo per ricavarne il nome. Potremmo aggiungere il nome del bene sia alle entità ‘bene’ e ‘visita’ per far scendere il costo delle operazioni 14 e 19 a 1. considerando che l’operazione 19 e una delle più frequenti ho deciso di aggiungere queste ridondanze. Ora i costi sono:

  OP3 – 1 L ‘visita’ = 1

  OP19 – 1 L ‘visite’ = 1

2) Eliminazione delle generalizzazioni

- Luogo

  La generalizzazione è parziale e per questo scartiamo l'accorpamento del genitore nei figli. I due figli sono molto diversi tra di loro e anche col genitore quindi l'accorpamento dei figli nel padre porterebbe ad una entità con molti attributi nulli e complicheremmo le relazioni. Inoltre le operazioni fanno molta differenza tra i figli; evitiamo perciò l'accorpamento dei figli nel padre sostituiamo quindi le generalizzazioni con delle associazioni

- Foto

  Non avendo operazioni particolari sulle foto e non avendo praticamente nessun attributo se non il nome del file possiamo inglobare i figli nel genitore.

- Sessione di controllo

  È evidente che conviene accorpare i figli nel genitore

- Viaggio

  Anche qui accorpiamo i figli nel genitore aggiungendo l'attributo "stato" che può essere "confermato" "cancellato" "organizzazione"; aggiungiamo anche l'attributo

  "data cancellazione"

3) Scelta degli identificatori primari

Non sono presenti entità con più di un identificatore.

Trasformazione di attributi e identificatori

Traduzione di entità e associazioni

Programma(**nome**, quota, min. Partecipanti)

Viaggio(**data partenza, itinerario**, numero part., ritorno, costo adulti, costo bambini, stato, hostess, data cancellazione)

Hostess(**tel**, nome, cognome)

Cliente(**tel,** indirizzo, nome, email, Fax)

Passeggero(nome, prenotatore, età, **posto**, **data viaggio, itinerario**,data prenotazione, data invio documenti)

Descrizione(**regione località, nome località, itinerario,** giornata arrivo, giornata partenza)

Località(**regione, nome**, stato)

Cartina(zona**, regione località, nome località**, livello dettaglio, **file**)

Luogo(**indirizzo**, nome, regione località, nome località)

Distanza(**indirizzo 1, indirizzo 2,** metri)

Albergo(**indirizzo**, email, categoria, telefono, fax)

Prenotazione stanza(**indirizzo albergo, data partenza, itinerario, tipo stanza,** costo, n° stanze prenotate)

Bene(**indirizzo,**orario apertura, telefono)

Visita(**itinerario, indirizzo bene, orario**, compresa, guidata, giornata)

Offre(**indirizzo albergo, servizio alberghiero**)

Servizio alberghiero**(nome,** descrizione**)**

Fermata(**indirizzo luogo, giornata**, **itinerario,** orario arrivo, orario partenza, giornata partenza)

Foto(**nome file**, regione località, nome località, pullman)

Autista(**telefono**, nome, cognome)

Pullman(**targa**, contatore, data immatricolazione, costruttore, nome modello)

Trasporto(**autista, pullman, data partenza, itinerario**)

Modello pullman(velocita max, cavalli, serbatoio, numero di posti, **costruttore, nome**)

Allestimento(**costruttore, nome modello, comfort**)

Comfort**(nome**, descrizione)

manutenzione(**pullman, controllo ordinario**, km prox controllo)

Controllo ordinario(**tipo**, intervallo)

Tipo sessione(**controllo ordinario, sessione controllo**)

Sessione controllo(**data inizio, pullman**, data fine, motivo)

Sostituzione(**ricambio, data inizio, pullman**)

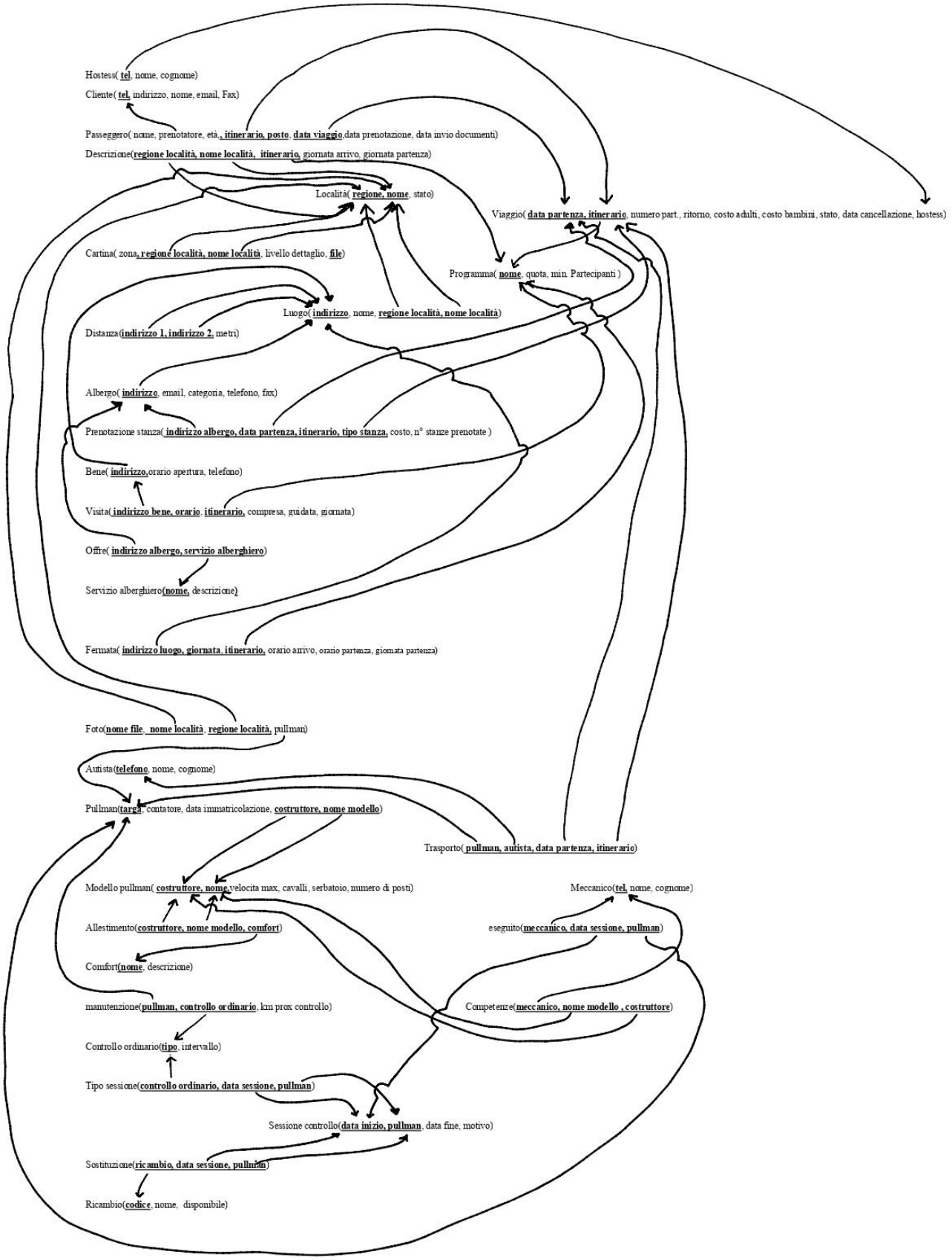
Ricambio(**codice**, nome, disponibile)

Meccanico(**tel,** nome, cognome)

eseguito(**meccanico, data inizio, pullman**)

Competenze(**meccanico, costruttore, nome modello**)

Rappresentazione grafica:


Hostess(tel, nome, cognome)
Cliente(tel, indirizzo, nome, email, Fax)
Passeggero(nome, prenotatore, età, itinerario, posto, data viaggio,data prenotazione, data invio documenti)
Descrizione(regione località, nome località, itinerario, giornata arrivo, giornata partenza)
Località(regione, nome, stato)
Viaggio(data partenza, itinerario, numero part., ritorno, costo adulti, costo bambini, stato, data cancellazione, hostess)
Cartina(zona, regione località, nome località, livello dettaglio, file)
Programma(nome, quota, min. Partecipanti)
Luogo(indirizzo, nome, regione località, nome località)
Distanza(indirizzo 1, indirizzo 2, metri)
Albergo(indirizzo, email, categoria, telefono, fax)
Prenotazione stanza(indirizzo albergo, data partenza, itinerario, tipo stanza, costo, n° stanze prenotate)
Bene(indirizzo,orario apertura, telefono)
Visita(indirizzo bene, orario, itinerario, compresa, guidata, giornata)
Offre(indirizzo albergo, servizio alberghiero)
Servizio alberghiero(nome, descrizione)
Fermata(indirizzo luogo, giornata, itinerario, orario arrivo, orario partenza, giornata partenza)
Foto(nome file, nome località, regione località, pullman)
Autista(telefono, nome, cognome)
Pullman(targa, contatore, data immatricolazione, costruttore, nome modello)
Trasporto(pullman, autista, data partenza, itinerario)
Modello pullman(costruttore, nome, velocita max, cavalli, serbatoio, numero di posti)
Meccanico(tel, nome, cognome)
Allestimento(costruttore, nome modello, comfort)
eseguito(meccanico, data sessione, pullman)
Comfort(nome, descrizione)
manutenzione(pullman, controllo ordinario, km prox controllo)
Competenze(meccanico, nome modello , costruttore)
Controllo ordinario(tipo, intervallo)
Tipo sessione(controllo ordinario, data sessione, pullman)
Sessione controllo(data inizio, pullman, data fine, motivo)
Sostituzione(ricambio, data sessione, pullman)
Ricambio(codice, nome, disponibile)

Normalizzazione del modello relazionale

Cosi come proposto lo schema è già in forma 1NF,2NF e 3NF.

La tabella prenotazione stanza si potrebbe pensare che non sia in forma 2NF dato che il costo potrebbe dipendere solo dall'albergo e dalla tipologia di stanza. Consideriamo però il costo variabile a seconda della stagione e sono le chiavi di viaggio a definire in che periodo viene prenotata la stanza.

5. Progettazione fisica

Ai fini del client ho aggiunto una tabella utenti contenente I dati di accesso per ogni utente. Ho aggiunto alle tabelle ‘passeggero’, ‘meccanico’, ‘hostess’, ‘autista’ un attributo username in vincolo di integrità referenziale con username nella tabella ‘utenti’.

Utenti e privilegi

Meccanico

- Execute lettura_contatore
- Execute prossima_revisione
- Execute inserisci_sessione_controllo
- Execute ricambi_disponibili
- Execute usa_ricambio
- Execute revisione_da_effettuare
- Execute ordina_ricambio

Autista

- Execute orario_fermate
- Execute viaggio_assegnato_autista
- Execute info_visite_autista
- Execute calcola_distanza
- Execute cartine_programma
- Execute info_alberghi

Hostess

- Execute inserisci_passeggero
- Execute prenota_stanza
- Execute viaggio_assegnato_hostess
- Execute conferma_viaggio
- Execute inserisci_cliente
- Execute inserisci_costo

Login

- Execute login
- Execute lista_viaggi

Amministratore

- Execute crea_utente
- Execute aggiungi_employee

Passeggero

- Execute stato_viaggio
- Execute info_visite_passeggero

Strutture di memorizzazione

Tabella <programma>		
Attributo	**Tipo di dato**	**Attributi[2]**
nome	VARCHAR (45)	PK, NN
quota	FLOAT	NN, UN
Minimo_partecipanti	INT	UN, NN
Tabella <Viaggio>		
Attributo	**Tipo di dato**	**Attributi[3]**
Data_partenza	DATE	PK, NN
Itinerario	VARCHAR (45)	PK, NN
Numero_partecipanti	INT	NN, UN
Ritorno	DATE	NN

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

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

Costo_adulti	FLOAT	NN, UN
Costo_bambini	FLOAT	NN, UN
Data_cancellazione	DATE	
Stato	ENUM(...)	
Hostess	VARCHAR (15)	
Tabella <Hostess>		
Attributo	**Tipo di dato**	**Attributi[4]**
Telefono	VARCHAR (15)	PK, NN
Nome	VARCHAR (45)	NN
Cognome	VARCHAR (45)	NN
Username	VARCHAR (45)	NN
Tabella <Cliente>		
Attributo	**Tipo di dato**	**Attributi[5]**
telefono	VARCHAR (15)	PK, NN
Indirizzo	VARCHAR (45)	NN
nome	VARCHAR (45)	NN
email	VARCHAR (45)	NN
Fax	VARCHAR (15)	NN
Tabella <Passeggero>		
Attributo	**Tipo di dato**	**Attributi[6]**
Prenotatore	VARCHAR (15)	NN
Nome	VARCHAR (45)	NN
Età	VARCHAR (2)	NN
Posto	VARCHAR (3)	PK, NN

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

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

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

Data_viaggio	DATE	PK, NN
Itinrario	VARCHAR(45)	PK, NN
Data_prenotazione	DATE	NN
Data_invio_documenti	DATE	
Username	VARCHAR (45)	NN
Tabella <Descrizione>		
Attributo	**Tipo di dato**	**Attributi[7]**
Nome_località	VARCHAR (45)	PK, NN
Regione_località	VARCHAR (45)	PK, NN
Itinerario	VARCHAR (45)	PK, NN
Giornata_arrivo	INT	NN, UN
Giornata_partenza	INT	NN, UN
Tabella <Località>		
Attributo	**Tipo di dato**	**Attributi[8]**
Regione	VARCHAR (45)	PK, NN
Nome	VARCHAR (45)	PK, NN
Stato	VARCHAR (45)	NN
Tabella <Cartina>		
Attributo	**Tipo di dato**	**Attributi[9]**
Zona	ENUM(...)	PK, NN
Nome_località	VARCHAR (45)	PK, NN
Regione_località	VARCHAR (45)	PK, NN
Livello_di_dettaglio	ENUM(...)	NN
file	VARCHAR (45)	NN

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

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

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

Tabella <Luogo>		
Attributo	**Tipo di dato**	**Attributi[10]**
Indirizzo	VARCHAR (45)	PK, NN
Nome	VARCHAR (45)	NN
Nome_località	VARCHAR (45)	NN
Regione_località	VARCHAR (45)	NN
Tabella <Distanza>		
Attributo	**Tipo di dato**	**Attributi[11]**
Indirizzo_primo_luogo	VARCHAR (45)	PK, NN
Indirizzo_secondo_luogo	VARCHAR (45)	PK, NN
Metri	VARCHAR (10)	NN
Tabella <Albergo>		
Attributo	**Tipo di dato**	**Attributi[12]**
Indirizzo	VARCHAR (45)	PK, NN
Email	VARCHAR (254)	NN
Fax	VARCHAR (15)	NN
Categoria	ENUM (...)	NN
Telefono	VARCHAR (15)	NN
Tabella <Prenotazione_stanza>		
Attributo	**Tipo di dato**	**Attributi[13]**
Indirizzo_albergo	VARCHAR (45)	PK, NN
Data_partenza	DATE	PK, NN

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

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

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

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

Itinerario	VARCHAR (45)	PK, NN
Tipo_stanza	ENUM (...)	NN
costo	FLOAT	NN, UN
Numero_stanze_prenotate	INT	NN
Tabella <Bene>		
Attributo	**Tipo di dato**	**Attributi[14]**
Indirzzo_bene	VARCHAR (45)	PK, NN
Nome_bene	VARCHAR (45)	NN
Oario_apertura	TIME	NN
telefono	VARCHAR (15)	NN
Tabella <Visita>		
Attributo	**Tipo di dato**	**Attributi[15]**
itinerario	VARCHAR (45)	PK, NN
Indirizzo_bene	VARCHAR (45)	PK, NN
Nome_bene	VARCHAR (45)	NN
Compresa	ENUM(...)	NN
Guidata	ENUM(...)	NN
giornata	INT	NN
Tabella <Offre>		
Attributo	**Tipo di dato**	**Attributi[16]**
Indirizzo_albergo	VARCHAR (45)	PK, NN
Servizio_alberghiero	VARCHAR (45)	PK, NN
Tabella <Servizio_alberghiero>		

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

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

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

Attributo	Tipo di dato	Attributi[17]
Nome	VARCHAR (45)	PK, NN
Descrizione	VARCHAR (100)	NN
Tabella <Fermata>		
Attributo	**Tipo di dato**	**Attributi[18]**
Indirizzo_luogo	VARCHAR (45)	PK, NN
Itininerario	VARCHAR (45)	PK, NN
Giornata	INT	PK, NN, UN
Orario_partenza	TIME	NN
Orario_arrivo	TIME	NN
Giornata_partenza	INT	NN
Tabella <Foto>		
Attributo	**Tipo di dato**	**Attributi[19]**
Nome_file	VARCHAR (45)	PK, NN
Indirizzo_luogo	VARCHAR (45)	
Nome_luogo	VARCHAR (45)	
pullman	VARCHAR (7)	
Tabella <Autista>		
Attributo	**Tipo di dato**	**Attributi[20]**
Telefono	VARCHAR (15)	PK, NN
Nome	VARCHAR (45)	NN
Cognome	VARCHAR (45)	NN

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

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

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

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

Username	VARCHAR (45)	NN
Tabella <Pullman>		
Attributo	**Tipo di dato**	**Attributi[21]**
Targa	VARCHAR (7)	PK, NN
Contatore	INT	UN, NN
Data_immatricolazione	DATE	NN
costruttore	VARCHAR (45)	NN
Nome_modello	VARCHAR (45)	NN
Tabella <Trasporto>		
Attributo	**Tipo di dato**	**Attributi[22]**
Autista	VARCHAR (45)	PK, NN
Pullman	VARCHAR (7)	PK, NN
Data_partenza	DATE	PK, NN
Itinerario	VARCHAR (45)	PK, NN
Tabella <Modello_pullman>		
Attributo	**Tipo di dato**	**Attributi[23]**
Costruttore	VARCHAR (45)	PK, NN
Nome	VARCHAR (45)	PK, NN
Velocità_max	VARCHAR (3)	NN
Serbatoio	VARCHAR (3)	NN
Cavalli	VARCHAR (3)	NN
Numero_di_posti	INT	NN, UN
Tabella <Manutenzione>		

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

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

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

Attributo	Tipo di dato	Attributi[24]
pullman	VARCHAR (7)	PK ,NN
Controllo_ordinario	ENUM (...)	PK, NN
prossimo_controllo	INT	UN, NN
Tabella <Controllo_ordinario>		
Attributo	**Tipo di dato**	**Attributi[25]**
tipo	ENUM (...)	PK, NN
intervallo	INT	UN, NN
Tabella <Tipo sessione>		
Attributo	**Tipo di dato**	**Attributi[26]**
Controllo_ordinario	ENUM (..)	PK, NN
Data_controllo	DATE	PK, NN
Pullman	VARCHAR (7)	PK, NN
Tabella <Sessione_controllo>		
Attributo	**Tipo di dato**	**Attributi[27]**
Data_inizio	DATE	PK, NN
Pullman	VARCHAR (7)	PK, NN
Data_fine	DATE	NN
Motivo	VARCHAR (100)	NN
Tipo	ENUM (...)	NN
Tabella <Sostituzione>		

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

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

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

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

Attributo	Tipo di dato	Attributi[28]
Data_controllo	DATE	PK, NN
pullman	VARCHAR (7)	PK, NN
Ricambio	VARCHAR (45)	PK, NN
Tabella <Ricambio>		
Attributo	**Tipo di dato**	**Attributi[29]**
Codice	VARCHAR (45)	PK, NN
Nome	VARCHAR (45)	NN
Disponibilità	INT	NN, UN
Ordinato	INT	NN, UN
Tabella <Meccanico>		
Attributo	**Tipo di dato**	**Attributi[30]**
Telefono	VARCHAR (15)	PK, NN
Nome	VARCHAR (45)	NN
Cognome	VARCHAR (45)	NN
Username	VARCHAR (45)	NN
Tabella <Eseguito>		
Attributo	**Tipo di dato**	**Attributi[31]**
Meccanico	VARCHAR (15)	PK, NN
Data_controllo	DATE	PK, NN
Pullman	VARCHAR (7)	PK, NN
Tabella <Competenze>		

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

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

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

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

Attributo	Tipo di dato	Attributi[32]
costruttore	VARCHAR (45)	PK, NN
Nome_modello	VARCHAR (45)	PK, NN
Meccanico	VARCHAR (15)	PK, NN
Tabella <Allestimento>		
Attributo	**Tipo di dato**	**Attributi[33]**
Comfort	VARCHAR (45)	PK, NN
Costruttore	VARCHAR (45)	PK, NN
Nome_modello	VARCHAR (15)	PK, NN
Tabella <Comfort>		
Attributo	**Tipo di dato**	**Attributi[34]**
Nome	VARCHAR (45)	PK, NN
Descrizione	VARCHAR (100)	PK, NN
Tabella <Utenti>		
Attributo	**Tipo di dato**	**Attributi[35]**
Username	VARCHAR (45)	PK, NN
Password	CHAR (32)	PK, NN
Ruolo	ENUM(...)	PK, NN

Indici

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

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

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

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

Tabella <albergo>	
Indice <PRIMARY>	**Tipo[36]:**
indirizzo	PR
Indice < Luogo_albergo_idx >	**Tipo[37]:**
indirizzo	IDX
Tabella <allestimento>	
Indice <PRIMARY>	**Tipo[38]:**
comfort	PR
costruttore	PR
Nome_modello	PR
Indice <modello_allestimento_idx>	**Tipo[39]:**
costruttore	IDX
Nome_modello	IDX
Tabella <autista>	
Indice <PRIMARY>	**Tipo[40]:**
telefono	PR
Indice <user_idx>	**Tipo[41]:**
username	IDX
Tabella <bene>	
Indice <PRIMARY>	**Tipo[42]:**

[36] IDX = index, UQ = unique, FT = full text, PR = primary.

[37] IDX = index, UQ = unique, FT = full text, PR = primary.

[38] IDX = index, UQ = unique, FT = full text, PR = primary.

[39] IDX = index, UQ = unique, FT = full text, PR = primary.

[40] IDX = index, UQ = unique, FT = full text, PR = primary.

[41] IDX = index, UQ = unique, FT = full text, PR = primary.

[42] IDX = index, UQ = unique, FT = full text, PR = primary.

Indirizzo_bene	PR
Indice <luogo_idx >	**Tipo[43]:**
Indirizzo_bene	IDX
Nome_bene	IDX
Tabella <cartina>	
Indice <PRIMARY>	**Tipo[44]:**
Nome_località	PR
Regione_località	PR
zona	PR
Indice <località_idx>	**Tipo[45]:**
Nome_località	IDX
Regione_località	IDX
Tabella <cliente>	
Indice <PRIMARY>	**Tipo[46]:**
telefono	PR
Tabella <comfort>	
Indice <PRIMARY>	**Tipo[47]:**
nome	PR
Tabella <competenze>	
Indice <PRIMARY>	**Tipo[48]:**
costruttore	PR
Nome_modello	PR

[43] IDX = index, UQ = unique, FT = full text, PR = primary.

[44] IDX = index, UQ = unique, FT = full text, PR = primary.

[45] IDX = index, UQ = unique, FT = full text, PR = primary.

[46] IDX = index, UQ = unique, FT = full text, PR = primary.

[47] IDX = index, UQ = unique, FT = full text, PR = primary.

[48] IDX = index, UQ = unique, FT = full text, PR = primary.

meccanico	PR
Indice <modello_pullman_idx>	**Tipo[49]:**
costruttore	**IDX**
Nome_modello	**IDX**
Indice <meccanico_idx >	**Tipo[50]:**
meccanico	**IDX**
Tabella <controllo_ordinario>	
Indice <PRIMARY>	**Tipo[51]:**
tipo	PR
Tabella <descrizione>	
Indice <PRIMARY>	**Tipo[52]:**
Nome_località	PR
Regione_località	PR
itinerario	PR
Indice <località_descrizione_idx>	**Tipo[53]:**
Nome_località	**IDX**
Regione_località	**IDX**
Indice <itinerario_descrizione_idx >	**Tipo[54]:**
meccanico	**IDX**
Tabella <distanza>	
Indice <PRIMARY>	**Tipo[55]:**

[49] IDX = index, UQ = unique, FT = full text, PR = primary.

[50] IDX = index, UQ = unique, FT = full text, PR = primary.

[51] IDX = index, UQ = unique, FT = full text, PR = primary.

[52] IDX = index, UQ = unique, FT = full text, PR = primary.

[53] IDX = index, UQ = unique, FT = full text, PR = primary.

[54] IDX = index, UQ = unique, FT = full text, PR = primary.

[55] IDX = index, UQ = unique, FT = full text, PR = primary.

Primo_indirizzo	PR
Secondo_indirizzo	PR
Indice <primo_luogo_idx>	**Tipo[56]:**
Primo_indirizzo	**IDX**
Indice <secondo_luogo_idx >	**Tipo[57]:**
Secondo_indirizzo	**IDX**
Tabella <eseguito>	
Indice <PRIMARY>	**Tipo[58]:**
meccanico	PR
Data_controllo	PR
pullman	PR
Indice <sesssione_controllo_idx>	**Tipo[59]:**
Data_controllo	IDX
pullman	IDX
Indice <meccanico_controllo_idx>	**Tipo[60]:**
meccanico	IDX
Tabella <fermata>	
Indice <PRIMARY>	**Tipo[61]:**
Indirizzo_luogo	PR
giornata	PR
itinerario	PR

[56] IDX = index, UQ = unique, FT = full text, PR = primary.

[57] IDX = index, UQ = unique, FT = full text, PR = primary.

[58] IDX = index, UQ = unique, FT = full text, PR = primary.

[59] IDX = index, UQ = unique, FT = full text, PR = primary.

[60] IDX = index, UQ = unique, FT = full text, PR = primary.

[61] IDX = index, UQ = unique, FT = full text, PR = primary.

Indice <itinerario_fermata _idx>	**Tipo[62]:**
itinerario	IDX
Indice <luogo_fermata _idx>	**Tipo[63]:**
Indirizzo_luogo	IDX
Tabella <foto>	
Indice <PRIMARY>	**Tipo[64]:**
Nome_file	PR
Indice <località_foto_idx>	**Tipo[65]:**
Nome_località	IDX
Regione_località	IDX
Indice <pullman_foto_idx>	**Tipo[66]:**
pullman	IDX
Tabella <hostess>	
Indice <PRIMARY>	**Tipo[67]:**
telefono	PR
Indice <user_idx>	**Tipo[68]:**
username	IDX
Tabella <località>	
Indice <PRIMARY>	**Tipo[69]:**
Nome	PR

[62] IDX = index, UQ = unique, FT = full text, PR = primary.

[63] IDX = index, UQ = unique, FT = full text, PR = primary.

[64] IDX = index, UQ = unique, FT = full text, PR = primary.

[65] IDX = index, UQ = unique, FT = full text, PR = primary.

[66] IDX = index, UQ = unique, FT = full text, PR = primary.

[67] IDX = index, UQ = unique, FT = full text, PR = primary.

[68] IDX = index, UQ = unique, FT = full text, PR = primary.

[69] IDX = index, UQ = unique, FT = full text, PR = primary.

regione	PR
Tabella <luogo>	
Indice <PRIMARY>	**Tipo[70]:**
Indirizzo	PR
Indice <località_idx>	**Tipo[71]:**
Nome_località	IDX
Regione_località	IDX
Indice <bene_idx>	**Tipo[72]:**
Indirizzo	IDX
bene	IDX
Tabella <manutenzione>	
Indice <PRIMARY>	**Tipo[73]:**
pullman	PR
Controllo_ordinario	PR
Indice <controllo_idx>	**Tipo[74]:**
Controllo_ordinario	IDX
Indice <pullman_manutenzione_idx>	**Tipo[75]:**
pullman	IDX
Tabella <meccanico>	
Indice <PRIMARY>	**Tipo[76]:**
telefono	PR

[70] IDX = index, UQ = unique, FT = full text, PR = primary.

[71] IDX = index, UQ = unique, FT = full text, PR = primary.

[72] IDX = index, UQ = unique, FT = full text, PR = primary.

[73] IDX = index, UQ = unique, FT = full text, PR = primary.

[74] IDX = index, UQ = unique, FT = full text, PR = primary.

[75] IDX = index, UQ = unique, FT = full text, PR = primary.

[76] IDX = index, UQ = unique, FT = full text, PR = primary.

Indice <user_idx>	**Tipo[77]:**
username	IDX
Tabella <modello_pullman>	
Indice <PRIMARY>	**Tipo[78]:**
costruttore	PR
nome	PR
Tabella <offre>	
Indice <PRIMARY>	**Tipo[79]:**
Indirizzo_albergo	PR
Servizio_alberghiero	PR
Indice <albergo_idx>	**Tipo[80]:**
Indirizzo_albergo	IDX
Indice <servizio_idx>	**Tipo[81]:**
Servizio_alberghiero	IDX
Tabella <offre>	
Indice <PRIMARY>	**Tipo[82]:**
posto	PR
Data_viaggio	PR
itinerario	PR
Indice <prenotatore_idx>	**Tipo[83]:**
prenotatore	IDX

[77] IDX = index, UQ = unique, FT = full text, PR = primary.

[78] IDX = index, UQ = unique, FT = full text, PR = primary.

[79] IDX = index, UQ = unique, FT = full text, PR = primary.

[80] IDX = index, UQ = unique, FT = full text, PR = primary.

[81] IDX = index, UQ = unique, FT = full text, PR = primary.

[82] IDX = index, UQ = unique, FT = full text, PR = primary.

[83] IDX = index, UQ = unique, FT = full text, PR = primary.

Indice <user_idx>	**Tipo[84]:**
username	IDX
Indice <viaggio_passeggero_idx>	**Tipo[85]:**
Data_viaggio	IDX
itinerario	IDX
Tabella <prenotazione_stanza>	
Indice <PRIMARY>	**Tipo[86]:**
Indirizzo_albergo	PR
Data_viaggio	PR
itinerario	PR
Tipo_stanza	PR
Indice <viaggio_idx>	**Tipo[87]:**
Data_viaggio	IDX
itinerario	IDX
Indice <albergo_idx>	**Tipo[88]:**
Indirizzo_albergo	IDX
Tabella <programma>	
Indice <PRIMARY>	**Tipo[89]:**
nome	PR
Tabella <pullman >	
Indice <PRIMARY>	**Tipo[90]:**

[84] IDX = index, UQ = unique, FT = full text, PR = primary.

[85] IDX = index, UQ = unique, FT = full text, PR = primary.

[86] IDX = index, UQ = unique, FT = full text, PR = primary.

[87] IDX = index, UQ = unique, FT = full text, PR = primary.

[88] IDX = index, UQ = unique, FT = full text, PR = primary.

[89] IDX = index, UQ = unique, FT = full text, PR = primary.

[90] IDX = index, UQ = unique, FT = full text, PR = primary.

targa	PR
Indice <modello_idx>	**Tipo[91]:**
costruttore	IDX
Nome_modello	IDX
Tabella <ricambio>	
Indice <PRIMARY>	**Tipo[92]:**
codice	PR
Tabella <servizio_alberghiero>	
Indice <PRIMARY>	**Tipo[93]:**
nome	PR
Tabella <sessione_controllo>	
Indice <PRIMARY>	**Tipo[94]:**
pullman	PR
Data_inizio	PR
Indice <pullman_idx>	**Tipo[95]:**
pullman	IDX
Tabella <sostituzione>	
Indice <PRIMARY>	**Tipo[96]:**
Data_controllo	PR
pullman	PR
ricambio	PR

[91] IDX = index, UQ = unique, FT = full text, PR = primary.

[92] IDX = index, UQ = unique, FT = full text, PR = primary.

[93] IDX = index, UQ = unique, FT = full text, PR = primary.

[94] IDX = index, UQ = unique, FT = full text, PR = primary.

[95] IDX = index, UQ = unique, FT = full text, PR = primary.

[96] IDX = index, UQ = unique, FT = full text, PR = primary.

Indice <sessione_controllo_idx>	**Tipo[97]:**
Data_controllo	IDX
pullman	IDX
Indice <ricambio_idx>	**Tipo[98]:**
codice	IDX
Tabella <tipo_sessione>	
Indice <PRIMARY>	**Tipo[99]:**
pullman	PR
Data_controllo	PR
Indice <sessione_tipo_idx>	**Tipo[100]:**
pullman	IDX
Data_controllo	IDX
Indice <controllo_tipo_idx>	**Tipo[101]:**
Controllo_ordinario	IDX
Tabella <trasporto>	
Indice <PRIMARY>	**Tipo[102]:**
autista	PR
Data_partenza	PR
itinerario	PR
pullman	PR
Indice <viaggio_idx>	**Tipo[103]:**

[97] IDX = index, UQ = unique, FT = full text, PR = primary.

[98] IDX = index, UQ = unique, FT = full text, PR = primary.

[99] IDX = index, UQ = unique, FT = full text, PR = primary.

[100] IDX = index, UQ = unique, FT = full text, PR = primary.

[101] IDX = index, UQ = unique, FT = full text, PR = primary.

[102] IDX = index, UQ = unique, FT = full text, PR = primary.

[103] IDX = index, UQ = unique, FT = full text, PR = primary.

Data_partenza	IDX
itinerario	IDX
Indice <autista_trasporto_idx>	**Tipo[104]:**
autista	IDX
Indice <pullman_idx>	**Tipo[105]:**
pullman	IDX
Tabella <utenti>	
Indice <PRIMARY>	**Tipo[106]:**
username	PR
Tabella <viaggio >	
Indice <PRIMARY>	**Tipo[107]:**
Data_partenza	PR
itinerario	PR
Indice <itinerario_viaggio_idx>	**Tipo[108]:**
itinerario	IDX
Indice <hostess_idx>	**Tipo[109]:**
hostess	IDX
Tabella <visita >	
Indice <PRIMARY>	**Tipo[110]:**
Indirizzo_bene	PR
itinerario	PR

[104] IDX = index, UQ = unique, FT = full text, PR = primary.

[105] IDX = index, UQ = unique, FT = full text, PR = primary.

[106] IDX = index, UQ = unique, FT = full text, PR = primary.

[107] IDX = index, UQ = unique, FT = full text, PR = primary.

[108] IDX = index, UQ = unique, FT = full text, PR = primary.

[109] IDX = index, UQ = unique, FT = full text, PR = primary.

[110] IDX = index, UQ = unique, FT = full text, PR = primary.

Indice <itinerario_visita_idx>	**Tipo[111]:**
itinerario	IDX

Trigger

```
CREATE DEFINER = CURRENT_USER TRIGGER `mydb`.`viaggio_BEFORE_UPDATE`
BEFORE UPDATE ON `viaggio` FOR EACH ROW
BEGIN
declare counter INT;
select count(*) from `viaggio` V
where
NEW.hostess = V.hostess
and NEW.data_partenza < V.ritorno
   and NEW.ritorno > V.data_partenza into counter;
if counter > 0 then
signal sqlstate "45000";
end if;
END
```

```
CREATE DEFINER = CURRENT_USER TRIGGER `mydb`.`viaggio_BEFORE_UPDATE_1`
BEFORE UPDATE ON `viaggio` FOR EACH ROW
BEGIN
declare counter INT;
select count(*) from `viaggio` V
```

[111] IDX = index, UQ = unique, FT = full text, PR = primary.

```
where NEW.stato = 'cancellato' and NEW.hostess is not null or NEW.stato = 'confermato' and
NEW.data_cancellazione is not null into counter;


if counter <> 0 then
signal sqlstate "45000";
end if;
END



CREATE DEFINER = CURRENT_USER TRIGGER `mydb`.`viaggio_BEFORE_UPDATE_2`
BEFORE UPDATE ON `viaggio` FOR EACH ROW
BEGIN
declare counter INT;
if NEW.stato = 'confermato'  then
select count(*) from `programma` P
where
P.nome = NEW.itinerario
and P.minimo_partecipanti > NEW.numero_partecipanti into counter;



if counter <> 0 then
signal sqlstate "45000";
end if;
end if;
END



CREATE DEFINER = CURRENT_USER TRIGGER `mydb`.`distanza_BEFORE_INSERT`
BEFORE INSERT ON `distanza` FOR EACH ROW
BEGIN
if NEW.primo_indirizzo = NEW.secondo_indirizzo then
signal sqlstate "45000";
   end if;
```

```
END



CREATE DEFINER = CURRENT_USER TRIGGER `mydb`.`distanza_BEFORE_INSERT_1`
BEFORE INSERT ON `distanza` FOR EACH ROW
BEGIN

   declare counter int;


   select count(*) from `distanza` D
   where
NEW.primo_indirizzo = D.secondo_indirizzo
and D.primo_indirizzo = NEW.secondo_indirizzo into counter;


if counter > 0 then
signal sqlstate "45000";
end if;

END



CREATE DEFINER = CURRENT_USER TRIGGER `mydb`.`visita_BEFORE_INSERT` BEFORE
INSERT ON `visita` FOR EACH ROW
BEGIN
declare counter INT;
SELECT
   COUNT(*)
FROM
   `descrizione` D,
   `luogo` L
WHERE
   L.indirizzo = NEW.indirizzo_bene
AND L.nome_località = D.nome_località
```

```
AND L.regione_località = D.regione_località
AND NEW.giornata >= D.giornata_arrivo
AND NEW.giornata <= giornata_partenza INTO counter;
if counter = 0 then
signal sqlstate "45000";
end if;
END


CREATE DEFINER = CURRENT_USER TRIGGER `mydb`.`fermata_BEFORE_INSERT`
BEFORE INSERT ON `fermata` FOR EACH ROW
BEGIN
declare counter INT;
SELECT
   COUNT(*)
FROM
   `albergo` A
WHERE
   A.indirizzo = NEW.indirizzo_luogo INTO counter;

if counter > 1 then

   select count(*) from `descrizione` D, `luogo` L
   where D.itinerario = NEW.itinerario
   and D.nome_località = L.nome_località
   and D.regione_località = L.regione_località
   and NEW.indirizzo_luogo = L.indirizzo
   and D.giornata_arrivo <> D.giornata_partenza into counter;

   if counter = 0 then
signal sqlstate "45000";
end if;
```

```
end if;
END
```

```
CREATE DEFINER = CURRENT_USER TRIGGER `mydb`.`fermata_BEFORE_INSERT_1`
BEFORE INSERT ON `fermata` FOR EACH ROW
BEGIN

declare counter int;

select count(*) from `albergo` A
   where
new.indirizzo_luogo = A.indirizzo into counter;

   if counter = 0 then
if new.giornata <> new.giornata_partenza then
signal sqlstate "45000";
      end if;
end if;
END
```

```
DROP TRIGGER IF EXISTS `mydb`.`foto_BEFORE_INSERT` $$
USE `mydb`$$
CREATE DEFINER = CURRENT_USER TRIGGER `mydb`.`foto_BEFORE_INSERT` BEFORE
INSERT ON `foto` FOR EACH ROW
BEGIN
if new.indirizzo_luogo is not null and new.pullman is not null then
signal sqlstate "45000" set message_text = 'la foto puo raffigurare sia un pullamn, sia una località;
uno dei due valori deve essere null';
   end if;
END
```

```
CREATE DEFINER = CURRENT_USER TRIGGER `mydb`.`trasporto_BEFORE_INSERT`
BEFORE INSERT ON `trasporto` FOR EACH ROW
BEGIN
declare n_autisti INT;
declare n_pullman INT;

select count(`autista`) from `trasporto` T
where
T.itinerario = NEW.itinerario
and T.data_partenza = NEW.data_partenza into n_autisti;

select count(`pullman`) from `trasporto` T
where
T.itinerario = NEW.itinerario
and T.data_partenza = NEW.data_partenza into n_pullman;

if n_autisti <> n_pullman then
signal sqlstate "45000";
end if;
END
```

```
CREATE DEFINER = CURRENT_USER TRIGGER `mydb`.`trasporto_BEFORE_INSERT_1`
BEFORE INSERT ON `trasporto` FOR EACH ROW
BEGIN
declare n_autisti INT;
declare rit DATE;

select `ritorno` from `viaggio` V
```

```
where
NEW.itinerario = V.itinerario
and NEW.data_partenza = V.data_partenza into rit;

select count(`autista`) from `trasporto` T, `viaggio` V
where
T.itinerario = V.itinerario
and T.data_partenza = V.data_partenza
   and T.autista = NEW.autista
   and NEW.data_partenza < V.ritorno
   and rit > V.data_partenza into n_autisti;


if n_autisti <> 0 then
signal sqlstate "45000" set message_text = 'le date dei viaggi si sovrappongono per l autista';
end if;
END



CREATE DEFINER = CURRENT_USER TRIGGER `mydb`.`trasporto_BEFORE_INSERT_2`
BEFORE INSERT ON `trasporto` FOR EACH ROW
BEGIN
declare n_pullman INT;
declare rit DATE;

   select `ritorno` from `viaggio` V
where
NEW.itinerario = V.itinerario
and NEW.data_partenza = V.data_partenza into rit;

select count(`pullman`) from `trasporto` T, `viaggio` V
```

```
where
T.itinerario = V.itinerario
and T.data_partenza = V.data_partenza
and T.pullman = NEW.pullman
and NEW.data_partenza < V.ritorno
and rit > V.data_partenza into n_pullman;

if n_pullman <> 0 then
signal sqlstate "45000" set message_text = 'le date dei viaggi si sovrappongono per il pullman';
end if;
END
```

```
CREATE DEFINER = CURRENT_USER TRIGGER
`mydb`.`sessione_controllo_BEFORE_INSERT` BEFORE INSERT ON `sessione_controllo` FOR
EACH ROW
BEGIN
declare counter1 INT;
declare counter2 INT;
if NEW.motivo = NULL then

SELECT
`contatore`
FROM
`pullman` P
WHERE
P.targa = NEW.pullman INTO counter1;

SELECT
M.prossimo_controllo
FROM
`manutenzione` M,
```

```
`tipo_sessione` T
WHERE
T.data_controllo = NEW.data_inizio
AND T.pullman = NEW.pullman
AND T.controllo_ordinario = M.controllo_ordinario INTO counter2;

if counter1 < counter2 then
signal sqlstate "45000";
end if;
end if;
END
```

Eventi

Viste

Stored Procedures e transazioni

```
-- -----------------------------------------------------
-- procedure orario_fermate
-- -----------------------------------------------------

USE `mydb`;
DROP procedure IF EXISTS `mydb`.`orario_fermate`;

DELIMITER $$
USE `mydb`$$
CREATE PROCEDURE `orario_fermate` (in itin varchar(45), in giorn INT)
```

```
BEGIN
	select
		`indirizzo_luogo`, `nome`, `orario_arrivo`, `orario_partenza` from `fermata` F, `luogo` L
	where
			F.`itinerario` = itin
		and F.indirizzo_luogo = L.indirizzo
		and F.`giornata` = giorn;
END

-- -----------------------------------------------------
-- procedure lettura_contatore
-- -----------------------------------------------------

USE `mydb`;
DROP procedure IF EXISTS `mydb`.`lettura_contatore`;

DELIMITER $$
USE `mydb`$$
CREATE PROCEDURE `lettura_contatore` (in pull varchar(7))
BEGIN
	select
		`contatore` from `pullman` P
	where
		P.`targa` = pull;
```

```
END



-- -----------------------------------------------------

-- procedure prossima_revisione

-- -----------------------------------------------------


USE `mydb`;

DROP procedure IF EXISTS `mydb`.`prossima_revisione`;


DELIMITER $$

USE `mydb`$$

CREATE PROCEDURE `prossima_revisione` (in pull varchar(7), in contr ENUM('motore', 'convergenza', 'freni', 'pneumatici', 'luci', 'batteria', 'sospensioni', 'carrozzeria'))

BEGIN

	select `prossimo_Controllo` from `manutenzione` M

	where M.pullman = pull and M.controllo_ordinario = contr;

END



-- -----------------------------------------------------

-- procedure login

-- -----------------------------------------------------


USE `mydb`;
```

```
DROP procedure IF EXISTS `mydb`.`login`;


DELIMITER $$

USE `mydb`$$

CREATE PROCEDURE `login` (in var_username varchar(45), in var_pass varchar(45), out var_role
INT)

BEGIN

        declare var_user_role ENUM('autista', 'hostess', 'meccanico', 'passeggero', 'amministratore');


        select `ruolo` from `utenti`

                where `username` = var_username

           and `password` = md5(var_pass) into var_user_role;


    -- See the corresponding enum in the client

                if var_user_role = 'autista' then

                        set var_role = 1;

                elseif var_user_role = 'hostess' then

                        set var_role = 2;

                elseif var_user_role = 'meccanico' then

                        set var_role = 3;

                elseif var_user_role = 'passeggero' then

                        set var_role = 4;

                elseif var_user_role = 'amministratore' then

                        set var_role = 5;

                else
```

```
                set var_role = 6;
            end if;
END

-- -----------------------------------------------------
-- procedure aggiungi_employee
-- -----------------------------------------------------
```

Ho scelto read uncommitted dato che ero solo interessato ad avere atomicità.

```
USE `mydb`;
DROP procedure IF EXISTS `mydb`.`aggiungi_employee`;

DELIMITER $$
USE `mydb`$$
CREATE PROCEDURE `aggiungi_employee` (IN tel VARCHAR(15), IN nome VARCHAR(45), IN cognome VARCHAR(45), IN username VARCHAR(45), IN pass VARCHAR(45), IN ruolo int)
BEGIN

declare exit handler for sqlexception

begin
    rollback; -- rollback any changes made in the transaction
    resignal; -- raise again the sql exception to the caller
end;
```

```
set transaction isolation level read uncommitted;

start transaction;
	if ruolo = 0 then
		insert into utenti VALUES(username, MD5(pass), 'hostess');
		insert into `hostess` values(tel, nome, cognome, username);
	elseif ruolo = 1 then
		insert into utenti VALUES(username, MD5(pass), 'meccanico');
		insert into `meccanico` values(tel, nome, cognome, username);
	elseif ruolo = 2 then
		insert into utenti VALUES(username, MD5(pass), 'autista');
		insert into `autista` values(tel, nome, cognome, username);
	end if;
commit;



END


-- --------------------------------------------------
-- procedure crea_utente
-- --------------------------------------------------

USE `mydb`;
```

```
DROP procedure IF EXISTS `mydb`.`crea_utente`;


DELIMITER $$

USE `mydb`$$

CREATE PROCEDURE `crea_utente` (IN username VARCHAR(45), IN pass VARCHAR(45), IN ruolo varchar(45))

BEGIN

	insert into utenti VALUES(username, MD5(pass), ruolo);

END



-- -----------------------------------------------------

-- procedure inserisci_sessione_controllo

-- -----------------------------------------------------
```

non sono presenti letture ripetute e quindi posso scegliere da read committed e uncommitted. Ho bisogno però di andare a estrarre dati committati dalla base di dati, ad esempio pensiamo a due meccanici che contemporaneamente cercano di effettuare una sessione di controllo. Ho scelto quindi read committed

```
USE `mydb`;

DROP procedure IF EXISTS `mydb`.`inserisci_sessione_controllo`;


DELIMITER $$

USE `mydb`$$
```

```
CREATE PROCEDURE `inserisci_sessione_controllo` ( in pull varchar(7), in data_i date, in data_f date, in motivo text, in motore int, in convergenza int, in freni int, in pneumatici int, in luci int, in batteria int, in sospensioni int, in carrozzeria int, IN uname varchar(15))
BEGIN
    declare tel varchar(15);
  declare counter int;
  declare exit handler for sqlexception

    begin
        rollback; -- rollback any changes made in the transaction
        resignal; -- raise again the sql exception to the caller
    end;

    set transaction isolation level read committed;

    start transaction;

  select `telefono` from `meccanico` M
  where M.username = uname into tel;

  insert into `sessione_controllo` values(data_i, pull, data_f, motivo);

  insert into `eseguito` values(tel, data_i, pull);

  if motivo IS NULL then
```

```
if motore <> 1 then
                    insert into tipo_sessione values('motore', data_i, pull);

  select
                    `intervallo` from `controllo_ordinario` C
  where
                    C.tipo = 'motore' into counter;


  UPDATE
                    `manutenzione` M, `pullman` P SET M.prossimo_controllo=
P.contatore + counter
  WHERE
                    M.pullman = pullman
                  and M.controllo_ordinario = 'motore';
end if;

if convergenza <> 1 then

                    insert into tipo_sessione values('convergenza', data_i, pull);

  select
                    `intervallo` from `controllo_ordinario` C
  where
```

```
                C.tipo = 'convergenza' into counter;

        UPDATE
                `manutenzione` M, `pullman` P SET M.prossimo_controllo=
P.contatore + counter
        WHERE
                M.pullman = pullman
            and M.controllo_ordinario = 'convergenza';
    end if;

    if freni <> 1 then

            insert into tipo_sessione values('freni', data_i, pull);

        select
                `intervallo` from `controllo_ordinario` C
        where
                C.tipo = 'freni' into counter;

        UPDATE
                `manutenzione` M, `pullman` P SET M.prossimo_controllo =
P.contatore + counter
        WHERE
```

```
                    M.pullman = pullman
                and M.controllo_ordinario = 'freni';
    end if;

    if pneumatici <> 1 then

                insert into tipo_sessione values('pneumatici', data_i, pull);

      select
                    `intervallo` from `controllo_ordinario` C
      where
                    C.tipo = 'pneumatici' into counter;


      UPDATE
                    `manutenzione` M, `pullman` P SET M.prossimo_controllo=
P.contatore + counter
      WHERE
                    M.pullman = pullman
                and M.controllo_ordinario = 'pneumatici';
    end if;

    if luci <> 1 then

                insert into tipo_sessione values('luci', data_i, pull);
```

```
        select
                `intervallo` from `controllo_ordinario` C
        where
                C.tipo = 'luci' into counter;


        UPDATE
                `manutenzione` M, `pullman` P SET M.prossimo_controllo=
P.contatore + counter
        WHERE
                M.pullman = pullman
            and M.controllo_ordinario = 'luci';
    end if;

    if batteria <> 1 then

            insert into tipo_sessione values('batteria', data_i, pull);

        select
                `intervallo` from `controllo_ordinario` C
        where
                C.tipo = 'batteria' into counter;
```

```
        UPDATE
                `manutenzione` M, `pullman` P SET M.prossimo_controllo=
P.contatore + counter
        WHERE
                M.pullman = pullman
                and M.controllo_ordinario = 'batteria';
    end if;

    if sospensioni <> 1 then

            insert into tipo_sessione values('sospensioni', data_i, pull);

        select
                `intervallo` from `controllo_ordinario` C
        where
                C.tipo = 'sospensioni' into counter;


        UPDATE
                `manutenzione` M, `pullman` P SET M.prossimo_controllo=
P.contatore + counter
       WHERE
                M.pullman = pullman
            and M.controllo_ordinario = 'sospensioni';
    end if;
```

```
        if carrozzeria <> 1 then

                        insert into tipo_sessione values('carrozzeria', data_i, pull);

            select
                        `intervallo` from `controllo_ordinario` C
            where
                        C.tipo = 'carrozzeria' into counter;



            UPDATE
                        `manutenzione` M, `pullman` P SET M.prossimo_controllo=
P.contatore + counter
            WHERE
                        M.pullman = pullman
                    and M.controllo_ordinario = 'carrozzeria';
           end if;
    end if;
   commit;
END



-- -----------------------------------------------
-- procedure usa_ricambio
```

-- ----------

Ho scelto repeatable read dato che inizialmente estraggo il dato 'disponibile' e dopo una certa verifica ci riaccedo per fare una update. Ho bisogno quindi di una lock per evitare che degli aggiornamenti contemporanei vadano a far perdere coerenza alla base di dati

```
USE `mydb`;
DROP procedure IF EXISTS `mydb`.`usa_ricambio`;

DELIMITER $$
USE `mydb`$$
CREATE PROCEDURE `usa_ricambio` (in cod varchar(5), in inizio date, in pull varchar(7))
BEGIN

	declare d int;
	declare exit handler for sqlexception

	begin
		rollback; -- rollback any changes made in the transaction
		resignal; -- raise again the sql exception to the caller
	end;

	set transaction isolation level repeatable read;

	start transaction;
```

```
            select
                `disponibile` from `ricambio` R
            where
                R.codice = cod into d;

            if d > 0 then
                insert into `sostituzione` values(inizio, pull, cod);

                 update
                    `ricambio` R set `disponibile` = d - 1
                where
                    R.codice = cod;
            else
                signal sqlstate "45000";
        end if;
    commit;
END


-- -----------------------------------------------------
-- procedure viaggio_assegnato_hostess
-- -----------------------------------------------------

USE `mydb`;
```

```
DROP procedure IF EXISTS `mydb`.`viaggio_assegnato_hostess`;


DELIMITER $$
USE `mydb`$$
CREATE PROCEDURE `viaggio_assegnato_hostess` (in uname varchar(45))
BEGIN
	select `data_partenza`, `itinerario`, `numero_partecipanti`, `ritorno`, `stato` from `viaggio` V, `hostess` H
	where
		H.username = uname
	    and V.hostess = H.telefono;
END



-- -----------------------------------------------------
-- procedure calcola_distanza
-- -----------------------------------------------------

USE `mydb`;
DROP procedure IF EXISTS `mydb`.`calcola_distanza`;

DELIMITER $$
USE `mydb`$$
CREATE PROCEDURE `calcola_distanza` (in ind_1 varchar(45), in ind_2 varchar(45))
BEGIN
```

```
        select `metri` from `distanza` D
        where
                (D.primo_indirizzo = ind_1 and D.secondo_indirizzo = ind_2)
            or (D.secondo_indirizzo = ind_1 and D.primo_indirizzo = ind_2);
END



-- -----------------------------------------------------
-- procedure inserisci_cliente
-- -----------------------------------------------------

USE `mydb`;
DROP procedure IF EXISTS `mydb`.`inserisci_cliente`;

DELIMITER $$
USE `mydb`$$
CREATE PROCEDURE `inserisci_cliente` (in tel varchar(15), in ind varchar(45), in n varchar(45), in
mail varchar(45), in f varchar(15))
BEGIN
        insert into `cliente` values(tel, ind, n, mail, f);
END



-- -----------------------------------------------------
-- procedure inserisci_passeggero
```

-- ---

Utilizzo serializable dato che per verificare se il viaggio non è stato cancellato utilizzo l'operatore count. Ovviamente non posso utilizzare un livello piu basso di repeatable read dato che ho bisogno che duarnte l'esecuzione il viaggio non venga cancellato. Modificando la select e sfruttando il campo 'data_cancellazione' avrei potuto evitare serializable. Avrei potuto abbassare il livello di isolamento aggiungendo un trigger on insert alla tabella 'passeggero' facendo in modo che un passeggero non potesse essere associato ad un viaggio cancellato. Cosi facendo avrei potuto evitare di lockare la tabella viaggio, anche perché non effettuo nessuna lettura ripetuta. Sarebbe bastato read committed.

```
USE `mydb`;

DROP procedure IF EXISTS `mydb`.`inserisci_passeggero`;


DELIMITER $$

USE `mydb`$$

CREATE PROCEDURE `inserisci_passeggero` (in pren varchar(15), in n varchar(45), in e varchar(2), in p varchar(3), in uname varchar(45), in pass varchar(45), in dv date, in itin varchar(45), in data_pren date)

BEGIN


    declare counter int;

    declare exit handler for sqlexception


    begin

        rollback; -- rollback any changes made in the transaction

        resignal; -- raise again the sql exception to the caller
```

```
    end;

    set transaction isolation level serializable;

    start transaction;
        select count(*) from `viaggio` V
    where
                V.data_partenza = dv
            and V.itinerario = itin
            and V.stato = 'organizzazione' or V.stato = 'confermato' into counter;

  if counter <> 0 then
            insert into `utenti` values(uname, md5(pass), 'passeggero');

            insert into `passeggero` values(pren, n, e, p, uname, dv, itin, data_pren, NULL);

            update `viaggio` V
            set V.numero_partecipanti = V.numero_partecipanti + 1
            where
                    V.itinerario = itin
                and V.data_partenza = dv;
    end if;
commit;
```

```
END


-- -----------------------------------------------------
-- procedure prenota_stanza
-- -----------------------------------------------------
```

Utilizzo serializable dato che per verificare se il viaggio non è stato cancellato utilizzo l'operatore count. Ovviamente non posso utilizzare un livello piu basso di repeatable read dato che ho bisogno che duarnte l'esecuzione il viaggio non venga cancellato. Modificando la select e sfruttando il campo 'data_cancellazione' avrei potuto evitare serializable. Avrei potuto abbassare il livello di isolamento aggiungendo un trigger on update alla tabella 'prenotazione_stanza' facendo in modo che una stanza non possa essere prenotata per un viaggio cancellato. Cosi facendo avrei potuto evitare di lockare la tabella viaggio, anche perché non effettuo nessuna lettura ripetuta. Sarebbe bastato read committed.

```
USE `mydb`;
DROP procedure IF EXISTS `mydb`.`prenota_stanza`;

DELIMITER $$
USE `mydb`$$
CREATE PROCEDURE `prenota_stanza` (in al varchar(45), in dp date, in itin varchar(45), in tipo ENUM('singola', 'doppia', 'tripla', 'quadrupla', 'suite', 'doppia con 3° letto', 'doppia con 4° letto'))
BEGIN

    declare counter int;
    declare exit handler for sqlexception
```

```
    begin
            rollback; -- rollback any changes made in the transaction
            resignal; -- raise again the sql exception to the caller
    end;

    set transaction isolation level serializable;

    start transaction;
            select count(*) from `viaggio` V
           where
                        V.data_partenza = dp
                    and V.itinerario = itin
                    and V.stato = 'organizzazione' or V.stato = 'confermato' into counter;

if counter <> 0 then
                    update
                            `prenotazione_stanza` P
                    set
                            P.numero_stanze_prenotate = P.numero_stanze_prenotate + 1
                    where
                                P.indirizzo_albergo = al
                            and P.data_partenza = dp
                            and P.itinerario = itin
                            and P.tipo_stanza = tipo;
```

```
            end if;
        commit;
END


-- -----------------------------------------------------
-- procedure inserisci_costo
-- -----------------------------------------------------

USE `mydb`;
DROP procedure IF EXISTS `mydb`.`inserisci_costo`;

DELIMITER $$
USE `mydb`$$
CREATE PROCEDURE `inserisci_costo` (in al varchar(45), in dp date, in itin varchar(45), in tipo ENUM('singola', 'doppia', 'tripla', 'quadrupla', 'suite', 'doppia con 3° letto', 'doppia con 4° letto'), in c int)
BEGIN
            insert into `prenotazione_stanza` values(al, dp, itin, tipo, c, 0);
END


-- -----------------------------------------------------
-- procedure conferma_viaggio
-- -----------------------------------------------------
```

```
USE `mydb`;
DROP procedure IF EXISTS `mydb`.`conferma_viaggio`;

DELIMITER $$
USE `mydb`$$
CREATE PROCEDURE `conferma_viaggio` (in dp date , in itin varchar(45), in hos varchar(15))
BEGIN
	update `viaggio` V
	set V.stato = 'confermato', V.hostess = hos
	where
			V.data_partenza = dp
		and V.itinerario = itin;
END

-- -----------------------------------------------------
-- procedure stato_viaggio
-- -----------------------------------------------------

USE `mydb`;
DROP procedure IF EXISTS `mydb`.`stato_viaggio`;

DELIMITER $$
USE `mydb`$$
```

```
CREATE PROCEDURE `stato_viaggio` (in uname varchar(45))
BEGIN
	select V.itinerario, `data_partenza`, `stato`, `hostess` from `viaggio` V, `passeggero` P
	where
			P.username = uname
		and P.itinerario = V.itinerario
		and P.data_viaggio = V.data_partenza;
END



-- --------------------------------------------------
-- procedure info_visite_passeggero
-- --------------------------------------------------

USE `mydb`;
DROP procedure IF EXISTS `mydb`.`info_visite_passeggero`;

DELIMITER $$
USE `mydb`$$
CREATE PROCEDURE `info_visite_passeggero` (in uname varchar(45))
BEGIN
	select `nome_bene`, `compresa`, `guidata`, `giornata`, `orario` from `passeggero` P, `visita` V
	where
			P.username = uname
```

```
                and P.itinerario = V.itinerario;
END$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure info_visite_autista
-- -----------------------------------------------------

USE `mydb`;
DROP procedure IF EXISTS `mydb`.`info_visite_autista`;

DELIMITER $$
USE `mydb`$$
CREATE PROCEDURE `info_visite_autista` (itin varchar(45))
BEGIN
    select `nome_bene`, `giornata`, `orario` from  `visita` V
    where
        V.itinerario = itin;
END$$

DELIMITER ;

-- -----------------------------------------------------
```

```
-- procedure viaggio_assegnato_autista
-- -----------------------------------------------------

USE `mydb`;
DROP procedure IF EXISTS `mydb`.`viaggio_assegnato_autista`;

DELIMITER $$
USE `mydb`$$
CREATE PROCEDURE `viaggio_assegnato_autista` (in uname varchar(45))
BEGIN
	 select `itinerario`, `data_partenza`, `pullman` from `trasporto` T, `autista` A
	where
			A.username = uname
		and A.telefono = T.autista;
END$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure lista_viaggi
-- -----------------------------------------------------

USE `mydb`;
DROP procedure IF EXISTS `mydb`.`lista_viaggi`;
```

```
DELIMITER $$

USE `mydb`$$

CREATE PROCEDURE `lista_viaggi` ()

BEGIN

	select `itinerario`, `data_partenza`, `ritorno`, `numero_partecipanti`, `costo_adulti`, `costo_bambini`, `stato` from `viaggio` V

	where

		V.stato = 'organizzazione'

	or V.stato = 'confermato';

END$$


DELIMITER ;



-- -----------------------------------------------------

-- procedure ricambi_disponibili

-- -----------------------------------------------------


USE `mydb`;

DROP procedure IF EXISTS `mydb`.`ricambi_disponibili`;


DELIMITER $$

USE `mydb`$$

CREATE PROCEDURE `ricambi_disponibili` ()
```

```
BEGIN
	select * from `ricambio` R
	where R.disponibile > 0;
END$$

DELIMITER ;

-- ----------------------------------------------------
-- procedure ordina_ricambio
-- ----------------------------------------------------

USE `mydb`;
DROP procedure IF EXISTS `mydb`.`ordina_ricambio`;

DELIMITER $$
USE `mydb`$$
CREATE PROCEDURE `ordina_ricambio` (in cod varchar(5))
BEGIN
	update `ricambio` R set `ordinato` = 1
	where
		R.codice = cod;
END$$

DELIMITER ;
```

```
-- -----------------------------------------------------
-- procedure revisioni_da_effettuare
-- -----------------------------------------------------

USE `mydb`;
DROP procedure IF EXISTS `mydb`.`revisioni_da_effettuare`;

DELIMITER $$
USE `mydb`$$
CREATE PROCEDURE `revisioni_da_effettuare` ()
BEGIN
	select M.pullman, M.controllo_ordinario from `manutenzione` M, `pullman` P
	where P.contatore > M.prossimo_controllo and P.targa = M.pullman
	order by M.pullman;
END$$
DELIMITER ;


-- -----------------------------------------------------
-- procedure cartine_programma
-- -----------------------------------------------------
```

La select del cursore è la stessa select che faccio all'inizio della transazione (la utilizzo poi nel client per costruirmi l'header). Ho bisogno quindi di un livello uguale o superiore a repeatable read; non utilizzo operatori aggregati quindi mi basta repeatable read.

```
USE `mydb`;

DROP procedure IF EXISTS `mydb`.`cartine_programma`;


DELIMITER $$

USE `mydb`$$

CREATE PROCEDURE `cartine_programma` (in itin varchar(45))

BEGIN

	declare var_nome varchar(45);

	declare var_regione varchar(45);

	declare done int default false;

	declare cur cursor for select `nome_località`, `regione_località` from `descrizione` D where D.itinerario = itin;

  declare continue handler for not found set done = true;



	declare exit handler for sqlexception


	begin

		rollback; -- rollback any changes made in the transaction

		resignal; -- raise again the sql exception to the caller

	end;
```

```
set transaction isolation level repeatable read;

start transaction;
        select `nome_località`, `regione_località` from `descrizione` D where D.itinerario =
itin;

open cur;
read_loop: loop

                fetch cur into var_nome, var_regione;
if done then
                        leave read_loop;
end if;

select `zona`, `livello_di_dettaglio` from `cartina` C
where C.nome_località = var_nome and C.regione_località = var_regione;

end loop;
close cur;
```

```
    commit;
END$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure info_alberghi
-- -----------------------------------------------------
```

Per il livello di isolamento scelto ho ipotizzato che qualcuno potrebbe modificare i dati relativi agli alloggi per un certo programma. Dato che sulla tabella 'fermata' faccio select ripetute con operatori aggregati ho bisogno di lockare la tabella con un livello serializable.

```
USE `mydb`;
DROP procedure IF EXISTS `mydb`.`info_alberghi`;

DELIMITER $$
USE `mydb`$$
CREATE PROCEDURE `info_alberghi` (in itin varchar(45))
BEGIN
        declare var_ind varchar(45);
        declare n varchar(45);
        declare nl varchar(45);
        declare rl varchar(45);
        declare g_arrivo int;
```

```
declare g_partenza int;
declare o_arrivo time;
declare o_partenza time;
declare done int default false;
declare cur cursor for select distinct `indirizzo` from (select distinct `indirizzo`, `giornata` from `albergo` A, `fermata` F where A.indirizzo = F.indirizzo_luogo and F.itinerario = itin order by `giornata`) as tb;
declare continue handler for not found set done = true;



declare exit handler for sqlexception

begin
	rollback; -- rollback any changes made in the transaction
	resignal; -- raise again the sql exception to the caller
end;

drop temporary table if exists `orari_albergo`;
  create temporary table `orari_albergo` (
  `nome` varchar(45),
  `indirizzo` varchar(45),
  `nome_località` varchar(45),
  `regione_località` varchar(45),
  `giornata_arrivo` int,
  `ora_arrivo` time,
```

```
`giornata_partenza` int,
`ora_partenza` time
);

    set transaction isolation level serializable;

    start transaction;

    open cur;
    read_loop: loop

                fetch cur into var_ind;
    if done then
                        leave read_loop;
    end if;

    select `nome`, `nome_località`, `regione_località` from `luogo` L
    where L.indirizzo = var_ind into n, nl, rl;

    select min(`giornata`) from `fermata` F
    where
                                F.indirizzo_luogo = var_ind
                        and F.itinerario = itin into g_arrivo;
```

```
        select min(`orario_arrivo`) from `fermata` F
        where
                                F.indirizzo_luogo = var_ind
                            and F.itinerario = itin
                            and F.giornata = g_arrivo into o_arrivo;

        select max(`giornata_partenza`) from `fermata` F
        where
                                    F.indirizzo_luogo = var_ind
                            and F.itinerario = itin into g_partenza;

        select max(`orario_partenza`) from `fermata` F
        where
                                    F.indirizzo_luogo = var_ind
                                and F.itinerario = itin
                                and F.giornata_partenza = g_partenza into o_partenza;

        insert into `orari_albergo` values(n, var_ind, nl, rl, g_arrivo, o_arrivo, g_partenza, o_partenza);

        end loop;
        close cur;

        select * from `orari_albergo`;
    commit;
```

```
END$$
```

Codice SQL per instanziare il database

```
-- MySQL Workbench Forward Engineering

SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0;
SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0;
SET @OLD_SQL_MODE=@@SQL_MODE, SQL_MODE='ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZERO_DATE,ERROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION';

-- -----------------------------------------------------
-- Schema mydb
-- -----------------------------------------------------
DROP SCHEMA IF EXISTS `mydb` ;

-- -----------------------------------------------------
-- Schema mydb
-- -----------------------------------------------------
CREATE SCHEMA IF NOT EXISTS `mydb` DEFAULT CHARACTER SET utf8 ;
USE `mydb` ;

-- -----------------------------------------------------
-- Table `mydb`.`utenti`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `mydb`.`utenti` ;

CREATE TABLE IF NOT EXISTS `mydb`.`utenti` (
  `username` VARCHAR(45) NOT NULL,
  `password` CHAR(32) NOT NULL,
  `ruolo` ENUM('autista', 'hostess', 'meccanico', 'passeggero', 'amministratore') NOT NULL,
```

```
  PRIMARY KEY (`username`))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `mydb`.`hostess`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `mydb`.`hostess` ;

CREATE TABLE IF NOT EXISTS `mydb`.`hostess` (
  `Telefono` VARCHAR(15) NOT NULL,
  `nome` VARCHAR(45) NOT NULL,
  `cognome` VARCHAR(45) NOT NULL,
  `username` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`Telefono`),
  CONSTRAINT `user_hostess`
    FOREIGN KEY (`username`)
    REFERENCES `mydb`.`utenti` (`username`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;

CREATE INDEX `user_idx` ON `mydb`.`hostess` (`username` ASC) VISIBLE;


-- -----------------------------------------------------
-- Table `mydb`.`programma`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `mydb`.`programma` ;

CREATE TABLE IF NOT EXISTS `mydb`.`programma` (
  `nome` VARCHAR(45) NOT NULL,
```

```
  `minimo_partecipanti` INT UNSIGNED NOT NULL,
  `quota` FLOAT UNSIGNED NOT NULL,
  PRIMARY KEY (`nome`))
ENGINE = InnoDB;



-- -----------------------------------------------------
-- Table `mydb`.`viaggio`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `mydb`.`viaggio` ;

CREATE TABLE IF NOT EXISTS `mydb`.`viaggio` (
  `data_partenza` DATE NOT NULL,
  `itinerario` VARCHAR(45) NOT NULL,
  `numero_partecipanti` INT UNSIGNED NOT NULL,
  `ritorno` DATE NOT NULL,
  `costo_adulti` FLOAT UNSIGNED NOT NULL,
  `costo_bambini` FLOAT UNSIGNED NOT NULL,
  `data_cancellazione` DATE NULL,
  `stato` ENUM('confermato', 'cancellato', 'organizzazione') NOT NULL,
  `hostess` VARCHAR(15) NULL,
  PRIMARY KEY (`data_partenza`, `itinerario`),
  CONSTRAINT `hostess_viaggio`
    FOREIGN KEY (`hostess`)
    REFERENCES `mydb`.`hostess` (`Telefono`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `itinerario_viaggio`
    FOREIGN KEY (`itinerario`)
    REFERENCES `mydb`.`programma` (`nome`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
```

```
ENGINE = InnoDB;

CREATE INDEX `hostess_idx` ON `mydb`.`viaggio` (`hostess` ASC) VISIBLE;

CREATE INDEX `itinerario_viaggio_idx` ON `mydb`.`viaggio` (`itinerario` ASC) VISIBLE;


-- -----------------------------------------------------
-- Table `mydb`.`cliente`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `mydb`.`cliente` ;

CREATE TABLE IF NOT EXISTS `mydb`.`cliente` (
  `Telefono` VARCHAR(15) NOT NULL,
  `indirizzo` VARCHAR(45) NOT NULL,
  `nome` VARCHAR(45) NOT NULL,
  `email` VARCHAR(45) NOT NULL,
  `fax` VARCHAR(15) NOT NULL,
  PRIMARY KEY (`Telefono`))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `mydb`.`passeggero`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `mydb`.`passeggero` ;

CREATE TABLE IF NOT EXISTS `mydb`.`passeggero` (
  `Prenotatore` VARCHAR(15) NOT NULL,
  `nome` VARCHAR(45) NOT NULL,
  `età` VARCHAR(2) NOT NULL,
  `posto` VARCHAR(3) NOT NULL,
```

```
  `username` VARCHAR(45) NOT NULL,
  `data_viaggio` DATE NOT NULL,
  `itinerario` VARCHAR(45) NOT NULL,
  `data_prenotazione` DATE NOT NULL,
  `data_invio_documenti` DATE NULL,
  PRIMARY KEY (`posto`, `data_viaggio`, `itinerario`),
  CONSTRAINT `prenotatore_passeggero`
    FOREIGN KEY (`Prenotatore`)
    REFERENCES `mydb`.`cliente` (`Telefono`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `user_passeggero`
    FOREIGN KEY (`username`)
    REFERENCES `mydb`.`utenti` (`username`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `viaggio_passeggero`
    FOREIGN KEY (`data_viaggio` , `itinerario`)
    REFERENCES `mydb`.`viaggio` (`data_partenza` , `itinerario`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


CREATE INDEX `prenotatore_idx` ON `mydb`.`passeggero` (`Prenotatore` ASC) VISIBLE;


CREATE INDEX `user_idx` ON `mydb`.`passeggero` (`username` ASC) VISIBLE;


CREATE INDEX `viaggio_passeggero_idx` ON `mydb`.`passeggero` (`data_viaggio` ASC,
`itinerario` ASC) VISIBLE;



-- -----------------------------------------------------
```

```
-- Table `mydb`.`località`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `mydb`.`località` ;

CREATE TABLE IF NOT EXISTS `mydb`.`località` (
  `nome` VARCHAR(45) NOT NULL,
  `regione` VARCHAR(45) NOT NULL,
  `stato` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`nome`, `regione`))
ENGINE = InnoDB;



-- -----------------------------------------------------
-- Table `mydb`.`descrizione`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `mydb`.`descrizione` ;

CREATE TABLE IF NOT EXISTS `mydb`.`descrizione` (
  `nome_località` VARCHAR(45) NOT NULL,
  `regione_località` VARCHAR(45) NOT NULL,
  `itinerario` VARCHAR(45) NOT NULL,
  `giornata_arrivo` INT UNSIGNED NOT NULL,
  `giornata_partenza` INT UNSIGNED NOT NULL,
  PRIMARY KEY (`nome_località`, `regione_località`, `itinerario`),
  CONSTRAINT `località_descrizione`
    FOREIGN KEY (`nome_località` , `regione_località`)
    REFERENCES `mydb`.`località` (`nome` , `regione`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `itinerario_descrizione`
    FOREIGN KEY (`itinerario`)
    REFERENCES `mydb`.`programma` (`nome`)
```

```
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;

CREATE INDEX `localita_descrizione_idx` ON `mydb`.`descrizione` (`nome_località` ASC, `regione_località` ASC) VISIBLE;

CREATE INDEX `itinerario_descrizione_idx` ON `mydb`.`descrizione` (`itinerario` ASC) VISIBLE;


-- -----------------------------------------------------
-- Table `mydb`.`cartina`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `mydb`.`cartina` ;

CREATE TABLE IF NOT EXISTS `mydb`.`cartina` (
  `Zona` ENUM('nord', 'sud', 'est', 'ovest', 'centro') NOT NULL,
  `nome_località` VARCHAR(45) NOT NULL,
  `regione_località` VARCHAR(45) NOT NULL,
  `livello_di_dettaglio` ENUM('1', '2', '3') NOT NULL,
  `file` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`nome_località`, `regione_località`, `Zona`),
  CONSTRAINT `località_cartina`
    FOREIGN KEY (`nome_località` , `regione_località`)
    REFERENCES `mydb`.`località` (`nome` , `regione`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;

CREATE INDEX `località_idx` ON `mydb`.`cartina` (`nome_località` ASC, `regione_località` ASC) VISIBLE;
```

```
-- -----------------------------------------------------
-- Table `mydb`.`luogo`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `mydb`.`luogo` ;

CREATE TABLE IF NOT EXISTS `mydb`.`luogo` (
  `indirizzo` VARCHAR(45) NOT NULL,
  `nome` VARCHAR(45) NOT NULL,
  `nome_località` VARCHAR(45) NOT NULL,
  `regione_località` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`indirizzo`),
  CONSTRAINT `Localita_luogo`
    FOREIGN KEY (`nome_località` , `regione_località`)
    REFERENCES `mydb`.`località` (`nome` , `regione`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;

CREATE INDEX `Località_idx` ON `mydb`.`luogo` (`nome_località` ASC, `regione_località` ASC)
VISIBLE;

CREATE INDEX `bene_idx` ON `mydb`.`luogo` (`indirizzo` ASC, `nome` ASC) VISIBLE;



-- -----------------------------------------------------
-- Table `mydb`.`distanza`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `mydb`.`distanza` ;

CREATE TABLE IF NOT EXISTS `mydb`.`distanza` (
  `Primo_indirizzo` VARCHAR(45) NOT NULL,
  `secondo_indirizzo` VARCHAR(45) NOT NULL,
```

```
  `metri` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`Primo_indirizzo`, `secondo_indirizzo`),
  CONSTRAINT `primo_luogo`
    FOREIGN KEY (`Primo_indirizzo`)
    REFERENCES `mydb`.`luogo` (`indirizzo`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `secondo_luogo`
    FOREIGN KEY (`secondo_indirizzo`)
    REFERENCES `mydb`.`luogo` (`indirizzo`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;

CREATE INDEX `primo luogo_idx` ON `mydb`.`distanza` (`Primo_indirizzo` ASC) VISIBLE;

CREATE INDEX `secondo luogo_idx` ON `mydb`.`distanza` (`secondo_indirizzo` ASC) VISIBLE;



-- -----------------------------------------------------
-- Table `mydb`.`albergo`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `mydb`.`albergo` ;

CREATE TABLE IF NOT EXISTS `mydb`.`albergo` (
  `indirizzo` VARCHAR(45) NOT NULL,
  `email` VARCHAR(45) NOT NULL,
  `fax` VARCHAR(15) NOT NULL,
  `Categoria` ENUM('1', '2', '3', '4', '5') NOT NULL,
  `Telefono` VARCHAR(15) NOT NULL,
  PRIMARY KEY (`indirizzo`),
  CONSTRAINT `luogo_albergo`
```

```
    FOREIGN KEY (`indirizzo`)
    REFERENCES `mydb`.`luogo` (`indirizzo`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;

CREATE INDEX `luogo_albergo_idx` ON `mydb`.`albergo` (`indirizzo` ASC) VISIBLE;



-- -----------------------------------------------------
-- Table `mydb`.`prenotazione_stanza`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `mydb`.`prenotazione_stanza` ;

CREATE TABLE IF NOT EXISTS `mydb`.`prenotazione_stanza` (
  `Indirizzo_albergo` VARCHAR(45) NOT NULL,
  `data_partenza` DATE NOT NULL,
  `itinerario` VARCHAR(45) NOT NULL,
  `tipo_stanza` ENUM('singola', 'doppia', 'tripla', 'quadrupla', 'suite', 'doppia con 3° letto', 'doppia con 4° letto') NOT NULL,
  `costo` FLOAT UNSIGNED NOT NULL,
  `numero_stanze_prenotate` INT UNSIGNED NOT NULL,
  PRIMARY KEY (`Indirizzo_albergo`, `itinerario`, `data_partenza`, `tipo_stanza`),
  CONSTRAINT `albergo_prenotazione`
    FOREIGN KEY (`Indirizzo_albergo`)
    REFERENCES `mydb`.`albergo` (`indirizzo`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `viaggio`
    FOREIGN KEY (`data_partenza` , `itinerario`)
    REFERENCES `mydb`.`viaggio` (`data_partenza` , `itinerario`)
    ON DELETE NO ACTION
```

```
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


CREATE INDEX `albergo_idx` ON `mydb`.`prenotazione_stanza` (`Indirizzo_albergo` ASC) VISIBLE;


CREATE INDEX `viaggio_idx` ON `mydb`.`prenotazione_stanza` (`data_partenza` ASC, `itinerario` ASC) VISIBLE;



-- -----------------------------------------------------
-- Table `mydb`.`bene`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `mydb`.`bene` ;


CREATE TABLE IF NOT EXISTS `mydb`.`bene` (
  `indirizzo_bene` VARCHAR(45) NOT NULL,
  `nome_bene` VARCHAR(45) NOT NULL,
  `orario_apertura` TIME NOT NULL,
  `telefono` VARCHAR(15) NOT NULL,
  PRIMARY KEY (`indirizzo_bene`),
  CONSTRAINT `luogo_bene`
    FOREIGN KEY (`indirizzo_bene` , `nome_bene`)
    REFERENCES `mydb`.`luogo` (`indirizzo` , `nome`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


CREATE INDEX `luogo_idx` ON `mydb`.`bene` (`indirizzo_bene` ASC, `nome_bene` ASC) VISIBLE;



-- -----------------------------------------------------
```

```
-- Table `mydb`.`visita`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `mydb`.`visita` ;

CREATE TABLE IF NOT EXISTS `mydb`.`visita` (
  `itinerario` VARCHAR(45) NOT NULL,
  `nome_bene` VARCHAR(45) NOT NULL,
  `indirizzo_bene` VARCHAR(45) NOT NULL,
  `compresa` ENUM('si', 'no') NOT NULL,
  `guidata` ENUM('si', 'no') NOT NULL,
  `giornata` INT NOT NULL,
  `orario` TIME NOT NULL,
  PRIMARY KEY (`itinerario`, `indirizzo_bene`),
  CONSTRAINT `bene_visita`
    FOREIGN KEY (`indirizzo_bene` , `nome_bene`)
    REFERENCES `mydb`.`bene` (`indirizzo_bene` , `nome_bene`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `itinerario_visita`
    FOREIGN KEY (`itinerario`)
    REFERENCES `mydb`.`programma` (`nome`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;

CREATE INDEX `itinerario_visita_idx` ON `mydb`.`visita` (`indirizzo_bene` ASC, `nome_bene` ASC) VISIBLE;



-- -----------------------------------------------------
-- Table `mydb`.`servizio_alberghiero`
-- -----------------------------------------------------
```

```
DROP TABLE IF EXISTS `mydb`.`servizio_alberghiero` ;

CREATE TABLE IF NOT EXISTS `mydb`.`servizio_alberghiero` (
  `nome` VARCHAR(45) NOT NULL,
  `descrizione` VARCHAR(100) NOT NULL,
  PRIMARY KEY (`nome`))
ENGINE = InnoDB;



-- -----------------------------------------------------
-- Table `mydb`.`offre`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `mydb`.`offre` ;

CREATE TABLE IF NOT EXISTS `mydb`.`offre` (
  `Indirizzo_albergo` VARCHAR(45) NOT NULL,
  `servizio_alberghiero` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`Indirizzo_albergo`, `servizio_alberghiero`),
  CONSTRAINT `albergo_offre`
    FOREIGN KEY (`Indirizzo_albergo`)
    REFERENCES `mydb`.`albergo` (`indirizzo`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `servizio_offerto`
    FOREIGN KEY (`servizio_alberghiero`)
    REFERENCES `mydb`.`servizio_alberghiero` (`nome`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;

CREATE INDEX `albergo_idx` ON `mydb`.`offre` (`Indirizzo_albergo` ASC) VISIBLE;
```

```
CREATE INDEX `servizio_idx` ON `mydb`.`offre` (`servizio_alberghiero` ASC) VISIBLE;


-- -----------------------------------------------------
-- Table `mydb`.`fermata`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `mydb`.`fermata` ;

CREATE TABLE IF NOT EXISTS `mydb`.`fermata` (
  `indirizzo_luogo` VARCHAR(45) NOT NULL,
  `orario_arrivo` TIME NOT NULL,
  `orario_partenza` TIME NOT NULL,
  `giornata` INT NOT NULL,
  `itinerario` VARCHAR(45) NOT NULL,
  `giornata_partenza` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`indirizzo_luogo`, `giornata`, `itinerario`),
  CONSTRAINT `luogo_fermata`
    FOREIGN KEY (`indirizzo_luogo`)
    REFERENCES `mydb`.`luogo` (`indirizzo`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `itinerario_fermata`
    FOREIGN KEY (`itinerario`)
    REFERENCES `mydb`.`programma` (`nome`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;

CREATE INDEX `itinerario_fermata_idx` ON `mydb`.`fermata` (`itinerario` ASC) VISIBLE;

CREATE INDEX `luogo_fermata_idx` ON `mydb`.`fermata` (`indirizzo_luogo` ASC) VISIBLE;
```

```
-- -----------------------------------------------------
-- Table `mydb`.`modello_pullman`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `mydb`.`modello_pullman` ;

CREATE TABLE IF NOT EXISTS `mydb`.`modello_pullman` (
  `costruttore` VARCHAR(45) NOT NULL,
  `nome` VARCHAR(45) NOT NULL,
  `velocità_massima` VARCHAR(45) NOT NULL,
  `serbatoio` VARCHAR(45) NOT NULL,
  `cavalli` VARCHAR(45) NOT NULL,
  `numero_posti` INT UNSIGNED NOT NULL,
  PRIMARY KEY (`costruttore`, `nome`))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `mydb`.`pullman`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `mydb`.`pullman` ;

CREATE TABLE IF NOT EXISTS `mydb`.`pullman` (
  `targa` VARCHAR(7) NOT NULL,
  `contatore` INT UNSIGNED NULL,
  `data_immatricolazione` DATE NULL,
  `costruttore` VARCHAR(45) NULL,
  `nome_modello` VARCHAR(45) NULL,
  PRIMARY KEY (`targa`),
  CONSTRAINT `modello_pullman`
    FOREIGN KEY (`costruttore` , `nome_modello`)
    REFERENCES `mydb`.`modello_pullman` (`costruttore` , `nome`)
```

```
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;

CREATE INDEX `modello_idx` ON `mydb`.`pullman` (`costruttore` ASC, `nome_modello` ASC)
VISIBLE;



-- -----------------------------------------------------
-- Table `mydb`.`foto`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `mydb`.`foto` ;

CREATE TABLE IF NOT EXISTS `mydb`.`foto` (
  `nome_file` INT NOT NULL,
  `nome_località` VARCHAR(45) NULL,
  `regione_località` VARCHAR(45) NULL,
  `pullman` VARCHAR(7) NULL,
  PRIMARY KEY (`nome_file`),
  CONSTRAINT `localita_foto`
    FOREIGN KEY (`nome_località` , `regione_località`)
    REFERENCES `mydb`.`località` (`nome` , `regione`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `pullman_foto`
    FOREIGN KEY (`pullman`)
    REFERENCES `mydb`.`pullman` (`targa`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;

CREATE INDEX `pullman_foto_idx` ON `mydb`.`foto` (`pullman` ASC) VISIBLE;
```

```
CREATE INDEX `localita_foto_idx` ON `mydb`.`foto` (`nome_località` ASC, `regione_località` ASC) VISIBLE;



-- -----------------------------------------------------
-- Table `mydb`.`autista`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `mydb`.`autista` ;

CREATE TABLE IF NOT EXISTS `mydb`.`autista` (
  `Telefono` VARCHAR(45) NOT NULL,
  `nome` VARCHAR(45) NOT NULL,
  `cognome` VARCHAR(45) NOT NULL,
  `username` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`Telefono`),
  CONSTRAINT `user_autista`
    FOREIGN KEY (`username`)
    REFERENCES `mydb`.`utenti` (`username`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;

CREATE INDEX `user_idx` ON `mydb`.`autista` (`username` ASC) VISIBLE;



-- -----------------------------------------------------
-- Table `mydb`.`trasporto`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `mydb`.`trasporto` ;

CREATE TABLE IF NOT EXISTS `mydb`.`trasporto` (
```

```
  `autista` VARCHAR(15) NOT NULL,
  `pullman` VARCHAR(45) NOT NULL,
  `data_partenza` DATE NOT NULL,
  `itinerario` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`autista`, `pullman`, `data_partenza`, `itinerario`),
  CONSTRAINT `autista_trasporto`
    FOREIGN KEY (`autista`)
    REFERENCES `mydb`.`autista` (`Telefono`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `pullman_trasporto`
    FOREIGN KEY (`pullman`)
    REFERENCES `mydb`.`pullman` (`targa`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `viaggio_trasporto`
    FOREIGN KEY (`data_partenza` , `itinerario`)
    REFERENCES `mydb`.`viaggio` (`data_partenza` , `itinerario`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


CREATE INDEX `pullman_idx` ON `mydb`.`trasporto` (`pullman` ASC) VISIBLE;


CREATE INDEX `viaggio_idx` ON `mydb`.`trasporto` (`data_partenza` ASC, `itinerario` ASC) VISIBLE;


CREATE INDEX `autista_trasporto_idx` ON `mydb`.`trasporto` (`autista` ASC) VISIBLE;



-- -----------------------------------------------------
-- Table `mydb`.`controllo_ordinario`
```

```
-- -----------------------------------------------------
DROP TABLE IF EXISTS `mydb`.`controllo_ordinario` ;


CREATE TABLE IF NOT EXISTS `mydb`.`controllo_ordinario` (
  `tipo` ENUM('motore', 'convergenza', 'freni', 'pneumatici', 'luci', 'batteria', 'sospensioni', 'carrozzeria') NOT NULL,
  `intervallo` INT UNSIGNED NOT NULL,
  PRIMARY KEY (`tipo`))
ENGINE = InnoDB;



-- -----------------------------------------------------
-- Table `mydb`.`manutenzione`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `mydb`.`manutenzione` ;


CREATE TABLE IF NOT EXISTS `mydb`.`manutenzione` (
  `pullman` VARCHAR(7) NOT NULL,
  `controllo_ordinario` ENUM('motore', 'convergenza', 'freni', 'pneumatici', 'luci', 'batteria', 'sospensioni', 'carrozzeria') NOT NULL,
  `prossimo_controllo` INT UNSIGNED NOT NULL,
  PRIMARY KEY (`pullman`, `controllo_ordinario`),
  CONSTRAINT `pullman_manutenzione`
    FOREIGN KEY (`pullman`)
    REFERENCES `mydb`.`pullman` (`targa`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `controllo_manutenzione`
    FOREIGN KEY (`controllo_ordinario`)
    REFERENCES `mydb`.`controllo_ordinario` (`tipo`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

```
CREATE INDEX `controllo_idx` ON `mydb`.`manutenzione` (`controllo_ordinario` ASC) VISIBLE;

CREATE INDEX `pullman_manutenzione_idx` ON `mydb`.`manutenzione` (`pullman` ASC) VISIBLE;



-- -----------------------------------------------------
-- Table `mydb`.`sessione_controllo`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `mydb`.`sessione_controllo` ;

CREATE TABLE IF NOT EXISTS `mydb`.`sessione_controllo` (
  `data_inizio` DATE NOT NULL,
  `pullman` VARCHAR(7) NOT NULL,
  `data_fine` DATE NULL,
  `motivo` VARCHAR(100) NULL,
  PRIMARY KEY (`data_inizio`, `pullman`),
  CONSTRAINT `pullman_sessione`
    FOREIGN KEY (`pullman`)
    REFERENCES `mydb`.`pullman` (`targa`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;

CREATE INDEX `pullman_idx` ON `mydb`.`sessione_controllo` (`pullman` ASC) VISIBLE;



-- -----------------------------------------------------
-- Table `mydb`.`tipo_sessione`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `mydb`.`tipo_sessione` ;
```

```
CREATE TABLE IF NOT EXISTS `mydb`.`tipo_sessione` (
  `controllo_ordinario` ENUM('motore', 'convergenza', 'freni', 'pneumatici', 'luci', 'batteria', 'sospensioni', 'carrozzeria') NOT NULL,
  `data_controllo` DATE NOT NULL,
  `pullman` VARCHAR(7) NOT NULL,
  PRIMARY KEY (`controllo_ordinario`, `data_controllo`, `pullman`),
  CONSTRAINT `controllo_tipo`
    FOREIGN KEY (`controllo_ordinario`)
    REFERENCES `mydb`.`controllo_ordinario` (`tipo`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `sessione_tipo`
    FOREIGN KEY (`pullman` , `data_controllo`)
    REFERENCES `mydb`.`sessione_controllo` (`pullman` , `data_inizio`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;

CREATE INDEX `sessione_tipo_idx` ON `mydb`.`tipo_sessione` (`pullman` ASC, `data_controllo` ASC) VISIBLE;

CREATE INDEX `controllo_tipo` ON `mydb`.`tipo_sessione` (`controllo_ordinario` ASC) VISIBLE;



-- -----------------------------------------------------
-- Table `mydb`.`ricambio`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `mydb`.`ricambio` ;

CREATE TABLE IF NOT EXISTS `mydb`.`ricambio` (
  `nome` VARCHAR(45) NOT NULL,
  `disponibile` INT UNSIGNED NOT NULL,
```

```
  `codice` VARCHAR(5) NOT NULL,
  `ordinato` INT UNSIGNED NOT NULL,
  PRIMARY KEY (`codice`))
ENGINE = InnoDB;



-- -----------------------------------------------------
-- Table `mydb`.`sostituzione`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `mydb`.`sostituzione` ;

CREATE TABLE IF NOT EXISTS `mydb`.`sostituzione` (
  `data_controllo` DATE NOT NULL,
  `pullman` VARCHAR(45) NOT NULL,
  `ricambio` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`data_controllo`, `pullman`, `ricambio`),
  CONSTRAINT `sessione_controllo_sostituzione`
    FOREIGN KEY (`data_controllo` , `pullman`)
    REFERENCES `mydb`.`sessione_controllo` (`data_inizio` , `pullman`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `ricambio_sostituzione`
    FOREIGN KEY (`ricambio`)
    REFERENCES `mydb`.`ricambio` (`codice`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;

CREATE INDEX `sessione controllo_idx` ON `mydb`.`sostituzione` (`data_controllo` ASC, `pullman` ASC) VISIBLE;

CREATE INDEX `ricambio_idx` ON `mydb`.`sostituzione` (`ricambio` ASC) VISIBLE;
```

```
-- -----------------------------------------------------
-- Table `mydb`.`meccanico`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `mydb`.`meccanico` ;

CREATE TABLE IF NOT EXISTS `mydb`.`meccanico` (
  `telefono` VARCHAR(15) NOT NULL,
  `nome` VARCHAR(45) NOT NULL,
  `cognome` VARCHAR(45) NOT NULL,
  `username` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`telefono`),
  CONSTRAINT `user_meccanico`
    FOREIGN KEY (`username`)
    REFERENCES `mydb`.`utenti` (`username`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;

CREATE INDEX `user_idx` ON `mydb`.`meccanico` (`username` ASC) VISIBLE;



-- -----------------------------------------------------
-- Table `mydb`.`eseguito`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `mydb`.`eseguito` ;

CREATE TABLE IF NOT EXISTS `mydb`.`eseguito` (
  `meccanico` VARCHAR(15) NOT NULL,
  `data_controllo` DATE NOT NULL,
  `pullman` VARCHAR(7) NOT NULL,
```

```
  PRIMARY KEY (`meccanico`, `data_controllo`, `pullman`),
  CONSTRAINT `meccanico_eseguito`
    FOREIGN KEY (`meccanico`)
    REFERENCES `mydb`.`meccanico` (`telefono`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `sessione_controllo_eseguito`
    FOREIGN KEY (`data_controllo` , `pullman`)
    REFERENCES `mydb`.`sessione_controllo` (`data_inizio` , `pullman`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


CREATE INDEX `sessione controllo_idx` ON `mydb`.`eseguito` (`data_controllo` ASC, `pullman` ASC) VISIBLE;


CREATE INDEX `meccanico_controllo_idx` ON `mydb`.`eseguito` (`meccanico` ASC) VISIBLE;



-- -----------------------------------------------------
-- Table `mydb`.`competenze`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `mydb`.`competenze` ;


CREATE TABLE IF NOT EXISTS `mydb`.`competenze` (
  `costruttore` VARCHAR(45) NOT NULL,
  `nome_modello` VARCHAR(45) NOT NULL,
  `meccanico` VARCHAR(15) NOT NULL,
  PRIMARY KEY (`costruttore`, `meccanico`, `nome_modello`),
  CONSTRAINT `modello_pullman_competenze`
    FOREIGN KEY (`costruttore` , `nome_modello`)
    REFERENCES `mydb`.`modello_pullman` (`costruttore` , `nome`)
```

```
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `meccanico_competenze`
    FOREIGN KEY (`meccanico`)
    REFERENCES `mydb`.`meccanico` (`telefono`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


CREATE INDEX `modello pullman_idx` ON `mydb`.`competenze` (`costruttore` ASC, `nome_modello` ASC) VISIBLE;


CREATE INDEX `meccanico_idx` ON `mydb`.`competenze` (`meccanico` ASC) VISIBLE;



-- -----------------------------------------------------
-- Table `mydb`.`comfort`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `mydb`.`comfort` ;


CREATE TABLE IF NOT EXISTS `mydb`.`comfort` (
  `descrizione` VARCHAR(100) NOT NULL,
  `nome` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`nome`))
ENGINE = InnoDB;



-- -----------------------------------------------------
-- Table `mydb`.`allestimento`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `mydb`.`allestimento` ;
```

```
CREATE TABLE IF NOT EXISTS `mydb`.`allestimento` (
  `comfort` VARCHAR(45) NOT NULL,
  `costruttore` VARCHAR(45) NOT NULL,
  `nome_modello` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`comfort`, `costruttore`, `nome_modello`),
  CONSTRAINT `comfort_allestimento`
    FOREIGN KEY (`comfort`)
    REFERENCES `mydb`.`comfort` (`nome`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `modello_allestimento`
    FOREIGN KEY (`costruttore` , `nome_modello`)
    REFERENCES `mydb`.`modello_pullman` (`costruttore` , `nome`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


CREATE INDEX `modello_allestimento_idx` ON `mydb`.`allestimento` (`costruttore` ASC, `nome_modello` ASC) VISIBLE;


USE `mydb` ;


-- -----------------------------------------------------
-- procedure orario_fermate
-- -----------------------------------------------------


USE `mydb`;
DROP procedure IF EXISTS `mydb`.`orario_fermate`;


DELIMITER $$
USE `mydb`$$
CREATE PROCEDURE `orario_fermate` (in itin varchar(45), in giorn INT)
```

```
BEGIN
	select
		`indirizzo_luogo`, `nome`, `orario_arrivo`, `orario_partenza` from `fermata` F, `luogo`
L
	where
			F.`itinerario` = itin
	and F.indirizzo_luogo = L.indirizzo
		and F.`giornata` = giorn;
END$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure lettura_contatore
-- -----------------------------------------------------

USE `mydb`;
DROP procedure IF EXISTS `mydb`.`lettura_contatore`;

DELIMITER $$
USE `mydb`$$
CREATE PROCEDURE `lettura_contatore` (in pull varchar(7))
BEGIN
	select
		`contatore` from `pullman` P
	where
		P.`targa` = pull;
END$$

DELIMITER ;

-- -----------------------------------------------------
```

```
-- procedure prossima_revisione
-- -----------------------------------------------------

USE `mydb`;
DROP procedure IF EXISTS `mydb`.`prossima_revisione`;

DELIMITER $$
USE `mydb`$$
CREATE PROCEDURE `prossima_revisione` (in pull varchar(7), in contr ENUM('motore', 'convergenza', 'freni', 'pneumatici', 'luci', 'batteria', 'sospensioni', 'carrozzeria'))
BEGIN
select `prossimo_Controllo` from `manutenzione` M
where M.pullman = pull and M.controllo_ordinario = contr;
END$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure login
-- -----------------------------------------------------

USE `mydb`;
DROP procedure IF EXISTS `mydb`.`login`;

DELIMITER $$
USE `mydb`$$
CREATE PROCEDURE `login` (in var_username varchar(45), in var_pass varchar(45), out var_role INT)
BEGIN
	declare var_user_role ENUM('autista', 'hostess', 'meccanico', 'passeggero', 'amministratore');

	select `ruolo` from `utenti`
		where `username` = var_username
```

```
	and `password` = md5(var_pass)
	into var_user_role;

	-- See the corresponding enum in the client
		if var_user_role = 'autista' then
			set var_role = 1;
		elseif var_user_role = 'hostess' then
			set var_role = 2;
		elseif var_user_role = 'meccanico' then
			set var_role = 3;
		elseif var_user_role = 'passeggero' then
			set var_role = 4;
		elseif var_user_role = 'amministratore' then
			set var_role = 5;
		else
			set var_role = 6;
		end if;
END$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure aggiungi_employee
-- -----------------------------------------------------

USE `mydb`;
DROP procedure IF EXISTS `mydb`.`aggiungi_employee`;

DELIMITER $$
USE `mydb`$$
CREATE PROCEDURE `aggiungi_employee` (IN tel VARCHAR(15), IN nome VARCHAR(45), IN
cognome VARCHAR(45), IN username VARCHAR(45), IN pass VARCHAR(45), IN ruolo int)
```

```
BEGIN

declare exit handler for sqlexception

begin
        rollback; -- rollback any changes made in the transaction
        resignal; -- raise again the sql exception to the caller
end;

set transaction isolation level read uncommitted;

start transaction;
   if ruolo = 0 then
                insert into utenti VALUES(username, MD5(pass), 'hostess');
                insert into `hostess` values(tel, nome, cognome, username);
        elseif ruolo = 1 then
                insert into utenti VALUES(username, MD5(pass), 'meccanico');
                insert into `meccanico` values(tel, nome, cognome, username);
   elseif ruolo = 2 then
                insert into utenti VALUES(username, MD5(pass), 'autista');
                insert into `autista` values(tel, nome, cognome, username);
        end if;
commit;


END$$

DELIMITER ;

-- ----------------------------------------------------
-- procedure crea_utente
-- ----------------------------------------------------
```

```
USE `mydb`;
DROP procedure IF EXISTS `mydb`.`crea_utente`;

DELIMITER $$
USE `mydb`$$
CREATE PROCEDURE `crea_utente` (IN username VARCHAR(45), IN pass VARCHAR(45), IN ruolo varchar(45))
BEGIN
	insert into utenti VALUES(username, MD5(pass), ruolo);
END$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure inserisci_sessione_controllo
-- -----------------------------------------------------

USE `mydb`;
DROP procedure IF EXISTS `mydb`.`inserisci_sessione_controllo`;

DELIMITER $$
USE `mydb`$$
CREATE PROCEDURE `inserisci_sessione_controllo` ( in pull varchar(7), in data_i date, in data_f date, in motivo text, in motore int, in convergenza int, in freni int, in pneumatici int, in luci int, in batteria int, in sospensioni int, in carrozzeria int, IN uname varchar(15))
BEGIN
	declare tel varchar(15);
   declare counter int;
   declare exit handler for sqlexception

	begin
		rollback; -- rollback any changes made in the transaction
```

```
            resignal; -- raise again the sql exception to the caller
        end;

        set transaction isolation level read committed;

        start transaction;

    select `telefono` from `meccanico` M
    where M.username = uname into tel;

            insert into `sessione_controllo` values(data_i, pull, data_f, motivo);

    insert into `eseguito` values(tel, data_i, pull);

    if motivo IS NULL then

        if motore <> 1 then
                        insert into tipo_sessione values('motore', data_i, pull);

            select
                        `intervallo` from `controllo_ordinario` C
            where
                        C.tipo = 'motore' into counter;


            UPDATE
                        `manutenzione` M, `pullman` P SET M.prossimo_controllo=
P.contatore + counter
                    WHERE
                        M.pullman = pullman
              and M.controllo_ordinario = 'motore';
        end if;
```

```
        if convergenza <> 1 then

                        insert into tipo_sessione values('convergenza', data_i, pull);

          select
                        `intervallo` from `controllo_ordinario` C
          where
                        C.tipo = 'convergenza' into counter;



          UPDATE
                        `manutenzione` M, `pullman` P SET M.prossimo_controllo=
P.contatore + counter
                    WHERE
                        M.pullman = pullman
            and M.controllo_ordinario = 'convergenza';
        end if;

        if freni <> 1 then

                        insert into tipo_sessione values('freni', data_i, pull);

          select
                        `intervallo` from `controllo_ordinario` C
          where
                        C.tipo = 'freni' into counter;



          UPDATE
                        `manutenzione` M, `pullman` P SET M.prossimo_controllo =
P.contatore + counter
                    WHERE
```

```
                                M.pullman = pullman
        and M.controllo_ordinario = 'freni';
    end if;

    if pneumatici <> 1 then

                        insert into tipo_sessione values('pneumatici', data_i, pull);

        select
                                `intervallo` from `controllo_ordinario` C
        where
                                C.tipo = 'pneumatici' into counter;


        UPDATE
                                `manutenzione` M, `pullman` P SET M.prossimo_controllo=
P.contatore + counter
                        WHERE
                                M.pullman = pullman
        and M.controllo_ordinario = 'pneumatici';
    end if;

    if luci <> 1 then

                        insert into tipo_sessione values('luci', data_i, pull);

        select
                                `intervallo` from `controllo_ordinario` C
        where
                                C.tipo = 'luci' into counter;
```

```
        UPDATE
                `manutenzione` M, `pullman` P SET M.prossimo_controllo=
P.contatore + counter
            WHERE
                M.pullman = pullman
          and M.controllo_ordinario = 'luci';
      end if;

      if batteria <> 1 then

            insert into tipo_sessione values('batteria', data_i, pull);

        select
                `intervallo` from `controllo_ordinario` C
        where
                C.tipo = 'batteria' into counter;


        UPDATE
                `manutenzione` M, `pullman` P SET M.prossimo_controllo=
P.contatore + counter
            WHERE
                M.pullman = pullman
                and M.controllo_ordinario = 'batteria';
      end if;

      if sospensioni <> 1 then

            insert into tipo_sessione values('sospensioni', data_i, pull);

        select
                `intervallo` from `controllo_ordinario` C
        where
```

```
                                C.tipo = 'sospensioni' into counter;


        UPDATE
                                `manutenzione` M, `pullman` P SET M.prossimo_controllo=
P.contatore + counter
                            WHERE
                                M.pullman = pullman
            and M.controllo_ordinario = 'sospensioni';
        end if;

        if carrozzeria <> 1 then

                            insert into tipo_sessione values('carrozzeria', data_i, pull);

            select
                                `intervallo` from `controllo_ordinario` C
            where
                                C.tipo = 'carrozzeria' into counter;



        UPDATE
                                `manutenzione` M, `pullman` P SET M.prossimo_controllo=
P.contatore + counter
                            WHERE
                                M.pullman = pullman
            and M.controllo_ordinario = 'carrozzeria';
        end if;
              end if;
        commit;
END$$

DELIMITER ;
```

```
-- -----------------------------------------------------
-- procedure usa_ricambio
-- -----------------------------------------------------

USE `mydb`;
DROP procedure IF EXISTS `mydb`.`usa_ricambio`;

DELIMITER $$
USE `mydb`$$
CREATE PROCEDURE `usa_ricambio` (in cod varchar(5), in inizio date, in pull varchar(7))
BEGIN

	declare d int;
	declare exit handler for sqlexception

	begin
		rollback; -- rollback any changes made in the transaction
		resignal; -- raise again the sql exception to the caller
	end;

	set transaction isolation level repeatable read;

	start transaction;
		select
			`disponibile` from `ricambio` R
	where
			R.codice = cod into d;

	if d > 0 then
			insert into `sostituzione` values(inizio, pull, cod);
```

```
        update
                        `ricambio` R set `disponibile` = d - 1
        where
                        R.codice = cod;
                else
                        signal sqlstate "45000";
    end if;
  commit;
END$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure viaggio_assegnato_hostess
-- -----------------------------------------------------

USE `mydb`;
DROP procedure IF EXISTS `mydb`.`viaggio_assegnato_hostess`;

DELIMITER $$
USE `mydb`$$
CREATE PROCEDURE `viaggio_assegnato_hostess` (in uname varchar(45))
BEGIN
        select `data_partenza`, `itinerario`, `numero_partecipanti`, `ritorno`, `stato` from `viaggio` V, `hostess` H
  where
                H.username = uname
    and V.hostess = H.telefono;
END$$

DELIMITER ;
```

```
-- -----------------------------------------------------
-- procedure calcola_distanza
-- -----------------------------------------------------

USE `mydb`;
DROP procedure IF EXISTS `mydb`.`calcola_distanza`;

DELIMITER $$
USE `mydb`$$
CREATE PROCEDURE `calcola_distanza` (in ind_1 varchar(45), in ind_2 varchar(45))
BEGIN
        select `metri` from `distanza` D
   where
                (D.primo_indirizzo = ind_1 and D.secondo_indirizzo = ind_2)
      or (D.secondo_indirizzo = ind_1 and D.primo_indirizzo = ind_2);
END$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure inserisci_cliente
-- -----------------------------------------------------

USE `mydb`;
DROP procedure IF EXISTS `mydb`.`inserisci_cliente`;

DELIMITER $$
USE `mydb`$$
CREATE PROCEDURE `inserisci_cliente` (in tel varchar(15), in ind varchar(45), in n varchar(45), in
mail varchar(45), in f varchar(15))
BEGIN
        insert into `cliente` values(tel, ind, n, mail, f);
```

```
END$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure inserisci_passeggero
-- -----------------------------------------------------

USE `mydb`;
DROP procedure IF EXISTS `mydb`.`inserisci_passeggero`;

DELIMITER $$
USE `mydb`$$
CREATE PROCEDURE `inserisci_passeggero` (in pren varchar(15), in n varchar(45), in e varchar(2), in p varchar(3), in uname varchar(45), in pass varchar(45), in dv date, in itin varchar(45), in data_pren date)
BEGIN

	declare counter int;
	declare exit handler for sqlexception

	begin
		rollback; -- rollback any changes made in the transaction
		resignal; -- raise again the sql exception to the caller
	end;

	set transaction isolation level serializable;

	start transaction;
		select count(*) from `viaggio` V
      where
				V.data_partenza = dv
	and V.itinerario = itin
```

```
        and V.stato = 'organizzazione' or V.stato = 'confermato' into counter;


    if counter <> 0 then
                    insert into `utenti` values(uname, md5(pass), 'passeggero');

                    insert into `passeggero` values(pren, n, e, p, uname, dv, itin, data_pren, NULL);

                    update `viaggio` V
                    set V.numero_partecipanti = V.numero_partecipanti + 1
                    where
                                V.itinerario = itin
                            and V.data_partenza = dv;
                end if;
  commit;
END$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure prenota_stanza
-- -----------------------------------------------------

USE `mydb`;
DROP procedure IF EXISTS `mydb`.`prenota_stanza`;

DELIMITER $$
USE `mydb`$$
CREATE PROCEDURE `prenota_stanza` (in al varchar(45), in dp date, in itin varchar(45), in tipo ENUM('singola', 'doppia', 'tripla', 'quadrupla', 'suite', 'doppia con 3° letto', 'doppia con 4° letto'))
BEGIN
```

```
    declare counter int;
    declare exit handler for sqlexception

    begin
        rollback; -- rollback any changes made in the transaction
        resignal; -- raise again the sql exception to the caller
    end;

    set transaction isolation level serializable;

    start transaction;
        select count(*) from `viaggio` V
  where
                V.data_partenza = dp
    and V.itinerario = itin
    and V.stato = 'organizzazione' or V.stato = 'confermato' into counter;

  if counter <> 0 then
            update
                `prenotazione_stanza` P
            set
                P.numero_stanze_prenotate = P.numero_stanze_prenotate + 1
            where
                    P.indirizzo_albergo = al
                and P.data_partenza = dp
                and P.itinerario = itin
                and P.tipo_stanza = tipo;
        end if;
    commit;
END$$

DELIMITER ;
```

```
-- -----------------------------------------------------
-- procedure inserisci_costo
-- -----------------------------------------------------

USE `mydb`;
DROP procedure IF EXISTS `mydb`.`inserisci_costo`;

DELIMITER $$
USE `mydb`$$
CREATE PROCEDURE `inserisci_costo` (in al varchar(45), in dp date, in itin varchar(45), in tipo ENUM('singola', 'doppia', 'tripla', 'quadrupla', 'suite', 'doppia con 3° letto', 'doppia con 4° letto'), in c int)
BEGIN
		insert into `prenotazione_stanza` values(al, dp, itin, tipo, c, 0);
END$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure conferma_viaggio
-- -----------------------------------------------------

USE `mydb`;
DROP procedure IF EXISTS `mydb`.`conferma_viaggio`;

DELIMITER $$
USE `mydb`$$
CREATE PROCEDURE `conferma_viaggio` (in dp date , in itin varchar(45), in hos varchar(15))
BEGIN
	update `viaggio` V
   set V.stato = 'confermato', V.hostess = hos
   where
```

```
                        V.data_partenza = dp
                and V.itinerario = itin;
END$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure stato_viaggio
-- -----------------------------------------------------

USE `mydb`;
DROP procedure IF EXISTS `mydb`.`stato_viaggio`;

DELIMITER $$
USE `mydb`$$
CREATE PROCEDURE `stato_viaggio` (in uname varchar(45))
BEGIN
        select V.itinerario, `data_partenza`, `stato`, `hostess` from `viaggio` V, `passeggero` P
   where
                        P.username = uname
      and P.itinerario = V.itinerario
      and P.data_viaggio = V.data_partenza;
END$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure info_visite_passeggero
-- -----------------------------------------------------

USE `mydb`;
DROP procedure IF EXISTS `mydb`.`info_visite_passeggero`;
```

```
DELIMITER $$
USE `mydb`$$
CREATE PROCEDURE `info_visite_passeggero` (in uname varchar(45))
BEGIN
        select `nome_bene`, `compresa`, `guidata`, `giornata`, `orario` from `passeggero` P, `visita` V
   where
                        P.username = uname
      and P.itinerario = V.itinerario;
END$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure info_visite_autista
-- -----------------------------------------------------

USE `mydb`;
DROP procedure IF EXISTS `mydb`.`info_visite_autista`;

DELIMITER $$
USE `mydb`$$
CREATE PROCEDURE `info_visite_autista` (itin varchar(45))
BEGIN
        select `nome_bene`, `giornata`, `orario` from  `visita` V
        where
                V.itinerario = itin;
END$$

DELIMITER ;

-- -----------------------------------------------------
```

```
-- procedure viaggio_assegnato_autista
-- -----------------------------------------------------

USE `mydb`;
DROP procedure IF EXISTS `mydb`.`viaggio_assegnato_autista`;

DELIMITER $$
USE `mydb`$$
CREATE PROCEDURE `viaggio_assegnato_autista` (in uname varchar(45))
BEGIN
	select `itinerario`, `data_partenza`, `pullman` from `trasporto` T, `autista` A
   where
				A.username = uname
		and A.telefono = T.autista;
END$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure lista_viaggi
-- -----------------------------------------------------

USE `mydb`;
DROP procedure IF EXISTS `mydb`.`lista_viaggi`;

DELIMITER $$
USE `mydb`$$
CREATE PROCEDURE `lista_viaggi` ()
BEGIN
	select `itinerario`, `data_partenza`, `ritorno`, `numero_partecipanti`, `costo_adulti`, `costo_bambini`, `stato` from `viaggio` V
   where
```

```
                V.stato = 'organizzazione'
    or V.stato = 'confermato';
END$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure ricambi_disponibili
-- -----------------------------------------------------

USE `mydb`;
DROP procedure IF EXISTS `mydb`.`ricambi_disponibili`;

DELIMITER $$
USE `mydb`$$
CREATE PROCEDURE `ricambi_disponibili` ()
BEGIN
        select * from `ricambio` R
  where R.disponibile > 0;
END$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure ordina_ricambio
-- -----------------------------------------------------

USE `mydb`;
DROP procedure IF EXISTS `mydb`.`ordina_ricambio`;

DELIMITER $$
USE `mydb`$$
```

```
CREATE PROCEDURE `ordina_ricambio` (in cod varchar(5))
BEGIN
	update `ricambio` R set `ordinato` = 1
   where
		R.codice = cod;
END$$

DELIMITER ;

-- ----------------------------------------------------
-- procedure revisioni_da_effettuare
-- ----------------------------------------------------

USE `mydb`;
DROP procedure IF EXISTS `mydb`.`revisioni_da_effettuare`;

DELIMITER $$
USE `mydb`$$
CREATE PROCEDURE `revisioni_da_effettuare` ()
BEGIN
	select M.pullman, M.controllo_ordinario from `manutenzione` M, `pullman` P
   where P.contatore > M.prossimo_controllo and P.targa = M.pullman
   order by M.pullman;
END$$

DELIMITER ;

-- ----------------------------------------------------
-- procedure cartine_programma
-- ----------------------------------------------------

USE `mydb`;
```

```
DROP procedure IF EXISTS `mydb`.`cartine_programma`;


DELIMITER $$
USE `mydb`$$
CREATE PROCEDURE `cartine_programma` (in itin varchar(45))
BEGIN
	declare var_nome varchar(45);
   declare var_regione varchar(45);
	declare done int default false;
   declare cur cursor for select `nome_località`, `regione_località` from `descrizione` D where D.itinerario = itin;
   declare continue handler for not found set done = true;



	declare exit handler for sqlexception


	begin
		rollback; -- rollback any changes made in the transaction
		resignal; -- raise again the sql exception to the caller
	end;


   set transaction isolation level repeatable read;


	start transaction;
		select `nome_località`, `regione_località` from `descrizione` D where D.itinerario = itin;



		open cur;
		read_loop: loop


			fetch cur into var_nome, var_regione;
	if done then
```

```
                                leave read_loop;
        end if;

        select `zona`, `livello_di_dettaglio` from `cartina` C
        where C.nome_località = var_nome and C.regione_località = var_regione;




            end loop;
            close cur;


   commit;
END$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure info_alberghi
-- -----------------------------------------------------

USE `mydb`;
DROP procedure IF EXISTS `mydb`.`info_alberghi`;

DELIMITER $$
USE `mydb`$$
CREATE PROCEDURE `info_alberghi` (in itin varchar(45))
BEGIN
        declare var_ind varchar(45);
   declare n varchar(45);
   declare nl varchar(45);
```

```
declare rl varchar(45);
declare g_arrivo int;
declare g_partenza int;
declare o_arrivo time;
declare o_partenza time;
	declare done int default false;
declare cur cursor for select distinct `indirizzo` from (select distinct `indirizzo`, `giornata` from `albergo` A, `fermata` F where A.indirizzo = F.indirizzo_luogo and F.itinerario = itin order by `giornata`) as tb;
declare continue handler for not found set done = true;



	declare exit handler for sqlexception


	begin
		rollback; -- rollback any changes made in the transaction
		resignal; -- raise again the sql exception to the caller
	end;


	drop temporary table if exists `orari_albergo`;
create temporary table `orari_albergo` (
`nome` varchar(45),
`indirizzo` varchar(45),
`nome_località` varchar(45),
`regione_località` varchar(45),
`giornata_arrivo` int,
`ora_arrivo` time,
`giornata_partenza` int,
`ora_partenza` time
);


set transaction isolation level serializable;
```

```
start transaction;

        open cur;
        read_loop: loop

                fetch cur into var_ind;
if done then
                        leave read_loop;
end if;

select `nome`, `nome_località`, `regione_località` from `luogo` L
where L.indirizzo = var_ind into n, nl, rl;

select min(`giornata`) from `fermata` F
where
                                F.indirizzo_luogo = var_ind
                        and F.itinerario = itin into g_arrivo;

                select min(`orario_arrivo`) from `fermata` F
where
                                F.indirizzo_luogo = var_ind
                        and F.itinerario = itin
   and F.giornata = g_arrivo into o_arrivo;

                select max(`giornata_partenza`) from `fermata` F
where
                                F.indirizzo_luogo = var_ind
                        and F.itinerario = itin into g_partenza;

select max(`orario_partenza`) from `fermata` F
where
                                F.indirizzo_luogo = var_ind
```

```
                                and F.itinerario = itin
            and F.giornata_partenza = g_partenza into o_partenza;

        insert into `orari_albergo` values(n, var_ind, nl, rl, g_arrivo, o_arrivo, g_partenza, o_partenza);

                end loop;
                close cur;

                select * from `orari_albergo`;
    commit;
END$$

DELIMITER ;
SET SQL_MODE = '';
DROP USER IF EXISTS login;
SET
SQL_MODE='ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO
_ZERO_DATE,ERROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION';
CREATE USER 'login' IDENTIFIED BY 'login';

GRANT ALL ON `mydb`.* TO 'login';
GRANT EXECUTE ON procedure `mydb`.`login` TO 'login';
GRANT EXECUTE ON procedure `mydb`.`lista_viaggi` TO 'login';
SET SQL_MODE = '';
DROP USER IF EXISTS hostess;
SET
SQL_MODE='ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO
_ZERO_DATE,ERROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION';
CREATE USER 'hostess' IDENTIFIED BY 'hostess';

GRANT SELECT, INSERT, TRIGGER ON TABLE `mydb`.* TO 'hostess';
GRANT EXECUTE ON procedure `mydb`.`inserisci_passeggero` TO 'hostess';
GRANT EXECUTE ON procedure `mydb`.`prenota_stanza` TO 'hostess';
```

```
GRANT EXECUTE ON procedure `mydb`.`viaggio_assegnato_hostess` TO 'hostess';

GRANT EXECUTE ON procedure `mydb`.`conferma_viaggio` TO 'hostess';

GRANT EXECUTE ON procedure `mydb`.`inserisci_cliente` TO 'hostess';

GRANT EXECUTE ON procedure `mydb`.`inserisci_costo` TO 'hostess';

SET SQL_MODE = '';

DROP USER IF EXISTS passeggero;

SET
SQL_MODE='ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO
_ZERO_DATE,ERROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION';

CREATE USER 'passeggero' IDENTIFIED BY 'passeggero';


GRANT SELECT, INSERT, TRIGGER, UPDATE, DELETE ON TABLE `mydb`.* TO 'passeggero';

GRANT EXECUTE ON procedure `mydb`.`stato_viaggio` TO 'passeggero';

GRANT EXECUTE ON procedure `mydb`.`info_visite_passeggero` TO 'passeggero';

SET SQL_MODE = '';

DROP USER IF EXISTS meccanico;

SET
SQL_MODE='ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO
_ZERO_DATE,ERROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION';

CREATE USER 'meccanico' IDENTIFIED BY 'meccanico';


GRANT EXECUTE ON procedure `mydb`.`lettura_contatore` TO 'meccanico';

GRANT EXECUTE ON procedure `mydb`.`prossima_revisione` TO 'meccanico';

GRANT EXECUTE ON procedure `mydb`.`inserisci_sessione_controllo` TO 'meccanico';

GRANT EXECUTE ON procedure `mydb`.`ricambi_disponibili` TO 'meccanico';

GRANT EXECUTE ON procedure `mydb`.`usa_ricambio` TO 'meccanico';

GRANT EXECUTE ON procedure `mydb`.`revisioni_da_effettuare` TO 'meccanico';

GRANT EXECUTE ON procedure `mydb`.`ordina_ricambio` TO 'meccanico';

SET SQL_MODE = '';

DROP USER IF EXISTS autista;

SET
SQL_MODE='ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO
_ZERO_DATE,ERROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION';

CREATE USER 'autista' IDENTIFIED BY 'autista';
```

```
GRANT EXECUTE ON procedure `mydb`.`orario_fermate` TO 'autista';

GRANT EXECUTE ON procedure `mydb`.`viaggio_assegnato_autista` TO 'autista';

GRANT EXECUTE ON procedure `mydb`.`info_visite_autista` TO 'autista';

GRANT EXECUTE ON procedure `mydb`.`calcola_distanza` TO 'autista';

GRANT EXECUTE ON procedure `mydb`.`cartine_programma` TO 'autista';

GRANT EXECUTE ON procedure `mydb`.`info_alberghi` TO 'autista';

SET SQL_MODE = '';

DROP USER IF EXISTS admin;

SET
SQL_MODE='ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO
_ZERO_DATE,ERROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION';

CREATE USER 'admin' IDENTIFIED BY 'admin';


GRANT EXECUTE ON procedure `mydb`.`aggiungi_employee` TO 'admin';

GRANT EXECUTE ON procedure `mydb`.`crea_utente` TO 'admin';


SET SQL_MODE=@OLD_SQL_MODE;

SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS;

SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS;


-- ----------------------------------------------------

-- Data for table `mydb`.`utenti`

-- ----------------------------------------------------

START TRANSACTION;

USE `mydb`;

INSERT INTO `mydb`.`utenti` (`username`, `password`, `ruolo`) VALUES ('mario.rossi',
'b036c299f278f684b8344755fadc403a', 'autista');

INSERT INTO `mydb`.`utenti` (`username`, `password`, `ruolo`) VALUES ('carla.carla',
'1fa4a2211b4e290f2a066de6b84187ec', 'hostess');

INSERT INTO `mydb`.`utenti` (`username`, `password`, `ruolo`) VALUES ('nus.nusnus', 'budicutie',
'passeggero');

INSERT INTO `mydb`.`utenti` (`username`, `password`, `ruolo`) VALUES ('omar.omar',
'037c53a1a1be94b8b2d9096b2d00ca65', 'meccanico');
```

```
INSERT INTO `mydb`.`utenti` (`username`, `password`, `ruolo`) VALUES ('kim.jong', '50c5b515b07342033448760f81ec07ac', 'amministratore');

INSERT INTO `mydb`.`utenti` (`username`, `password`, `ruolo`) VALUES ('luca.ranieri', 'ff377aff39a9345a9cca803fb5c5c081', 'autista');

INSERT INTO `mydb`.`utenti` (`username`, `password`, `ruolo`) VALUES ('luka.doncic', '2f3a4fccca6406e35bcf33e92dd93135', 'passeggero');


COMMIT;



-- -----------------------------------------------------

-- Data for table `mydb`.`hostess`

-- -----------------------------------------------------

START TRANSACTION;

USE `mydb`;

INSERT INTO `mydb`.`hostess` (`Telefono`, `nome`, `cognome`, `username`) VALUES ('333333333', 'carla', 'carla', 'carla.carla');


COMMIT;



-- -----------------------------------------------------

-- Data for table `mydb`.`programma`

-- -----------------------------------------------------

START TRANSACTION;

USE `mydb`;

INSERT INTO `mydb`.`programma` (`nome`, `minimo_partecipanti`, `quota`) VALUES ('spoleto-rosignano', 1, 150);

INSERT INTO `mydb`.`programma` (`nome`, `minimo_partecipanti`, `quota`) VALUES ('svizzera', 10, 100);


COMMIT;
```

```
-- -----------------------------------------------------

-- Data for table `mydb`.`viaggio`

-- -----------------------------------------------------

START TRANSACTION;

USE `mydb`;

INSERT INTO `mydb`.`viaggio` (`data_partenza`, `itinerario`, `numero_partecipanti`, `ritorno`, `costo_adulti`, `costo_bambini`, `data_cancellazione`, `stato`, `hostess`) VALUES ('2021-08-30', 'spoleto-rosignano', 0, '2021-09-06', 200, 170, NULL, 'organizzazione', NULL);

INSERT INTO `mydb`.`viaggio` (`data_partenza`, `itinerario`, `numero_partecipanti`, `ritorno`, `costo_adulti`, `costo_bambini`, `data_cancellazione`, `stato`, `hostess`) VALUES ('2021-09-04', 'svizzera', 0, '2021-09-06', 120, 70, NULL, 'organizzazione', NULL);


COMMIT;



-- -----------------------------------------------------

-- Data for table `mydb`.`cliente`

-- -----------------------------------------------------

START TRANSACTION;

USE `mydb`;

INSERT INTO `mydb`.`cliente` (`Telefono`, `indirizzo`, `nome`, `email`, `fax`) VALUES ('777777777', 'via gianni', 'gianni morandi', 'gianni@gianni.it', '06060606');


COMMIT;



-- -----------------------------------------------------

-- Data for table `mydb`.`passeggero`

-- -----------------------------------------------------

START TRANSACTION;

USE `mydb`;

INSERT INTO `mydb`.`passeggero` (`Prenotatore`, `nome`, `età`, `posto`, `username`, `data_viaggio`, `itinerario`, `data_prenotazione`, `data_invio_documenti`) VALUES ('777777777', 'luka doncic', '22', '10', 'luka.doncic', '2021-08-30', 'spoleto-rosignano', '2021-08-29', '2021-08-29');
```

```
COMMIT;


-- -----------------------------------------------------
-- Data for table `mydb`.`località`
-- -----------------------------------------------------
START TRANSACTION;
USE `mydb`;
INSERT INTO `mydb`.`località` (`nome`, `regione`, `stato`) VALUES ('spoleto', 'umbria', 'italia');
INSERT INTO `mydb`.`località` (`nome`, `regione`, `stato`) VALUES ('perugia', 'umbria', 'italia');
INSERT INTO `mydb`.`località` (`nome`, `regione`, `stato`) VALUES ('orvieto', 'umbria', 'italia');
INSERT INTO `mydb`.`località` (`nome`, `regione`, `stato`) VALUES ('grosseto', 'toscana', 'italia');
INSERT INTO `mydb`.`località` (`nome`, `regione`, `stato`) VALUES ('rosignano solvay', 'toscana', 'italia');
INSERT INTO `mydb`.`località` (`nome`, `regione`, `stato`) VALUES ('zurigo', 'zurigo', 'svizzera');

COMMIT;


-- -----------------------------------------------------
-- Data for table `mydb`.`descrizione`
-- -----------------------------------------------------
START TRANSACTION;
USE `mydb`;
INSERT INTO `mydb`.`descrizione` (`nome_località`, `regione_località`, `itinerario`, `giornata_arrivo`, `giornata_partenza`) VALUES ('spoleto', 'umbria', 'spoleto-rosignano', 1, 5);
INSERT INTO `mydb`.`descrizione` (`nome_località`, `regione_località`, `itinerario`, `giornata_arrivo`, `giornata_partenza`) VALUES ('perugia', 'umbria', 'spoleto-rosignano', 5, 6);
INSERT INTO `mydb`.`descrizione` (`nome_località`, `regione_località`, `itinerario`, `giornata_arrivo`, `giornata_partenza`) VALUES ('orvieto', 'umbria', 'spoleto-rosignano', 6, 8);
INSERT INTO `mydb`.`descrizione` (`nome_località`, `regione_località`, `itinerario`, `giornata_arrivo`, `giornata_partenza`) VALUES ('grosseto', 'toscana', 'spoleto-rosignano', 8, 9);
```

```
INSERT INTO `mydb`.`descrizione` (`nome_località`, `regione_località`, `itinerario`, `giornata_arrivo`, `giornata_partenza`) VALUES ('rosignano solvay', 'toscana', 'spoleto-rosignano', 9, 10);

INSERT INTO `mydb`.`descrizione` (`nome_località`, `regione_località`, `itinerario`, `giornata_arrivo`, `giornata_partenza`) VALUES ('zurigo', 'zurigo', 'svizzera', 1, 3);


COMMIT;



-- -----------------------------------------------------

-- Data for table `mydb`.`cartina`

-- -----------------------------------------------------

START TRANSACTION;

USE `mydb`;

INSERT INTO `mydb`.`cartina` (`Zona`, `nome_località`, `regione_località`, `livello_di_dettaglio`, `file`) VALUES ('nord', 'spoleto', 'umbria', '2', 'uhbvhijhnbnjikjn');

INSERT INTO `mydb`.`cartina` (`Zona`, `nome_località`, `regione_località`, `livello_di_dettaglio`, `file`) VALUES ('centro', 'spoleto', 'umbria', '3', 'tghbn');

INSERT INTO `mydb`.`cartina` (`Zona`, `nome_località`, `regione_località`, `livello_di_dettaglio`, `file`) VALUES ('est', 'rosignano solvay', 'toscana', '1', 'dfrdftgyh');


COMMIT;



-- -----------------------------------------------------

-- Data for table `mydb`.`luogo`

-- -----------------------------------------------------

START TRANSACTION;

USE `mydb`;

INSERT INTO `mydb`.`luogo` (`indirizzo`, `nome`, `nome_località`, `regione_località`) VALUES ('via otello', 'hotel uliassi', 'spoleto', 'umbria');

INSERT INTO `mydb`.`luogo` (`indirizzo`, `nome`, `nome_località`, `regione_località`) VALUES ('via delle ghiande', 'hotel monterosso', 'perugia', 'umbria');

INSERT INTO `mydb`.`luogo` (`indirizzo`, `nome`, `nome_località`, `regione_località`) VALUES ('via della noia', 'hotel baia del santo', 'orvieto', 'umbria');
```

```
INSERT INTO `mydb`.`luogo` (`indirizzo`, `nome`, `nome_località`, `regione_località`) VALUES ('viale arsenico', 'hotel il sodio', 'rosignano solvay', 'toscana');

INSERT INTO `mydb`.`luogo` (`indirizzo`, `nome`, `nome_località`, `regione_località`) VALUES ('via del mare', 'hotel l\'approdo', 'grosseto', 'toscana');

INSERT INTO `mydb`.`luogo` (`indirizzo`, `nome`, `nome_località`, `regione_località`) VALUES ('via del metallo pesante', 'fabbrica solvay', 'rosignano solvay', 'toscana');

INSERT INTO `mydb`.`luogo` (`indirizzo`, `nome`, `nome_località`, `regione_località`) VALUES ('via del santo graal', 'chiesa del santo', 'orvieto', 'umbria');

INSERT INTO `mydb`.`luogo` (`indirizzo`, `nome`, `nome_località`, `regione_località`) VALUES ('via dell\'orologio svizzero', 'neutraland', 'zurigo', 'zurigo');

INSERT INTO `mydb`.`luogo` (`indirizzo`, `nome`, `nome_località`, `regione_località`) VALUES ('via eth', 'eth university', 'zurigo', 'zurigo');


COMMIT;



-- -----------------------------------------------------

-- Data for table `mydb`.`distanza`

-- -----------------------------------------------------

START TRANSACTION;

USE `mydb`;

INSERT INTO `mydb`.`distanza` (`Primo_indirizzo`, `secondo_indirizzo`, `metri`) VALUES ('viale arsenico', 'via del mare', '100');

INSERT INTO `mydb`.`distanza` (`Primo_indirizzo`, `secondo_indirizzo`, `metri`) VALUES ('via del mare', 'via delle ghiande', '120');

INSERT INTO `mydb`.`distanza` (`Primo_indirizzo`, `secondo_indirizzo`, `metri`) VALUES ('via del mare', 'via otello', '50');


COMMIT;



-- -----------------------------------------------------

-- Data for table `mydb`.`albergo`

-- -----------------------------------------------------

START TRANSACTION;
```

```
USE `mydb`;

INSERT INTO `mydb`.`albergo` (`indirizzo`, `email`, `fax`, `Categoria`, `Telefono`) VALUES ('via otello', 'uliassi@uliassi.it', '01020304', '3', '06123456');

INSERT INTO `mydb`.`albergo` (`indirizzo`, `email`, `fax`, `Categoria`, `Telefono`) VALUES ('via delle ghiande', 'monterosso@monterosso.it', '25589658', '3', '06254478');

INSERT INTO `mydb`.`albergo` (`indirizzo`, `email`, `fax`, `Categoria`, `Telefono`) VALUES ('via della noia', 'santo@santo.it', '89654889', '4', '06987569');

INSERT INTO `mydb`.`albergo` (`indirizzo`, `email`, `fax`, `Categoria`, `Telefono`) VALUES ('viale arsenico', 'sodio@sodio.it', '26589665', '5', '06741921');

INSERT INTO `mydb`.`albergo` (`indirizzo`, `email`, `fax`, `Categoria`, `Telefono`) VALUES ('via del mare', 'approdo@approdo.it', '74185969', '2', '06854230');

INSERT INTO `mydb`.`albergo` (`indirizzo`, `email`, `fax`, `Categoria`, `Telefono`) VALUES ('via dell\'orologio svizzero', 'neutraland@neutraland.sw', '88563202', '5', '41052696');


COMMIT;



-- -----------------------------------------------------

-- Data for table `mydb`.`bene`

-- -----------------------------------------------------

START TRANSACTION;

USE `mydb`;

INSERT INTO `mydb`.`bene` (`indirizzo_bene`, `nome_bene`, `orario_apertura`, `telefono`) VALUES ('via del metallo pesante', 'fabbrica solvay', '8:00', '06000000');

INSERT INTO `mydb`.`bene` (`indirizzo_bene`, `nome_bene`, `orario_apertura`, `telefono`) VALUES ('via del santo graal', 'chiesa del santo', '7:00', '07419634');

INSERT INTO `mydb`.`bene` (`indirizzo_bene`, `nome_bene`, `orario_apertura`, `telefono`) VALUES ('via eth', 'eth university', '7:00', '89653215');


COMMIT;



-- -----------------------------------------------------

-- Data for table `mydb`.`visita`

-- -----------------------------------------------------
```

START TRANSACTION;

USE `mydb`;

INSERT INTO `mydb`.`visita` (`itinerario`, `nome_bene`, `indirizzo_bene`, `compresa`, `guidata`, `giornata`, `orario`) VALUES ('spoleto-rosignano', 'fabbrica solvay', 'via del metallo pesante', 'si', 'si', 9, '17:30');

INSERT INTO `mydb`.`visita` (`itinerario`, `nome_bene`, `indirizzo_bene`, `compresa`, `guidata`, `giornata`, `orario`) VALUES ('spoleto-rosignano', 'chiesa del santo', 'via del santo graal', 'no', 'no', 7, '16:00');

INSERT INTO `mydb`.`visita` (`itinerario`, `nome_bene`, `indirizzo_bene`, `compresa`, `guidata`, `giornata`, `orario`) VALUES ('svizzera', 'eth university', 'via eth', 'si', 'si', 1, '08:00');

COMMIT;

-- ---

-- Data for table `mydb`.`fermata`

-- ---

START TRANSACTION;

USE `mydb`;

INSERT INTO `mydb`.`fermata` (`indirizzo_luogo`, `orario_arrivo`, `orario_partenza`, `giornata`, `itinerario`, `giornata_partenza`) VALUES ('via otello', '15:30', '8:30', 1, 'spoleto-rosignano', '2');

INSERT INTO `mydb`.`fermata` (`indirizzo_luogo`, `orario_arrivo`, `orario_partenza`, `giornata`, `itinerario`, `giornata_partenza`) VALUES ('via otello', '21:30', '8:30', 2, 'spoleto-rosignano', '3');

INSERT INTO `mydb`.`fermata` (`indirizzo_luogo`, `orario_arrivo`, `orario_partenza`, `giornata`, `itinerario`, `giornata_partenza`) VALUES ('via otello', '21:30', '8:30', 3, 'spoleto-rosignano', '4');

INSERT INTO `mydb`.`fermata` (`indirizzo_luogo`, `orario_arrivo`, `orario_partenza`, `giornata`, `itinerario`, `giornata_partenza`) VALUES ('via otello', '21:30', '8:25', 4, 'spoleto-rosignano', '5');

INSERT INTO `mydb`.`fermata` (`indirizzo_luogo`, `orario_arrivo`, `orario_partenza`, `giornata`, `itinerario`, `giornata_partenza`) VALUES ('via delle ghiande', '20:00', '8:00', 5, 'spoleto-rosignano', '6');

INSERT INTO `mydb`.`fermata` (`indirizzo_luogo`, `orario_arrivo`, `orario_partenza`, `giornata`, `itinerario`, `giornata_partenza`) VALUES ('via della noia', '21:00', '7:30', 6, 'spoleto-rosignano', '7');

INSERT INTO `mydb`.`fermata` (`indirizzo_luogo`, `orario_arrivo`, `orario_partenza`, `giornata`, `itinerario`, `giornata_partenza`) VALUES ('via della noia', '20:00', '9:00', 7, 'spoleto-rosignano', '8');

INSERT INTO `mydb`.`fermata` (`indirizzo_luogo`, `orario_arrivo`, `orario_partenza`, `giornata`, `itinerario`, `giornata_partenza`) VALUES ('via del mare', '17:00', '7:00', 8, 'spoleto-rosignano', '9');

INSERT INTO `mydb`.`fermata` (`indirizzo_luogo`, `orario_arrivo`, `orario_partenza`, `giornata`, `itinerario`, `giornata_partenza`) VALUES ('viale arsenico', '23:00', '8:30', 9, 'spoleto-rosignano', '10');

INSERT INTO `mydb`.`fermata` (`indirizzo_luogo`, `orario_arrivo`, `orario_partenza`, `giornata`, `itinerario`, `giornata_partenza`) VALUES ('via del metallo pesante', '17:00', '19:00', 9, 'spoleto-rosignano', '9');

COMMIT;

-- ---

-- Data for table `mydb`.`modello_pullman`

-- ---

START TRANSACTION;

USE `mydb`;

INSERT INTO `mydb`.`modello_pullman` (`costruttore`, `nome`, `velocità_massima`, `serbatoio`, `cavalli`, `numero_posti`) VALUES ('mercedes', 'voyager', '200', '250', '300', 100);

INSERT INTO `mydb`.`modello_pullman` (`costruttore`, `nome`, `velocità_massima`, `serbatoio`, `cavalli`, `numero_posti`) VALUES ('iveco', 'evadys', '200', '230', '350', 120);

INSERT INTO `mydb`.`modello_pullman` (`costruttore`, `nome`, `velocità_massima`, `serbatoio`, `cavalli`, `numero_posti`) VALUES ('man', 'lions', '200', '220', '330', 140);

COMMIT;

-- ---

-- Data for table `mydb`.`pullman`

-- ---

START TRANSACTION;

USE `mydb`;

INSERT INTO `mydb`.`pullman` (`targa`, `contatore`, `data_immatricolazione`, `costruttore`, `nome_modello`) VALUES ('ca320dc', 50000, '2005-11-23', 'mercedes', 'voyager');

INSERT INTO `mydb`.`pullman` (`targa`, `contatore`, `data_immatricolazione`, `costruttore`, `nome_modello`) VALUES ('ca321dc', 25000, '2005-11-23', 'mercedes', 'voyager');

INSERT INTO `mydb`.`pullman` (`targa`, `contatore`, `data_immatricolazione`, `costruttore`, `nome_modello`) VALUES ('ca322dc', 43000, '2005-10-25', 'man', 'lions');

```
INSERT INTO `mydb`.`pullman` (`targa`, `contatore`, `data_immatricolazione`, `costruttore`, `nome_modello`) VALUES ('ca323dc', 150000, '2003-02-09', 'iveco', 'evadys');


COMMIT;



-- -----------------------------------------------------

-- Data for table `mydb`.`autista`

-- -----------------------------------------------------

START TRANSACTION;

USE `mydb`;

INSERT INTO `mydb`.`autista` (`Telefono`, `nome`, `cognome`, `username`) VALUES ('369258147', 'mario', 'rossi', 'mario.rossi');

INSERT INTO `mydb`.`autista` (`Telefono`, `nome`, `cognome`, `username`) VALUES ('339864478', 'luca', 'ranieri', 'luca.ranieri');


COMMIT;



-- -----------------------------------------------------

-- Data for table `mydb`.`trasporto`

-- -----------------------------------------------------

START TRANSACTION;

USE `mydb`;

INSERT INTO `mydb`.`trasporto` (`autista`, `pullman`, `data_partenza`, `itinerario`) VALUES ('369258147', 'ca320dc', '2021-08-30', 'spoleto-rosignano');

INSERT INTO `mydb`.`trasporto` (`autista`, `pullman`, `data_partenza`, `itinerario`) VALUES ('339864478', 'ca321dc', '2021-09-04', 'svizzera');


COMMIT;



-- -----------------------------------------------------

-- Data for table `mydb`.`controllo_ordinario`
```

```
-- -----------------------------------------------------
START TRANSACTION;
USE `mydb`;
INSERT INTO `mydb`.`controllo_ordinario` (`tipo`, `intervallo`) VALUES ('motore', 1000);
INSERT INTO `mydb`.`controllo_ordinario` (`tipo`, `intervallo`) VALUES ('convergenza', 5000);
INSERT INTO `mydb`.`controllo_ordinario` (`tipo`, `intervallo`) VALUES ('freni', 20000);
INSERT INTO `mydb`.`controllo_ordinario` (`tipo`, `intervallo`) VALUES ('pneumatici', 40000);
INSERT INTO `mydb`.`controllo_ordinario` (`tipo`, `intervallo`) VALUES ('batteria', 30000);
INSERT INTO `mydb`.`controllo_ordinario` (`tipo`, `intervallo`) VALUES ('sospensioni', 60000);
INSERT INTO `mydb`.`controllo_ordinario` (`tipo`, `intervallo`) VALUES ('luci', 70000);
INSERT INTO `mydb`.`controllo_ordinario` (`tipo`, `intervallo`) VALUES ('carrozzeria', 100000);

COMMIT;


-- -----------------------------------------------------
-- Data for table `mydb`.`manutenzione`
-- -----------------------------------------------------
START TRANSACTION;
USE `mydb`;
INSERT INTO `mydb`.`manutenzione` (`pullman`, `controllo_ordinario`, `prossimo_controllo`) VALUES ('ca320dc', 'motore', 0);
INSERT INTO `mydb`.`manutenzione` (`pullman`, `controllo_ordinario`, `prossimo_controllo`) VALUES ('ca320dc', 'convergenza', 0);

COMMIT;


-- -----------------------------------------------------
-- Data for table `mydb`.`sessione_controllo`
-- -----------------------------------------------------
START TRANSACTION;
USE `mydb`;
```

```
INSERT INTO `mydb`.`sessione_controllo` (`data_inizio`, `pullman`, `data_fine`, `motivo`) VALUES ('2021-08-28', 'ca320dc', '2021-08-28', NULL);


COMMIT;



-- -----------------------------------------------------

-- Data for table `mydb`.`tipo_sessione`

-- -----------------------------------------------------

START TRANSACTION;

USE `mydb`;

INSERT INTO `mydb`.`tipo_sessione` (`controllo_ordinario`, `data_controllo`, `pullman`) VALUES ('motore', '2021-08-28', 'ca320dc');


COMMIT;



-- -----------------------------------------------------

-- Data for table `mydb`.`ricambio`

-- -----------------------------------------------------

START TRANSACTION;

USE `mydb`;

INSERT INTO `mydb`.`ricambio` (`nome`, `disponibile`, `codice`, `ordinato`) VALUES ('testata', 1, '00001', 0);


COMMIT;



-- -----------------------------------------------------

-- Data for table `mydb`.`meccanico`

-- -----------------------------------------------------

START TRANSACTION;

USE `mydb`;
```

```
INSERT INTO `mydb`.`meccanico` (`telefono`, `nome`, `cognome`, `username`) VALUES ('555555555', 'omar', 'omar', 'omar.omar');


COMMIT;


USE `mydb`;


DELIMITER $$


USE `mydb`$$
DROP TRIGGER IF EXISTS `mydb`.`viaggio_BEFORE_UPDATE` $$
USE `mydb`$$
CREATE DEFINER = CURRENT_USER TRIGGER `mydb`.`viaggio_BEFORE_UPDATE` BEFORE UPDATE ON `viaggio` FOR EACH ROW
BEGIN
declare counter INT;
select count(*) from `viaggio` V
where
        NEW.hostess = V.hostess
        and NEW.data_partenza < V.ritorno
   and NEW.ritorno > V.data_partenza into counter;
if counter > 0 then
        signal sqlstate "45000";
end if;
END$$



USE `mydb`$$
DROP TRIGGER IF EXISTS `mydb`.`viaggio_BEFORE_UPDATE_1` $$
USE `mydb`$$
CREATE DEFINER = CURRENT_USER TRIGGER `mydb`.`viaggio_BEFORE_UPDATE_1` BEFORE UPDATE ON `viaggio` FOR EACH ROW
BEGIN
```

```
declare counter INT;
select count(*) from `viaggio` V
where NEW.stato = 'cancellato' and NEW.hostess is not null or NEW.stato = 'confermato' and NEW.data_cancellazione is not null into counter;

if counter <> 0 then
	signal sqlstate "45000";
end if;
END$$


USE `mydb`$$
DROP TRIGGER IF EXISTS `mydb`.`viaggio_BEFORE_UPDATE_2` $$
USE `mydb`$$
CREATE DEFINER = CURRENT_USER TRIGGER `mydb`.`viaggio_BEFORE_UPDATE_2` BEFORE UPDATE ON `viaggio` FOR EACH ROW
BEGIN
declare counter INT;
if NEW.stato = 'confermato'  then
select count(*) from `programma` P
where
		P.nome = NEW.itinerario
	and P.minimo_partecipanti > NEW.numero_partecipanti into counter;

if counter <> 0 then
	signal sqlstate "45000";
end if;
end if;
END$$


USE `mydb`$$
```

```
DROP TRIGGER IF EXISTS `mydb`.`distanza_BEFORE_INSERT` $$
USE `mydb`$$
CREATE DEFINER = CURRENT_USER TRIGGER `mydb`.`distanza_BEFORE_INSERT` BEFORE INSERT ON `distanza` FOR EACH ROW
BEGIN
	if NEW.primo_indirizzo = NEW.secondo_indirizzo then
		signal sqlstate "45000";
   end if;
END$$



USE `mydb`$$
DROP TRIGGER IF EXISTS `mydb`.`distanza_BEFORE_INSERT_1` $$
USE `mydb`$$
CREATE DEFINER = CURRENT_USER TRIGGER `mydb`.`distanza_BEFORE_INSERT_1` BEFORE INSERT ON `distanza` FOR EACH ROW
BEGIN

   declare counter int;


	select count(*) from `distanza` D
   where
			NEW.primo_indirizzo = D.secondo_indirizzo
		and D.primo_indirizzo = NEW.secondo_indirizzo into counter;


	if counter > 0 then
		signal sqlstate "45000";
	end if;

END$$



USE `mydb`$$
```

```
DROP TRIGGER IF EXISTS `mydb`.`visita_BEFORE_INSERT` $$
USE `mydb`$$
CREATE DEFINER = CURRENT_USER TRIGGER `mydb`.`visita_BEFORE_INSERT` BEFORE INSERT ON `visita` FOR EACH ROW
BEGIN
declare counter INT;
SELECT
   COUNT(*)
FROM
   `descrizione` D,
   `luogo` L
WHERE
   L.indirizzo = NEW.indirizzo_bene
        AND L.nome_località = D.nome_località
        AND L.regione_località = D.regione_località
        AND NEW.giornata >= D.giornata_arrivo
        AND NEW.giornata <= giornata_partenza INTO counter;
if counter = 0 then
        signal sqlstate "45000";
end if;
END$$



USE `mydb`$$
DROP TRIGGER IF EXISTS `mydb`.`fermata_BEFORE_INSERT` $$
USE `mydb`$$
CREATE DEFINER = CURRENT_USER TRIGGER `mydb`.`fermata_BEFORE_INSERT` BEFORE INSERT ON `fermata` FOR EACH ROW
BEGIN
declare counter INT;
SELECT
   COUNT(*)
FROM
```

```
   `albergo` A
WHERE
   A.indirizzo = NEW.indirizzo_luogo INTO counter;

if counter > 1 then

   select count(*) from `descrizione` D, `luogo` L
   where D.itinerario = NEW.itinerario
   and D.nome_località = L.nome_località
   and D.regione_località = L.regione_località
   and NEW.indirizzo_luogo = L.indirizzo
   and D.giornata_arrivo <> D.giornata_partenza into counter;

   if counter = 0 then
                signal sqlstate "45000";
        end if;
end if;
END$$



USE `mydb`$$
DROP TRIGGER IF EXISTS `mydb`.`fermata_BEFORE_INSERT_1` $$
USE `mydb`$$
CREATE DEFINER = CURRENT_USER TRIGGER `mydb`.`fermata_BEFORE_INSERT_1`
BEFORE INSERT ON `fermata` FOR EACH ROW
BEGIN

        declare counter int;

        select count(*) from `albergo` A
   where
                new.indirizzo_luogo = A.indirizzo into counter;
```

```
   if counter = 0 then
                if new.giornata <> new.giornata_partenza then
                        signal sqlstate "45000";
      end if;
         end if;
END$$



USE `mydb`$$
DROP TRIGGER IF EXISTS `mydb`.`foto_BEFORE_INSERT` $$
USE `mydb`$$
CREATE DEFINER = CURRENT_USER TRIGGER `mydb`.`foto_BEFORE_INSERT` BEFORE INSERT ON `foto` FOR EACH ROW
BEGIN
        if new.nome_località is not null and new.pullman is not null then
                signal sqlstate "45000" set message_text = 'la foto puo raffigurare sia un pullamn, sia una località; uno dei due valori deve essere null';
   end if;
END$$



USE `mydb`$$
DROP TRIGGER IF EXISTS `mydb`.`trasporto_BEFORE_INSERT` $$
USE `mydb`$$
CREATE DEFINER = CURRENT_USER TRIGGER `mydb`.`trasporto_BEFORE_INSERT` BEFORE INSERT ON `trasporto` FOR EACH ROW
BEGIN
declare n_autisti INT;
declare n_pullman INT;

select count(`autista`) from `trasporto` T
where
```

```
        T.itinerario = NEW.itinerario
        and T.data_partenza = NEW.data_partenza into n_autisti;

select count(`pullman`) from `trasporto` T
where
        T.itinerario = NEW.itinerario
        and T.data_partenza = NEW.data_partenza into n_pullman;

if n_autisti <> n_pullman then
        signal sqlstate "45000";
end if;
END$$



USE `mydb`$$
DROP TRIGGER IF EXISTS `mydb`.`trasporto_BEFORE_INSERT_1` $$
USE `mydb`$$
CREATE DEFINER = CURRENT_USER TRIGGER `mydb`.`trasporto_BEFORE_INSERT_1`
BEFORE INSERT ON `trasporto` FOR EACH ROW
BEGIN
declare n_autisti INT;
declare rit DATE;

select `ritorno` from `viaggio` V
where
                NEW.itinerario = V.itinerario
        and NEW.data_partenza = V.data_partenza into rit;

select count(`autista`) from `trasporto` T, `viaggio` V
where
        T.itinerario = V.itinerario
        and T.data_partenza = V.data_partenza
```

```
   and T.autista = NEW.autista
   and NEW.data_partenza < V.ritorno
   and rit > V.data_partenza into n_autisti;



if n_autisti <> 0 then
        signal sqlstate "45000" set message_text = 'le date dei viaggi si sovrappongono per l autista';
end if;
END$$



USE `mydb`$$
DROP TRIGGER IF EXISTS `mydb`.`trasporto_BEFORE_INSERT_2` $$
USE `mydb`$$
CREATE DEFINER = CURRENT_USER TRIGGER `mydb`.`trasporto_BEFORE_INSERT_2`
BEFORE INSERT ON `trasporto` FOR EACH ROW
BEGIN
        declare n_pullman INT;
        declare rit DATE;

   select `ritorno` from `viaggio` V
        where
                        NEW.itinerario = V.itinerario
                and NEW.data_partenza = V.data_partenza into rit;

        select count(`pullman`) from `trasporto` T, `viaggio` V
        where
                        T.itinerario = V.itinerario
                and T.data_partenza = V.data_partenza
                and T.pullman = NEW.pullman
                and NEW.data_partenza < V.ritorno
                and rit > V.data_partenza into n_pullman;
```

```
        if n_pullman <> 0 then
        signal sqlstate "45000" set message_text = 'le date dei viaggi si sovrappongono per il pullman';
        end if;
END$$



USE `mydb`$$
DROP TRIGGER IF EXISTS `mydb`.`sessione_controllo_BEFORE_INSERT` $$
USE `mydb`$$
CREATE DEFINER = CURRENT_USER TRIGGER `mydb`.`sessione_controllo_BEFORE_INSERT` BEFORE INSERT ON `sessione_controllo` FOR EACH ROW
BEGIN
declare counter1 INT;
declare counter2 INT;
if NEW.motivo = NULL then

        SELECT
                `contatore`
        FROM
                `pullman` P
        WHERE
                P.targa = NEW.pullman INTO counter1;

        SELECT
                M.prossimo_controllo
        FROM
                `manutenzione` M,
                `tipo_sessione` T
        WHERE
                T.data_controllo = NEW.data_inizio
                AND T.pullman = NEW.pullman
```

```
                AND T.controllo_ordinario = M.controllo_ordinario INTO counter2;

        if counter1 < counter2 then
                signal sqlstate "45000";
        end if;
end if;
END$$


DELIMITER ;
```

Codice del Front-End

MAIN.C:

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

#include "defines.h"

typedef enum {
        AUTISTA = 1,
        HOSTESS,
        MECCANICO,
        PASSEGGERO,
        AMMINISTRATORE,
        FAILED_LOGIN
} role_t;


struct configuration conf;
```

```
static MYSQL *conn;


static void lista_viaggi(MYSQL *conn){
	MYSQL_STMT *prepared_stmt;

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

	if (mysql_stmt_execute(prepared_stmt) != 0) {
		print_stmt_error (prepared_stmt, "An error occurred while searching travels.");
		goto out;
	}

	// Dump the result set
	dump_result_set(conn, prepared_stmt, "\nDi seguito la lista dei viaggi:");


	//per consumare il result set
	if (mysql_stmt_next_result(prepared_stmt) > 0){
		finish_with_stmt_error(conn, prepared_stmt, "Unexpected condition", true);
	}

 out:
	mysql_stmt_close(prepared_stmt);
}


static void run_as_guest(MYSQL *conn){
```

```
        char options[2] = {'1','2'};
        char r;

        while(true){
                printf("\ninserire 1 o 2: \n1) visualizzare i viaggi della compagnia \n2) Quit");

                r = multiChoice("\nSeleziona una scelta", options, 2);


                switch(r) {
                        case '1':
                                lista_viaggi(conn);
                                break;
                        case '2':
                                return;
                        default:
                                fprintf(stderr, "Invalid condition at %s:%d\n", __FILE__, __LINE__);
                                abort();
                }
        }

}

static role_t attempt_login(MYSQL *conn, char *username, char *password) {
        MYSQL_STMT *login_procedure;

        MYSQL_BIND param[3]; // Used both for input and output
        int role = 0;

        if(!setup_prepared_stmt(&login_procedure, "call login(?, ?, ?)", conn)) {
```

```
        print_stmt_error(login_procedure, "Unable to initialize login statement\n");
        goto err2;
}

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

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

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

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

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

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

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

```
        param[0].buffer_type = MYSQL_TYPE_LONG; // OUT
        param[0].buffer = &role;
        param[0].buffer_length = sizeof(role);

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

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

        mysql_stmt_close(login_procedure);
        return role;

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

int main(void) {
        role_t role;


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

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

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

	char options[2] = {'1','2'};
	char r;
	int guest;

	printf("\nscegliere se si vuole: \n1) accedere tramite username e password \n2) continuare come ospite");

	r = multiChoice("\nSeleziona una scelta", options, 2);


	switch(r) {
		case '1':
			guest = 0;
			break;
		case '2':
			guest = 1;
			break;
		default:
			fprintf(stderr, "Invalid condition at %s:%d\n", __FILE__, __LINE__);
```

```
                abort();
        }

        if(guest == 0){
                printf("Username: ");
                getInput(128, conf.username, false);
                printf("Password: ");
                getInput(128, conf.password, true);

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


                switch(role) {


                        case HOSTESS:
                                run_as_hostess(conn);
                                break;

                        case AUTISTA:
                                run_as_autista(conn);
                                break;

                        case PASSEGGERO:
                                run_as_passeggero(conn);
                                break;

                        case MECCANICO:
                                run_as_meccanico(conn);
                                break;

                        case AMMINISTRATORE:
```

```
                        run_as_amministratore(conn);
                        break;

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

                default:
                        fprintf(stderr, "Invalid condition at %s:%d\n", __FILE__, __LINE__);
                        abort();
            }
        }
        else if(guest == 1){
            run_as_guest(conn);
        }
        printf("Bye!\n");

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

ADMINISTRATOR.C:

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

#include "defines.h"


static void add_employee(MYSQL *conn, int ruolo)
```

```
{
        MYSQL_STMT *prepared_stmt;
        MYSQL_BIND param[6];

        // Input for the registration routine
        char tel[16];
        char name[46];
        char surname[46];
        char username[46];
        char password[46];
        char *r[3] = {"hostess", "meccanico", "autista"};

        // Get the required information
        printf("\n %s telephone number: ", r[ruolo]);
        getInput(16, tel, false);
        printf("\n%s name: ", r[ruolo]);
        getInput(46, name, false);
        printf("%s surname: ", r[ruolo]);
        getInput(46, surname, false);
        printf("%s username: ", r[ruolo]);
        getInput(46, username, false);
        printf("create a password for the %s: ", r[ruolo]);
        getInput(46, password, false);
        // Prepare stored procedure call
        if(!setup_prepared_stmt(&prepared_stmt, "call aggiungi_employee(?, ?, ?, ?, ?, ?)", conn)) {
                finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize employee insertion statement\n", false);
        }

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

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

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

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

        param[3].buffer_type = MYSQL_TYPE_VAR_STRING; // OUT
        param[3].buffer = username;
        param[3].buffer_length = strlen(username);

        param[4].buffer_type = MYSQL_TYPE_VAR_STRING; // OUT
        param[4].buffer = password;
        param[4].buffer_length = strlen(password);

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

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

        // Run procedure
        if (mysql_stmt_execute(prepared_stmt) != 0) {
                print_stmt_error(prepared_stmt, "An error occurred while adding the employee.");
```

```
        } else{
                printf("%s correctly added \n", r[ruolo]);
        }

        mysql_stmt_close(prepared_stmt);
}

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

        // Input for the registration routine
        char username[46];
        char password[46];
        char *ruolo = "amministratore";

        // Get the required information
        printf("\nUsername: ");
        getInput(46, username, false);
        printf("password: ");
        getInput(46, password, true);



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

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

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

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

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

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

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

	mysql_stmt_close(prepared_stmt);
}



void run_as_amministratore(MYSQL *conn)
{
```

```
char options[5] = {'1', '2', '3', '4', '5'};
char op;

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

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

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

while(true) {
        printf("\033[2J\033[H");
        printf("*** What should I do for you? ***\n\n");
        printf("1) Add new hostess\n");
        printf("2) Add new mechanic\n");
        printf("3) Add new driver\n");
        printf("4) Add new administrator\n");
        printf("5) Quit\n");

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

        switch(op) {
                case '1':
                        add_employee(conn, 0);
                        break;
                case '2':
                        add_employee(conn, 1);
```

```
                        break;
                case '3':
                        add_employee(conn, 2);
                        break;
                case '4':
                        create_administrator(conn);
                        break;
                case '5':
                        return;

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

        getchar();
    }
}
```

AUTISTA.C:

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

#include "defines.h"

#define MAX_DEGREES 128

struct localita {
    char nome[46];
```

```
        char regione[46];
};

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

        // Input for info_visite routine
        char itinerario[46];

        printf("\ninserire l'itinerario: ");
        getInput(46, itinerario, false);



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

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

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



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

        // Run procedure
```

```
	if (mysql_stmt_execute(prepared_stmt) != 0) {
		print_stmt_error (prepared_stmt, "An error occurred while searching visits' information.");
		goto out;
	}

	// Dump the result set
	dump_result_set(conn, prepared_stmt, "\nDi seguito le informazioni sulle visite:");



	//per consumare il result set
	if (mysql_stmt_next_result(prepared_stmt) > 0){
		finish_with_stmt_error(conn, prepared_stmt, "Unexpected condition", true);
	}

 out:
	mysql_stmt_close(prepared_stmt);
}

static size_t parse_localita(MYSQL *conn, MYSQL_STMT *stmt, struct localita **ret) {
	int status;
	size_t row = 0;
	MYSQL_BIND param[4];

	char nome[46];
	char regione[46];

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

```
	*ret = malloc(mysql_stmt_num_rows(stmt) * sizeof(struct localita));

	memset(param, 0, sizeof(param));
	param[0].buffer_type = MYSQL_TYPE_VAR_STRING;
	param[0].buffer = nome;
	param[0].buffer_length = 46;

	param[1].buffer_type = MYSQL_TYPE_VAR_STRING;
	param[1].buffer = regione;
	param[1].buffer_length = 46;

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

	/* assemble loc general information */
	while (true) {
		status = mysql_stmt_fetch(stmt);

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

		strcpy((*ret)[row].nome, nome);
		strcpy((*ret)[row].regione, regione);

		row++;
	}

	return row;
}
```

```
static void cartine(MYSQL *conn) {
	MYSQL_STMT *prepared_stmt;
	MYSQL_BIND param[1];
	int status;
	bool first = true;
	struct localita *loc;
	size_t localita = 0;
	char header[512];

	char itinerario[46];

	printf("\ninserire l'itinerario: ");
	getInput(46, itinerario, false);

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

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

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

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

	// Run procedure
```

```
        if (mysql_stmt_execute(prepared_stmt) != 0) {
                print_stmt_error(prepared_stmt, "An error occurred while retrieving the career report.");
                goto out;
        }

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

                if(first) {
                        parse_localita(conn, prepared_stmt, &loc);
                        first = false;
                } else {
                        sprintf(header, "\nCartine disponibili per la seguente località: \nNome: %s\nregione: %s\n", loc[localita].nome, loc[localita].regione);
                        dump_result_set(conn, prepared_stmt, header);
                        localita++;
                }

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

        } while (status == 0);

    out:
        mysql_stmt_close(prepared_stmt);
```

```
}

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

	// Input for viaggi_assegnati routine



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

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

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

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

	// Run procedure
	if (mysql_stmt_execute(prepared_stmt) != 0) {
		print_stmt_error (prepared_stmt, "An error occurred while searching assigned travels.");
		goto out;
	}
```

```
        // Dump the result set
        dump_result_set(conn, prepared_stmt, "\nDi seguito i viaggi e il pullman assegnati:");



        //per consumare il result set
        if (mysql_stmt_next_result(prepared_stmt) > 0){
                finish_with_stmt_error(conn, prepared_stmt, "Unexpected condition", true);
        }

 out:
        mysql_stmt_close(prepared_stmt);
}

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

        // Input for orario_fermate routine
        char itinerario[46];
        char temp[10];
        int giornata;

        printf("\ninserire l'itinerario: ");
        getInput(46, itinerario, false);

        printf("\ninserire la giornata di interesse: ");
        getInput(10, temp, false);

        giornata = atoi(temp);

        if(!setup_prepared_stmt(&prepared_stmt, "call orario_fermate(?,?)", conn)) {
```

```
        finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize orario_fermate statement\n", false);
    }

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

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

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

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

    // Run procedure
    if (mysql_stmt_execute(prepared_stmt) != 0) {
        print_stmt_error (prepared_stmt, "An error occurred while searching stops' time.");
        goto out;
    }

    // Dump the result set
    dump_result_set(conn, prepared_stmt, "\nDi seguito l'orario delle fermate per la giornata selezionata:");


    //per consumare il result set
    if (mysql_stmt_next_result(prepared_stmt) > 0){
```

```
            finish_with_stmt_error(conn, prepared_stmt, "Unexpected condition", true);
        }

 out:
        mysql_stmt_close(prepared_stmt);
}

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

        // Input for info_alberghi routine
        char itinerario[46];

        printf("\ninserire l'itinerario: ");
        getInput(46, itinerario, false);



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

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

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



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

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

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

        // Dump the result set
        dump_result_set(conn, prepared_stmt, "\nDi seguito le informazioni sugli hotel in cui si alloggerà durante il viaggio:");


        //per consumare il result set
        if (mysql_stmt_next_result(prepared_stmt) > 0){
            finish_with_stmt_error(conn, prepared_stmt, "Unexpected condition", true);
        }

 out:
        mysql_stmt_close(prepared_stmt);
}

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

        // Input for distanze routine
        char indirizzo_1[46];
        char indirizzo_2[46];

        printf("\ninserire il primo indirizzo: ");
```

```
        getInput(46, indirizzo_1, false);

        printf("\ninserire il secondo indirizzo: ");
        getInput(46, indirizzo_2, false);



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

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

        param[0].buffer_type = MYSQL_TYPE_VAR_STRING;
        param[0].buffer = indirizzo_1;
        param[0].buffer_length = strlen(indirizzo_1);

        param[1].buffer_type = MYSQL_TYPE_VAR_STRING;
        param[1].buffer = indirizzo_2;
        param[1].buffer_length = strlen(indirizzo_2);

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

        // Run procedure
        if (mysql_stmt_execute(prepared_stmt) != 0) {
                print_stmt_error (prepared_stmt, "An error occurred while searching distance between two addresses.");
                goto out;
        }
```

```
        // Dump the result set
        dump_result_set(conn, prepared_stmt, "\nDi seguito la distanza richiesta:");



        //per consumare il result set
        if (mysql_stmt_next_result(prepared_stmt) > 0){
                finish_with_stmt_error(conn, prepared_stmt, "Unexpected condition", true);
        }

 out:
        mysql_stmt_close(prepared_stmt);
}




void run_as_autista(MYSQL *conn) {
        char options[7] = {'1','2','3','4','5','6','7'};
        char op;

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

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

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

```
while(true) {
        printf("\033[2J\033[H");
        printf("*** What should I do for you? ***\n\n");
        printf("1) ricerca viaggi assegnati \n");
        printf("2) informazioni fermate del giorno \n");
        printf("3) informazioni sulle visite di un viaggio \n");
        printf("4) cerca distanza tra due luoghi \n");
        printf("5) cerca le informazioni degli alloggi per un certo viaggio\n");
        printf("6) cerca le cartine disponibili per un certo viaggio\n");
        printf("7) Quit\n");

        op = multiChoice("Select an option", options, 7);

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

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

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

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

                case '5':
```

```
                        info_alberghi(conn);
                        break;

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

                case '7':
                        return;

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

        getchar();
    }
}
```

HOSTESS:

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

#include "defines.h"

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

```
// Input for viaggi_assegnati routine



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

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

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

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

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

// Dump the result set
dump_result_set(conn, prepared_stmt, "\nDi seguito i viaggi assegnati:");



//per consumare il result set
```

```
        if (mysql_stmt_next_result(prepared_stmt) > 0){
                finish_with_stmt_error(conn, prepared_stmt, "Unexpected condition", true);
        }

 out:
        mysql_stmt_close(prepared_stmt);
}

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

        // Input for inserisci cliente routine
        char telefono[16];
        char indirizzo[46];
        char nome[46];
        char email[46];
        char fax[46];


        printf("\ninserire il numero di telefono del cliente: ");
        getInput(16, telefono, false);

        printf("\ninserire un indirizzo del cliente: ");
        getInput(46, indirizzo, false);

        printf("\ninserire il nome del cliente: ");
        getInput(46, nome, false);

        printf("\ninserire la mail del cliente: ");
        getInput(46, email, false);
```

```
        printf("\ninserire il fax del cliente: ");
        getInput(16, fax, false);



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

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

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

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

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

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

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

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

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

	printf("\ncliente inserito correttamente\n");



 out:
	mysql_stmt_close(prepared_stmt);
}

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

	// Input for inserisci_passeggero routine
	char telefono[16];
	char nome[46];
	char eta[3];
	char posto[4];
	char username[46];
	MYSQL_TIME data;
	char itinerario[46];
	MYSQL_TIME oggi;
```

```
int n;
char password[46];

printf("\ninserire il numero di telefono del cliente prenotante: ");
getInput(16, telefono, false);

printf("\ninserire il nome del passeggero: ");
getInput(46, nome, false);

printf("\ninserire l'età del passeggero: ");
getInput(3, eta, false);

printf("\ninserire un posto per il passeggero: ");
getInput(4, posto, false);

printf("\ninserire un username per il passeggero: ");
getInput(46, username, false);

printf("\ninserire una password per il passeggero: ");
getInput(46, password, false);

char temp[10];

memset(&data, 0, sizeof(data));

printf("\ninserire ora la data del viaggio in cui si vole inserire il passeggero");

G:
printf("\ninserire il giorno del viaggio, da 1 a 31: ");
getInput(10, temp, false);
n = atoi(temp);
if(n < 1 || n > 31){
```

```
        printf("\ngiorno non valido, riprovare: ");
        goto G;
}

data.day = n;

M:
printf("\ndigitare il mese del viaggio, da 1 a 12: ");
getInput(10, temp, false);
n = atoi(temp);
if(n < 1 || n > 12){
        printf("\nmese non valido, riprovare: ");
        goto M;
}

data.month = n;

A:
printf("\ninserire l'anno del viaggio, da 2000 a 2022: ");
getInput(10, temp, false);
n = atoi(temp);
if(n < 2000 || n > 2022){
        printf("\nanno non valido, riprovare: ");
        goto A;
}

data.year = n;

printf("\ndigitare il nome del viaggio in cui si vuole inserire il passeggero: ");
getInput(46, itinerario, false);
```

```
        struct tm *tempo;
  time_t t;
  time(&t);
  tempo = localtime(&t);

  oggi.day = tempo->tm_mday;
        oggi.month = tempo->tm_mon + 1;
        oggi.year = tempo->tm_year + 1900;

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

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

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

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

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

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

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

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

param[6].buffer_type = MYSQL_TYPE_DATE;
param[6].buffer = &data;
param[6].buffer_length = sizeof(data);

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

param[8].buffer_type = MYSQL_TYPE_DATE;
param[8].buffer = &oggi;
param[8].buffer_length = sizeof(oggi);

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

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

```
        printf("\npasseggero inserito correttamente\n");



 out:
        mysql_stmt_close(prepared_stmt);
}

static void prenota_stanza(MYSQL *conn) {
        MYSQL_STMT *prepared_stmt;
        MYSQL_BIND param[4];
        char options[7] = {'1','2','3','4','5','6','7'};
        char r;

        // Input for prenota_stanza routine
        char indirizzo[46];
        MYSQL_TIME data;
        char itinerario[46];
        char *stanze[7] = {"singola", "doppia", "tripla", "quadrupla", "suite", "doppia con 3° letto", "doppia con 4° letto"};
        char *tipo;
        int n;




        printf("\ninserire l'indirizzo dell'albergo: ");
        getInput(46, indirizzo, false);

        printf("\ninserire l'itinerario: ");
        getInput(46, itinerario, false);



        char temp[10];
```

```
memset(&data, 0, sizeof(data));

printf("\ninserire ora la data di partenza del viaggio");

G:
printf("\ninserire il giorno di partenza del viaggio, da 1 a 31: ");
getInput(10, temp, false);
n = atoi(temp);
if(n < 1 || n > 31){
        printf("\ngiorno non valido, riprovare: ");
        goto G;
}

data.day = n;

M:
printf("\ndigitare il mese di partenza del viaggio, da 1 a 12: ");
getInput(10, temp, false);
n = atoi(temp);
if(n < 1 || n > 12){
        printf("\nmese non valido, riprovare: ");
        goto M;
}

data.month = n;

A:
printf("\ninserire l'anno di partenza del viaggio, da 2000 a 2022: ");
getInput(10, temp, false);
n = atoi(temp);
if(n < 2000 || n > 2022){
```

```
            printf("\nanno non valido, riprovare: ");
            goto A;
        }

        data.year = n;


        printf("\ninserire un numero da 1 a 7 per scegliere il tipo di stanza da prenotare: \n1)singola \n2)doppia \n3)tripla \n4)quadrupla \n5)suite \n6)doppia con 3° letto \n7)doppia con 4° letto");

        r = multiChoice("Seleziona una tipologia di stanza", options, 7);

        // Salvo il giorno coerentemente a come è salvato sul DB
        switch(r) {
            case '1':
                tipo=stanze[0];
                break;
            case '2':
                tipo=stanze[1];
                break;
            case '3':
                tipo=stanze[2];
                break;
            case '4':
                tipo=stanze[3];
                break;
            case '5':
                tipo=stanze[4];
                break;
            case '6':
                tipo=stanze[5];
                break;
```

```
            case '7':
                tipo=stanze[6];
                break;
            default:
                fprintf(stderr, "Invalid condition at %s:%d\n", __FILE__, __LINE__);
                abort();
        }



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

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

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

        param[1].buffer_type = MYSQL_TYPE_DATE;
        param[1].buffer = &data;
        param[1].buffer_length = sizeof(data);

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

        param[3].buffer_type = MYSQL_TYPE_VAR_STRING;
        param[3].buffer = tipo;
```

```
	param[3].buffer_length = strlen(tipo);



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

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

	printf("\nstanza prenotata correttamente\n");



 out:
	mysql_stmt_close(prepared_stmt);
}

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

	// Input for counter routine
	char telefono[16];
	MYSQL_TIME data;
	char itinerario[46];
	int n;
```

```
printf("\ninserire il numero di telefono dell'hostess da assegnare al viaggio: ");
getInput(16, telefono, false);

printf("\ninserire l'itinerario del viaggio che si vuole confermare: ");
getInput(46, itinerario, false);

char temp[10];

memset(&data, 0, sizeof(data));

printf("\ninserire ora la data del viaggio che si vuole confermare");

G:
printf("\ninserire il giorno del viaggio, da 1 a 31: ");
getInput(10, temp, false);
n = atoi(temp);
if(n < 1 || n > 31){
        printf("\ngiorno non valido, riprovare: ");
        goto G;
}

data.day = n;

M:
printf("\ndigitare il mese del viaggio, da 1 a 12: ");
getInput(10, temp, false);
n = atoi(temp);
if(n < 1 || n > 12){
        printf("\nmese non valido, riprovare: ");
        goto M;
}
```

```
data.month = n;

A:
printf("\ninserire l'anno del viaggio, da 2000 a 2022: ");
getInput(10, temp, false);
n = atoi(temp);
if(n < 2000 || n > 2022){
        printf("\nanno non valido, riprovare: ");
        goto A;
}

data.year = n;




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

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

param[0].buffer_type = MYSQL_TYPE_DATE;
param[0].buffer = &data;
param[0].buffer_length = sizeof(data);

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

param[2].buffer_type = MYSQL_TYPE_VAR_STRING;
```

```
	param[2].buffer = telefono;
	param[2].buffer_length = strlen(telefono);



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

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

	printf("\nviaggio confermato correttamente\n");



 out:
	mysql_stmt_close(prepared_stmt);
}

static void inserisci_costo(MYSQL *conn) {
	MYSQL_STMT *prepared_stmt;
	MYSQL_BIND param[5];
	char options[7] = {'1','2','3','4','5','6','7'};
	char r;

	// Input for inserisci_costo routine
	char indirizzo[46];
	MYSQL_TIME data;
	char itinerario[46];
```

```
char *stanze[7] = {"singola", "doppia", "tripla", "quadrupla", "suite", "doppia con 3° letto", "doppia con 4° letto"};
char *tipo;
int costo;
int n;



printf("\ninserire l'indirizzo dell'albergo: ");
getInput(46, indirizzo, false);

printf("\ninserire l'itinerario: ");
getInput(46, itinerario, false);



char temp[10];

memset(&data, 0, sizeof(data));

printf("\ninserire ora la data di partenza del viaggio");

G:
printf("\ninserire il giorno di partenza del viaggio, da 1 a 31: ");
getInput(10, temp, false);
n = atoi(temp);
if(n < 1 || n > 31){
        printf("\ngiorno non valido, riprovare: ");
        goto G;
}

data.day = n;
```

```
M:
printf("\ndigitare il mese di partenza del viaggio, da 1 a 12: ");
getInput(10, temp, false);
n = atoi(temp);
if(n < 1 || n > 12){
        printf("\nmese non valido, riprovare: ");
        goto M;
}

data.month = n;

A:
printf("\ninserire l'anno di partenza del viaggio, da 2000 a 2022: ");
getInput(10, temp, false);
n = atoi(temp);
if(n < 2000 || n > 2022){
        printf("\nanno non valido, riprovare: ");
        goto A;
}

data.year = n;



printf("\ninserire un numero da 1 a 7 per scegliere il tipo di stanza da prenotare: \n1)singola \n2)doppia \n3)tripla \n4)quadrupla \n5)suite \n6)doppia con 3° letto \n7)doppia con 4° letto");

r = multiChoice("Seleziona una tipologia di stanza", options, 7);

// Salvo il giorno coerentemente a come è salvato sul DB
switch(r) {
        case '1':
```

```
                tipo=stanze[0];
                break;
        case '2':
                tipo=stanze[1];
                break;
        case '3':
                tipo=stanze[2];
                break;
        case '4':
                tipo=stanze[3];
                break;
        case '5':
                tipo=stanze[4];
                break;
        case '6':
                tipo=stanze[5];
                break;
        case '7':
                tipo=stanze[6];
                break;
        default:
                fprintf(stderr, "Invalid condition at %s:%d\n", __FILE__, __LINE__);
                abort();
}

printf("\ninserire il costo della stanza: ");
getInput(5, temp, false);

costo = atoi(temp);

if(!setup_prepared_stmt(&prepared_stmt, "call inserisci_costo(?,?,?,?,?)", conn)) {
```

```
        finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize inserisci_costo statement\n", false);
    }

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

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

    param[1].buffer_type = MYSQL_TYPE_DATE;
    param[1].buffer = &data;
    param[1].buffer_length = sizeof(data);

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

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

    param[4].buffer_type = MYSQL_TYPE_LONG;
    param[4].buffer = &costo;
    param[4].buffer_length = sizeof(costo);

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

    // Run procedure
```

```
        if (mysql_stmt_execute(prepared_stmt) != 0) {
                print_stmt_error (prepared_stmt, "An error occurred while booking room.");
                goto out;
        }

        printf("\nstanza prenotata correttamente\n");


 out:
        mysql_stmt_close(prepared_stmt);
}




void run_as_hostess(MYSQL *conn)
{
        char options[7] = {'1','2','3','4','5','6', '7'};
        char op;

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

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

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

```
while(true) {
        printf("\033[2J\033[H");
        printf("*** What should I do for you? ***\n\n");
        printf("1) ricerca viaggi assegnati \n");
        printf("2) inserisci un cliente \n");
        printf("3) inserisci un passeggero \n");
        printf("4) conferma un viaggio\n");
        printf("5) prenota una stanza\n");
        printf("6) inserisci il costo di una stanza\n");
        printf("7) Quit\n");

        op = multiChoice("Select an option", options, 7);

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

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

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

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

                case '5':
```

```
                prenota_stanza(conn);
                break;

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

            case '7':
                return;

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

        getchar();
    }
}
```

MECCANICO.C:

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

#include "defines.h"


static void ordina_ricambio(MYSQL *conn) {
    MYSQL_STMT *prepared_stmt;
```

```
MYSQL_BIND param[1];

// Input for ordina_ricambio routine
char codice[6];




printf("\ncodice ricambio: ");
getInput(6, codice, false);

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

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

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

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

// Run procedure
if (mysql_stmt_execute(prepared_stmt) != 0) {
        print_stmt_error (prepared_stmt, "An error occurred while searching counter value.");
        goto out;
}
```

```
        printf("\n pezzo di ricambio ordinato correttamente");


  out:
        mysql_stmt_close(prepared_stmt);
}



static void utilizza_ricambio(MYSQL *conn) {
        MYSQL_STMT *prepared_stmt;

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

        if (mysql_stmt_execute(prepared_stmt) != 0) {
                print_stmt_error (prepared_stmt, "An error occurred while searchin available spare .");
                mysql_stmt_close(prepared_stmt);
                return;
        }

        // Dump the result set
        dump_result_set(conn, prepared_stmt, "\nDi seguito i pezzi di ricambio disponibili:");



        //per consumare il result set
        if (mysql_stmt_next_result(prepared_stmt) > 0){
                finish_with_stmt_error(conn, prepared_stmt, "Unexpected condition", true);
        }

        mysql_stmt_close(prepared_stmt);
```

```
char options[2] = {'1','2'};
char r;
int disp;

printf("\n il pezzo di tuo interesse è disponibile?: \n1) si \n2) no");

r = multiChoice("\n scegliere un opzione", options, 2);

switch(r) {
	case '1':
		disp = 1;
		break;
	case '2':
		disp = 0;
		break;
	default:
		fprintf(stderr, "Invalid condition at %s:%d\n", __FILE__, __LINE__);
		abort();
}

if(disp == 1){



	// Input for counter routine
	MYSQL_BIND param[3];
	char pullman[8];
	char codice[6];
	MYSQL_TIME inizio;
	int n;
	char temp[10];
```

```
printf("\ninserire targa pullman: ");
getInput(8, pullman, false);
printf("\n inserire codice del pezzo: ");
getInput(6, codice, false);

memset(&inizio, 0, sizeof(inizio));

G:
printf("\ninserire il giorno di inizio, da 1 a 31: ");
getInput(10, temp, false);
n = atoi(temp);
if(n < 1 || n > 31){
        printf("\ngiorno non valido, riprovare: ");
        goto G;
}

inizio.day = n;

M:
printf("\ninserire il mese di inizio, da 1 a 12: ");
getInput(10, temp, false);
n = atoi(temp);
if(n < 1 || n > 12){
        printf("\nmese non valido, riprovare: ");
        goto M;
}

inizio.month = n;

A:
```

```
		printf("\ninserire l'anno di inizio, da 2000 a 2022: ");
		getInput(10, temp, false);
		n = atoi(temp);
		if(n < 2000 || n > 2022){
			printf("\nanno non valido, riprovare: ");
			goto A;
		}

		inizio.year = n;




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

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

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

		param[1].buffer_type = MYSQL_TYPE_DATE;
		param[1].buffer = &inizio;
		param[1].buffer_length = sizeof(inizio);

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

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

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

		printf("\n pezzo di ricambio utilizzato correttamente");

		out:
		mysql_stmt_close(prepared_stmt);
	}
	else{
		printf("\n è possiile ordinare i pezzi dalla schermata iniziale");
	}
}


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

	// Input for counter routine
	char pullman[8];



	printf("\ntarga pullman: ");
```

```
	getInput(8, pullman, false);

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

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

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

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

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

	// Dump the result set
	dump_result_set(conn, prepared_stmt, "\nDi seguito il valore del contatore:");


	//per consumare il result set
	if (mysql_stmt_next_result(prepared_stmt) > 0){
		finish_with_stmt_error(conn, prepared_stmt, "Unexpected condition", true);
	}
```

```
 out:
        mysql_stmt_close(prepared_stmt);
}



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

        // Input for counter routine
        char pullman[8];
        char options[8] = {'1','2','3','4','5','6','7','8'};
        char *controlli[8] = {"motore", "convergenza", "freni", "pneumatici", "luci", "batteria", "sospensioni", "carrozzeria"};
 char *tipo;
 char r;

        printf("\ntarga pullman: ");
        getInput(8, pullman, false);

        printf("\ninserire un numero da 1 a 8 per scegliere il controllo ordinario di interesse: \n1)motore \n2)convergenza \n3)freni \n4)pneumatici \n5)luci \n6)batteria \n7)sospensioni \n8)carrozzeria ");

        r = multiChoice("\nSeleziona un controllo", options, 8);

        // Salvo il giorno coerentemente a come è salvato sul DB
        switch(r) {
                case '1':
                        tipo=controlli[0];
                        break;
                case '2':
                        tipo=controlli[1];
```

```
                break;
        case '3':
                tipo=controlli[2];
                break;
        case '4':
                tipo=controlli[3];
                break;
        case '5':
                tipo=controlli[4];
                break;
        case '6':
                tipo=controlli[5];
                break;
        case '7':
                tipo=controlli[6];
                break;
        case '8':
                tipo=controlli[7];
                break;
        default:
                fprintf(stderr, "Invalid condition at %s:%d\n", __FILE__, __LINE__);
                abort();
}

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

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

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

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

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

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

        // Dump the result set
        dump_result_set(conn, prepared_stmt, "\nDi seguito a quanti chilometri il pullamn dovrà ripetere il controllo:");



        //per consumare il result set
        if (mysql_stmt_next_result(prepared_stmt) > 0){
                finish_with_stmt_error(conn, prepared_stmt, "Unexpected condition", true);
        }

 out:
        mysql_stmt_close(prepared_stmt);
}
```

```
static void update_next_inspection(MYSQL *conn) {
	MYSQL_STMT *prepared_stmt;


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

	if (mysql_stmt_execute(prepared_stmt) != 0) {
		print_stmt_error (prepared_stmt, "An error occurred while searchin next revisions .");
		goto out;
	}

	// Dump the result set
	dump_result_set(conn, prepared_stmt, "\nDi seguito la lista dei controlli da effettuare:");


	//per consumare il result set
	if (mysql_stmt_next_result(prepared_stmt) > 0){
		finish_with_stmt_error(conn, prepared_stmt, "Unexpected condition", true);
	}

 out:
	mysql_stmt_close(prepared_stmt);


	MYSQL_BIND param[13];
 bool is_null = 1;

	// Input for next_inspection routine
```

```
char pullman[8];
char motivo[1000];
int n;
char options_b[2] = {'1','2'};
char r;
int nulls[8] = {1,1,1,1,1,1,1,1};
MYSQL_TIME inizio;
MYSQL_TIME fine;
char temp[10];
param[2].is_null = &is_null;


printf("\ntarga pullman: ");
getInput(8, pullman, false);


printf("\ndigitare 1 se un controllo ordinario oppure 2 se straordinario, poi premere invio: ");

r = multiChoice("digitare 1 o 2", options_b, 2);

switch(r) {
        case '1':
                n=1;
                break;
        case '2':
                n=2;
                break;
        default:
                fprintf(stderr, "Invalid condition at %s:%d\n", __FILE__, __LINE__);
                abort();
}
```

```
int motivo_bool = 0;

if(n == 1){

        motivo_bool = 1;

        char options_c[8] = {'1','2','3','4','5','6','7','8'};




        printf("\ndigitare il numero di controlli che si vogliono effettuare: ");
        getInput(1, temp, false);
        n = atoi(temp);




        printf("\nscegliere %d controlli digitando il numero e premendo invio, uno per volta: ",
n);

        int i;

        for(i = 0; i < n; i++) {
                printf("\ninserire un numero da 1 a 8 per scegliere il controllo ordinario di
interesse: \n1)motore \n2)convergenza \n3)freni \n4)pneumatici \n5)luci \n6)batteria \n7)sospensioni
\n8)carrozzeria ");


                r = multiChoice("Seleziona un controllo", options_c, 8);


                switch(r) {
```

```
                case '1':
                        if(nulls[0] == 0){
                                printf("\ncontrollo gia selezionato, si prega di selezionarne un altro");
                                i = i - 1;
                        }
                        else{
                                nulls[0] = 0;
                        }
                        break;
                case '2':
                        if(nulls[1] == 0){
                                printf("\ncontrollo gia selezionato, si prega di selezionarne un altro");
                                i = i - 1;
                        }
                        else{
                                nulls[1] = 0;
                        }
                        break;
                case '3':
                        if(nulls[2] == 0){
                                printf("\ncontrollo gia selezionato, si prega di selezionarne un altro");
                                i = i - 1;
                        }
                        else{
                                nulls[2] = 0;
                        }
                        break;
                case '4':
                        if(nulls[3] == 0){
```

```
                printf("\ncontrollo gia selezionato, si prega di selezionarne un altro");
                i = i - 1;
            }
            else{
                nulls[3] = 0;
            }
            break;
        case '5':
            if(nulls[4] == 0){
                printf("\ncontrollo gia selezionato, si prega di selezionarne un altro");
                i = i - 1;
            }
            else{
                nulls[4] = 0;
            }
            break;
        case '6':
            if(nulls[5] == 0){
                printf("\ncontrollo gia selezionato, si prega di selezionarne un altro");
                i = i - 1;
            }
            else{
                nulls[5] = 0;
            }
            break;
        case '7':
            if(nulls[6] == 0){
                printf("\ncontrollo gia selezionato, si prega di selezionarne un altro");
                i = i - 1;
```

```
                                        }
                                        else{
                                                nulls[6] = 0;
                                        }
                                        break;
                                case '8':
                                        if(nulls[7] == 0){
                                                printf("\ncontrollo gia selezionato, si prega di selezionarne un altro");
                                                i = i - 1;
                                        }
                                        else{
                                                nulls[7] = 0;
                                        }
                                        break;
                                default:
                                        fprintf(stderr, "Invalid condition at %s:%d\n", __FILE__, __LINE__);
                                        abort();
                                }
                        }
                }
                else{
                        printf("\ninserire il motivo ella revisione straordinaria: ");
                        getInput(1000, motivo, false);

                }


                memset(&inizio, 0, sizeof(inizio));

                G:
                printf("\ninserire il giorno di inizio, da 1 a 31: ");
```

```
getInput(10, temp, false);
n = atoi(temp);
if(n < 1 || n > 31){
        printf("\ngiorno non valido, riprovare: ");
        goto G;
}

inizio.day = n;

M:
printf("\ninserire il mese di inizio, da 1 a 12: ");
getInput(10, temp, false);
n = atoi(temp);
if(n < 1 || n > 12){
        printf("\nmese non valido, riprovare: ");
        goto M;
}

inizio.month = n;

A:
printf("\ninserire l'anno di inizio, da 2000 a 2022: ");
getInput(10, temp, false);
n = atoi(temp);
if(n < 2000 || n > 2022){
        printf("\nanno non valido, riprovare: ");
        goto A;
}

inizio.year = n;

int fine_bool;
```

```
printf("\ndigitare  1 se si vuole inserire la data di fine, 2 se la revisione è in corso");

r = multiChoice("selezionare una delle op", options_b, 2);

switch(r) {
	case '1':
		fine_bool = 0;
		break;
	case '2':
		fine_bool = 1;
		break;
	default:
		fprintf(stderr, "Invalid condition at %s:%d\n", __FILE__, __LINE__);
		abort();
}

if(fine_bool == 0){

	memset(&fine, 0, sizeof(fine));

	Gg:
	printf("\ninserire il giorno di fine, da 1 a 31: ");
	getInput(10, temp, false);
	n = atoi(temp);
	if(n < 1 || n > 31){
		printf("\ngiorno non valido, riprovare: ");
		goto Gg;
	}

	fine.day = n;
```

```
		Mm:
		printf("\ninserire il mese di fine, da 1 a 12: ");
		getInput(10, temp, false);
		n = atoi(temp);
		if(n < 1 || n > 12){
			printf("\nmese non valido, riprovare: ");
			goto Mm;
		}

		fine.month = n;

		Aa:
		printf("\ninserire l'anno di fine, da 2000 a 2022: ");
		getInput(10, temp, false);
		n = atoi(temp);
		if(n < 2000 || n > 2022){
			printf("\nanno non valido, riprovare: ");
			goto Aa;
		}

		fine.year = n;
	}

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

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

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

```
param[0].buffer = pullman;
param[0].buffer_length = strlen(pullman);

param[1].buffer_type = MYSQL_TYPE_DATE;
param[1].buffer = &inizio;
param[1].buffer_length = sizeof(inizio);


param[2].buffer_type = MYSQL_TYPE_DATE;
param[2].buffer = &fine;
param[2].buffer_length = sizeof(fine);
if(fine_bool == 1){
        param[2].is_null = &is_null;
}


param[3].buffer_type = MYSQL_TYPE_STRING;
param[3].buffer = motivo;
param[3].buffer_length = strlen(motivo);
if(motivo_bool == 1){
        param[3].is_null = &is_null;
}

param[4].buffer_type = MYSQL_TYPE_LONG;
param[4].buffer = &nulls[0];
param[4].buffer_length = sizeof(nulls[0]);

param[5].buffer_type = MYSQL_TYPE_LONG;
param[5].buffer = &nulls[1];
param[5].buffer_length = sizeof(nulls[1]);

param[6].buffer_type = MYSQL_TYPE_LONG;
```

```
    param[6].buffer = &nulls[2];
    param[6].buffer_length = sizeof(nulls[2]);

    param[7].buffer_type = MYSQL_TYPE_LONG;
    param[7].buffer = &nulls[3];
    param[7].buffer_length = sizeof(nulls[3]);

    param[8].buffer_type = MYSQL_TYPE_LONG;
    param[8].buffer = &nulls[4];
    param[8].buffer_length = sizeof(nulls[4]);

    param[9].buffer_type = MYSQL_TYPE_LONG;
    param[9].buffer = &nulls[5];
    param[9].buffer_length = sizeof(nulls[5]);

    param[10].buffer_type = MYSQL_TYPE_LONG;
    param[10].buffer = &nulls[6];
    param[10].buffer_length = sizeof(nulls[6]);

    param[11].buffer_type = MYSQL_TYPE_LONG;
    param[11].buffer = &nulls[7];
    param[11].buffer_length = sizeof(nulls[7]);

    param[12].buffer_type = MYSQL_TYPE_STRING;
    param[12].buffer = conf.username;
    param[12].buffer_length = strlen(conf.username);

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

```
	// Run procedure
	if (mysql_stmt_execute(prepared_stmt) != 0) {
		print_stmt_error (prepared_stmt, "An error occurred while inserting new inspection session.");
		goto out_2;
	}

	printf("revisione inserita correttamente\n");

	out_2:
	mysql_stmt_close(prepared_stmt);

}


static void revisioni(MYSQL *conn) {
	MYSQL_STMT *prepared_stmt;




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

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

	// Dump the result set
```

```
        dump_result_set(conn, prepared_stmt, "\nDi seguito le rervisoni da effettuare:");


        //per consumare il result set
        if (mysql_stmt_next_result(prepared_stmt) > 0){
                finish_with_stmt_error(conn, prepared_stmt, "Unexpected condition", true);
        }

 out:
        mysql_stmt_close(prepared_stmt);
}




void run_as_meccanico(MYSQL *conn)
{
        char options[7] = {'1','2','3','4','5','6','7'};
        char op;

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

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

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

```
while(true) {
        printf("\033[2J\033[H");
        printf("*** What should I do for you? ***\n\n");
        printf("1) ricerca contatore pullman \n");
        printf("2) ricerca prossima revisione di un certo tipo per un pullman \n");
        printf("3) effettua una revisione \n");
        printf("4) utilizza un pezzo di ricambio \n");
        printf("5) ordina ricambio \n");
        printf("6) visualizza le revisioni che bisogna effettuare \n");
        printf("7) Quit\n");

        op = multiChoice("Select an option", options, 7);

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

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

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

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

                case '5':
                        ordina_ricambio(conn);
```

```
                        break;

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

                case '7':
                        return;

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

        getchar();
    }
}
```

PASSEGGERO.C:

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

#include "defines.h"

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

    // Input for verifica_viaggio routine
```

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

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

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

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

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

// Dump the result set
dump_result_set(conn, prepared_stmt, "\nDi seguito lo stato del viaggio:");


//per consumare il result set
if (mysql_stmt_next_result(prepared_stmt) > 0){
        finish_with_stmt_error(conn, prepared_stmt, "Unexpected condition", true);
}
```

```
 out:
        mysql_stmt_close(prepared_stmt);
}

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

        // Input for verifica_viaggio routine




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

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

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

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

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

```
            print_stmt_error (prepared_stmt, "An error occurred while searching visits' informations.");
            goto out;
        }

        // Dump the result set
        dump_result_set(conn, prepared_stmt, "\nDi seguito lo stato del viaggio:");



        //per consumare il result set
        if (mysql_stmt_next_result(prepared_stmt) > 0){
            finish_with_stmt_error(conn, prepared_stmt, "Unexpected condition", true);
        }

 out:
        mysql_stmt_close(prepared_stmt);
}




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

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

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

```
}

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

while(true) {
        printf("\033[2J\033[H");
        printf("*** What should I do for you? ***\n\n");
        printf("1) verifica lo stato del viaggio \n");
        printf("2) visualizza informazioni sulle visite del viaggio \n");
        printf("3) Quit\n");

        op = multiChoice("Select an option", options, 3);

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

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

                case '3':
                        return;

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

```
                getchar();
        }
}
```

PARSE.C:

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

#include "defines.h"

#define BUFF_SIZE 4096

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

/**
 * JSON type identifier. Basic types are:
 *      o Object
 *      o Array
 *      o String
 *      o Other primitive: number, boolean (true/false) or null
 */
typedef enum {
        JSMN_UNDEFINED = 0,
        JSMN_OBJECT = 1,
        JSMN_ARRAY = 2,
        JSMN_STRING = 3,
        JSMN_PRIMITIVE = 4
```

```
} jsmntype_t;

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

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

/**
 * JSON parser. Contains an array of token blocks available. Also stores
 * the string being parsed now and current position in that string
 */
typedef struct {
```

```
	unsigned int pos; /* offset in the JSON string */
	unsigned int toknext; /* next token to allocate */
	int toksuper; /* superior token node, e.g parent object or array */
} jsmn_parser;

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

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

```
/**
 * Fills next available token with JSON primitive.
 */
static int jsmn_parse_primitive(jsmn_parser *parser, const char *js,
		size_t len, jsmntok_t *tokens, size_t num_tokens) {
	jsmntok_t *token;
	int start;

	start = parser->pos;

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

```
found:
	if (tokens == NULL) {
		parser->pos--;
		return 0;
	}
	token = jsmn_alloc_token(parser, tokens, num_tokens);
	if (token == NULL) {
		parser->pos = start;
		return JSMN_ERROR_NOMEM;
	}
	jsmn_fill_token(token, JSMN_PRIMITIVE, start, parser->pos);
#ifdef JSMN_PARENT_LINKS
	token->parent = parser->toksuper;
#endif
	parser->pos--;
	return 0;
}

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

	int start = parser->pos;

	parser->pos++;

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

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

		/* Backslash: Quoted symbol expected */
		if (c == '\\' && parser->pos + 1 < len) {
			int i;
			parser->pos++;
			switch (js[parser->pos]) {
				/* Allowed escaped symbols */
				case '\"': case '/' : case '\\' : case 'b' :
				case 'f' : case 'r' : case 'n'  : case 't' :
					break;
				/* Allows escaped symbol \uXXXX */
				case 'u':
					parser->pos++;
					for(i = 0; i < 4 && parser->pos < len && js[parser->pos] != '\0';
i++) {
```

```
                                        /* If it isn't a hex character we have an error */
                                        if(!((js[parser->pos] >= 48 && js[parser->pos] <= 57) ||
/* 0-9 */
                                                        (js[parser->pos]    >=    65    &&
js[parser->pos] <= 70) || /* A-F */
                                                        (js[parser->pos]    >=    97    &&
js[parser->pos] <= 102))) { /* a-f */
                                                parser->pos = start;
                                                return JSMN_ERROR_INVAL;
                                        }
                                        parser->pos++;
                                }
                                parser->pos--;
                                break;
                        /* Unexpected symbol */
                        default:
                                parser->pos = start;
                                return JSMN_ERROR_INVAL;
                        }
                }
        }
        parser->pos = start;
        return JSMN_ERROR_PART;
}

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

```
	int count = parser->toknext;

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

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

```
		return JSMN_ERROR_INVAL;
	}
	token = &tokens[parser->toknext - 1];
	for (;;) {
		if (token->start != -1 && token->end == -1) {
			if (token->type != type) {
				return JSMN_ERROR_INVAL;
			}
			token->end = parser->pos + 1;
			parser->toksuper = token->parent;
			break;
		}
		if (token->parent == -1) {
			if(token->type != type || parser->toksuper == -1) {
				return JSMN_ERROR_INVAL;
			}
			break;
		}
		token = &tokens[token->parent];
	}
#else
	for (i = parser->toknext - 1; i >= 0; i--) {
		token = &tokens[i];
		if (token->start != -1 && token->end == -1) {
			if (token->type != type) {
				return JSMN_ERROR_INVAL;
			}
			parser->toksuper = -1;
			token->end = parser->pos + 1;
			break;
		}
	}
```

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

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

```
                                tokens[parser->toksuper].size++;
                        break;

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

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

        return count;
}

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

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

static size_t load_file(char *filename)
{
	FILE *f = fopen(filename, "rb");
	if(f == NULL) {
		fprintf(stderr, "Unable to open file %s\n", filename);
		exit(1);
	}

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

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

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

	config[fsize] = 0;
```

```
	return fsize;
}

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

	load_file(path);

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

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

	/* Loop over all keys of the root object */
	for (i = 1; i < r; i++) {
		if (jsoneq(config, &t[i], "host") == 0) {
			/* We may use strndup() to fetch string value */
			conf->host = strndup(config + t[i+1].start, t[i+1].end-t[i+1].start);
			i++;
		} else if (jsoneq(config, &t[i], "username") == 0) {
```

```
                conf->db_username = strndup(config + t[i+1].start, t[i+1].end-t[i+1].start);
                i++;
            } else if (jsoneq(config, &t[i], "password") == 0) {
                conf->db_password = strndup(config + t[i+1].start, t[i+1].end-t[i+1].start);
                i++;
            } else if (jsoneq(config, &t[i], "port") == 0) {
                conf->port = strtol(config + t[i+1].start, NULL, 10);
                i++;
            } else if (jsoneq(config, &t[i], "database") == 0) {
                conf->database = strndup(config + t[i+1].start, t[i+1].end-t[i+1].start);
                i++;
            } else {
                printf("Unexpected key: %.*s\n", t[i].end-t[i].start, config + t[i].start);
            }
        }
        return 1;
}
```

UTILS.C:

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

#include "defines.h"

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

```
                        mysql_stmt_errno (stmt),
                        mysql_stmt_sqlstate(stmt),
                        mysql_stmt_error (stmt));
        }
}


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

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

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

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

        mysql_stmt_attr_set(*stmt, STMT_ATTR_UPDATE_MAX_LENGTH, &update_length);

        return true;
}

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

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

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

        mysql_field_seek(res_set, 0);
        putchar('+');
```

```
        for (i = 0; i < mysql_num_fields(res_set); i++) {
                field = mysql_fetch_field(res_set);

                if (field->type == MYSQL_TYPE_FLOAT) {
                        for (j = 0; j < 13; j++) putchar('-');
                        putchar('+');
                }
                else {
                        for (j = 0; j < field->max_length + 2; j++) putchar('-');
                        putchar('+');
                }
        }
        putchar('\n');
}

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

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

        mysql_field_seek (res_set, 0);

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

                if (col_len < field->max_length)
                        col_len = field->max_length;
```

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

        print_dashes(res_set);
        putchar('|');
        mysql_field_seek (res_set, 0);
        for (i = 0; i < mysql_num_fields(res_set); i++) {
            field = mysql_fetch_field(res_set);
            if (field->type == MYSQL_TYPE_FLOAT){
                printf(" %-*s |", 11, field->name);
            }
            else{
                printf(" %-*s |", (int)field->max_length, field->name);
            }
        }
        putchar('\n');

        print_dashes(res_set);
}

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

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

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

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

        if(mysql_stmt_num_rows(stmt) == 0) {
                printf("\nLa tabella è vuota\n");
                return;
        }

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

        dump_result_set_header(rs_metadata);
```

```
fields = mysql_fetch_fields(rs_metadata);

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

bool is_null[num_fields];

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

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

```
                        attr_size = sizeof(short int);
                        break;
                case MYSQL_TYPE_LONG:
                case MYSQL_TYPE_INT24:
                        attr_size = sizeof(int);
                        break;
                case MYSQL_TYPE_LONGLONG:
                        attr_size = sizeof(int);
                        break;
                default:
                        attr_size = fields[i].max_length;
                        break;
        }

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

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

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

/* fetch and display result set rows */
while (true) {
```

```
status = mysql_stmt_fetch(stmt);

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

putchar('|');

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

        if (is_null[i] == true) {
                printf (" %-*s |", (int)fields[i].max_length, "NULL");
                continue;
        }

        switch (rs_bind[i].buffer_type) {

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

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

                case MYSQL_TYPE_TIME:
                        date = (MYSQL_TIME *)rs_bind[i].buffer;
                        printf(" %02d:%-*.02d |", date->hour,
(int)fields[i].max_length-3, date->minute);
```

```
                    break;

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

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

                case MYSQL_TYPE_LONG:
                case MYSQL_TYPE_SHORT:
                case MYSQL_TYPE_TINY:
                case MYSQL_TYPE_LONGLONG:
                case MYSQL_TYPE_INT24:
                    printf(" %-*d |", (int)fields[i].max_length, *(int *)rs_bind[i].buffer);
                    break;

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

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

```
                print_dashes(rs_metadata);
        }

        mysql_free_result(rs_metadata); /* free metadata */

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

INOUT.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"

// Per la gestione dei segnali
static volatile sig_atomic_t signo;
```

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

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

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

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

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

		// Disattiva l'output su schermo
```

```
        if (tcgetattr(fileno(stdin), &oterm) == 0) {
                (void) memcpy(&term, &oterm, sizeof(struct termios));
                term.c_lflag &= ~(ECHO|ECHONL);
                (void) tcsetattr(fileno(stdin), TCSAFLUSH, &term);
        } else {
                (void) memset(&term, 0, sizeof(struct termios));
                (void) memset(&oterm, 0, sizeof(struct termios));
        }
}

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

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

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

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

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

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

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

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

	return stringa;
}
```

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


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

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

	// Decide quale delle due lettere mostrare come predefinite
	char s, n;
	if(predef) {
		s = toupper(yes);
		n = no;
	} else {
		s = yes;
		n = toupper(no);
	}

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

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

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

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

        // Genera la stringa delle possibilità
        char *possib = malloc(2 * num * sizeof(char));
        int i, j = 0;
        for(i = 0; i < num; i++) {
                possib[j++] = choices[i];
                possib[j++] = '/';
        }
        possib[j-1] = '\0'; // Per eliminare l'ultima '/'

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

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

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

DEFINES.H:

```
#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 dump_result_set(MYSQL *conn, MYSQL_STMT *stmt, char *title);

extern void run_as_passeggero(MYSQL *conn);

extern void run_as_autista(MYSQL *conn);

extern void run_as_meccanico(MYSQL *conn);

extern void run_as_hostess(MYSQL *conn);

extern void run_as_amministratore(MYSQL *conn);
```