Basi di Dati

Progetto A.A. 2024/2025

Agenzia di viaggio

0321928

David Julian Belfiori

Indice

1. Descrizione del Minimondo

Un'agenzia di viaggio organizza viaggi organizzati della durata da un giorno ad una settimana in località italiane ed europee. I viaggi utilizzano autobus privati. La compagnia intende realizzare un sistema informativo per la gestione dei viaggi e delle prenotazioni.

La segreteria dell'agenzia crea nuovi itinerari di viaggio. Un itinerario è caratterizzato da un insieme di pernottamenti in località potenzialmente differenti e di durata differente e da un costo. Un viaggio segue un certo itinerario, con una specifica data di partenza ed una data di rientro.

Per ciascun pernottamento associato ad un viaggio è di interesse tenere traccia di quale sia l'albergo scelto per ospitare i partecipanti. La scelta avviene all'interno di un insieme di alberghi mantenuto nel sistema informativo. Ciascun albergo è associato ad una capienza massima di persone, una città, un indirizzo, un insieme di recapiti (e-mail, telefono, fax) ed un referente.
Quando un piano di viaggio viene creato, questo è definito a partire dalle tappe che verranno coperte. Gli utenti del sistema possono prenotarsi al viaggio specificando il numero di persone per cui effettuano la prenotazione. Al termine della prenotazione viene fornito un codice che può essere utilizzato per disdire la prenotazione, fino a 20 giorni prima della partenza.

Quando le prenotazioni sono concluse (ovverosia, 20 giorni prima della partenza), la segreteria è a conoscenza di quante persone parteciperanno al viaggio. Assegnano quindi un insieme di autobus per consentire il trasporto di tutte le persone. Vari autobus possono infatti avere una capienza differente. Ciascun autobus ha un costo giornaliero forfettario, comprensivo del costo dell'autista e del carburante.

Una volta definito il numero di autobus necessari, la segreteria, per ciascuna città toccata dal viaggio, associa un albergo che abbia una disponibilità massima di posti superiore alle persone che effettuano il viaggio. Ciascun hotel ha un costo per notte per persona che è conservato nel sistema.
Per tutti i viaggi terminati, la segreteria può generare un report che mostri le informazioni sui partecipanti e il guadagno (o perdita) derivante da tale viaggio.

2. Analisi dei Requisiti

Identificazione dei termini ambigui e correzioni possibili

Linea	Termine	Nuovo termine	Motivo correzione
5	Insieme	Sequenza	Un itinerario segue una sequenza temporale di pernottamenti non un insieme
12	Tappe che verranno coperte	Itinerario	Il termine tappe e vario e generico , la sostituzione con itinerario rende la frase più coerente con la terminologia definita dal sistema.
17	Persone	Partecipanti	"Partecipanti" è più preciso e legato al contesto del viaggio.
23	Hotel	Albergo	Per seguire la terminologia
	Codice	Codice di prenotazione	
12-13	Utenti del sistema	Agenti di viaggio	Il termine utente del sistema è ambiguo, e non chiarisce se è un cliente finale ad effettuare la prenotazione o un agente di viaggio

Specifica disambiguata

1 Un'agenzia di viaggio organizza viaggi organizzati della durata da un giorno ad una settimana
2 in località italiane ed europee. I viaggi utilizzano autobus privati. La compagnia intende
3 realizzare un sistema informativo per la gestione dei viaggi e delle prenotazioni.

4 La segreteria dell'agenzia crea nuovi itinerari di viaggio. Un itinerario è caratterizzato da una
5 sequenza di pernottamenti in località potenzialmente differenti e di durata differente e da un
6 costo. Un viaggio segue un certo itinerario, con una specifica data di partenza ed una data di
7 rientro.

8 Per ciascun pernottamento associato ad un viaggio è di interesse tenere traccia di quale sia
9 l'albergo scelto per ospitare i partecipanti. La scelta avviene all'interno di un insieme di
10 alberghi mantenuto nel sistema informativo. Ciascun albergo è associato ad una capienza
11 massima di persone, una città, un indirizzo, un insieme di recapiti (e-mail, telefono, fax) ed un
referente.

12 Quando un piano di viaggio viene creato, questo è definito a partire dall'itinerario. Gli agenti

di viaggio possono prenotare un viaggio specificando il numero di persone per cui effettuano la prenotazione. Al termine della prenotazione viene fornito un codice di prenotazione che può essere utilizzato per disdire la prenotazione, fino a 20 giorni prima della partenza.

Quando le prenotazioni sono concluse (ovverosia, 20 giorni prima della partenza), la segreteria è a conoscenza di quante sono i partecipanti al viaggio. Assegnano quindi un insieme di autobus per consentire il trasporto di tutte le persone. Vari autobus possono infatti avere una capienza differente. Ciascun autobus ha un costo giornaliero forfettario, comprensivo del costo dell'autista e del carburante.

Una volta definito il numero di autobus necessari, la segreteria, per ciascuna città toccata dal viaggio, associa un albergo che abbia una disponibilità massima di posti superiore alle persone che effettuano il viaggio. Ciascun albergo ha un costo per notte per persona che è conservato nel sistema. Per tutti i viaggi terminati, la segreteria può generare un report che mostri le informazioni sui partecipanti e il guadagno (o perdita) derivante da tale viaggio.

Glossario dei Termini

Termine	Descrizione	Sinonimi	Collegamenti
Itinerario	Piano che descrive le tappe, i pernottamenti e il costo complessivo	Piano di viaggio, percorso	Viaggio
Viaggio	Segue un itinerario , caratterizzato da date di partenza e di rientro specifiche.	Tour	Itinerario, autobus, albergo
Prenotazione	Atto di riservare posti per un viaggio, specificando il numero di partecipanti. Una prenotazione è identificata da un codice e può essere annullata fino a 20 giorni prima della partenza attraverso l'uso del codice di disdetta.	Acquisto	Viaggio
Autobus	Mezzo di trasporto utilizzato per spostare i partecipanti da una tappa all'altra. Gli autobus hanno capienze differenti e costi giornalieri forfettari.	Pullman, veicolo	Viaggio

Albergo	Struttura ricettiva utilizzata per ospitare i partecipanti durante i pernottamenti. Ogni albergo è caratterizzato da capienza massima, città, indirizzo, recapiti e costi per persona a notte.	Hotel, struttura ricettiva	Pernottamento
Pernottamento	Sosta per trascorrere la notte. Caratterizzato da una località , dalla durata del pernottamento e dall'albergo associato	Soggiorno	Albergo
Report	Alla fine di un viaggio la segreteria può generare un report sulle persone che hanno partecipato e un bilancio economico finale del viaggio	Riepilogo	Viaggio

Raggruppamento dei requisiti in insiemi omogenei

Frasi relative a itinerario
La segreteria dell'agenzia crea nuovi itinerari di viaggio. Un itinerario è caratterizzato da una sequenza di pernottamenti in località potenzialmente differenti e di durata differente e da un costo.

Frasi relative a viaggio
Un viaggio segue un certo itinerario, con una specifica data di partenza ed una data di rientro. Quando un piano di viaggio viene creato, questo è definito a partire dall'itinerario. Gli utenti del sistema possono prenotarsi al viaggio specificando il numero di persone per cui effettuano la prenotazione. Una volta definito il numero di autobus necessari, la segreteria, per ciascuna città toccata dal viaggio, associa un albergo che abbia una disponibilità massima di posti superiore alle persone che effettuano il viaggio. Per tutti i viaggi terminati, la segreteria può generare un report che mostri le informazioni sui partecipanti e il guadagno (o perdita) derivante da tale viaggio.

Frasi relative a prenotazione
Quando un piano di viaggio viene creato, questo è definito a partire dall'itinerario. Gli agenti di viaggio possono prenotare un viaggio specificando il numero di persone per cui effettuano la prenotazione. Al termine della prenotazione viene fornito un codice di prenotazione che può essere utilizzato per disdire la prenotazione, fino a 20 giorni prima della partenza.

Quando le prenotazioni sono concluse (ovverosia, 20 giorni prima della partenza), la segreteria è a conoscenza di quante sono i partecipanti al viaggio. Assegnano quindi un insieme di autobus per consentire il trasporto di tutte le persone

Frasi relative a autobus
Assegnano quindi un insieme di autobus per consentire il trasporto di tutte le persone. Vari autobus possono infatti avere una capienza differente. Ciascun autobus ha un costo giornaliero forfettario, comprensivo del costo dell'autista e del carburante.

Frasi relative a albergo
Per ciascun pernottamento associato ad un viaggio è di interesse tenere traccia di quale sia l'albergo scelto per ospitare i partecipanti. La scelta avviene all'interno di un insieme di alberghi mantenuto nel sistema informativo. Ciascun albergo è associato ad una capienza massima di persone, una città, un indirizzo, un insieme di recapiti (e-mail, telefono, fax) ed un referente. Per ciascuna città toccata dal viaggio, associa un albergo che abbia una disponibilità massima di posti superiore alle persone che effettuano il viaggio. Ciascun albergo ha un costo per notte per persona che è conservato nel sistema.

Frasi relative a pernottamento
[…] caratterizzato da una sequenza di pernottamenti in località potenzialmente differenti e di durata differente e da un costo Per ciascun pernottamento associato ad un viaggio è di interesse tenere traccia di quale sia l'albergo scelto per ospitare i partecipanti.

Frasi relative a report
Per tutti i viaggi terminati, la segreteria può generare un report che mostri le informazioni sui partecipanti e il guadagno (o perdita) derivante da tale viaggio.

3. Progettazione concettuale

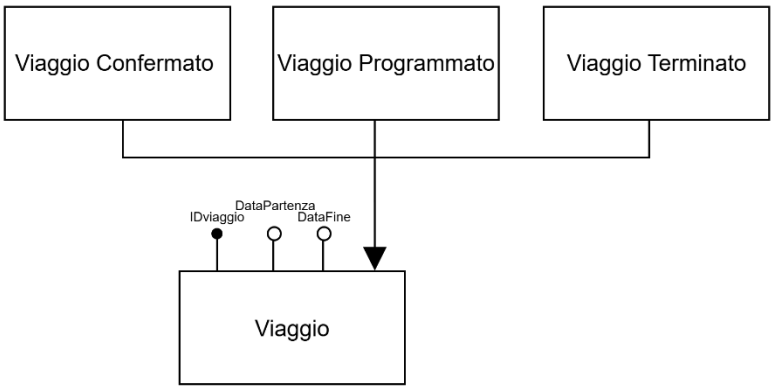
Costruzione dello schema E-R

In questa sezione è necessario riportare tutti passi seguiti per la costruzione dello schema E-R finale, a partire dalle specifiche raccolte ed organizzate nel capitolo precedente. Non è richiesto un procedimento specifico: si può adottare una strategia top-down, bottom-up, a macchia d'olio o mista. L'importante è descrivere e commentare tutti i passi della costruzione, andando anche ad inserire "schemi parziali" utilizzati nel processo.

La progettazione del modello E-R è stata realizzata seguendo la strategia inside-out. Questo approccio consiste nell'identificare inizialmente un nucleo ristretto di concetti chiave, che vengono poi ampliati in modo graduale ed espansivo per sviluppare progressivamente il modello completo.

Analizzando le specifiche dell'agenzia di viaggi, ho identificato **VIAGGIO** come entità centrale del dominio, poiché rappresenta il core business dell'agenzia e collega tutte le altre entità.

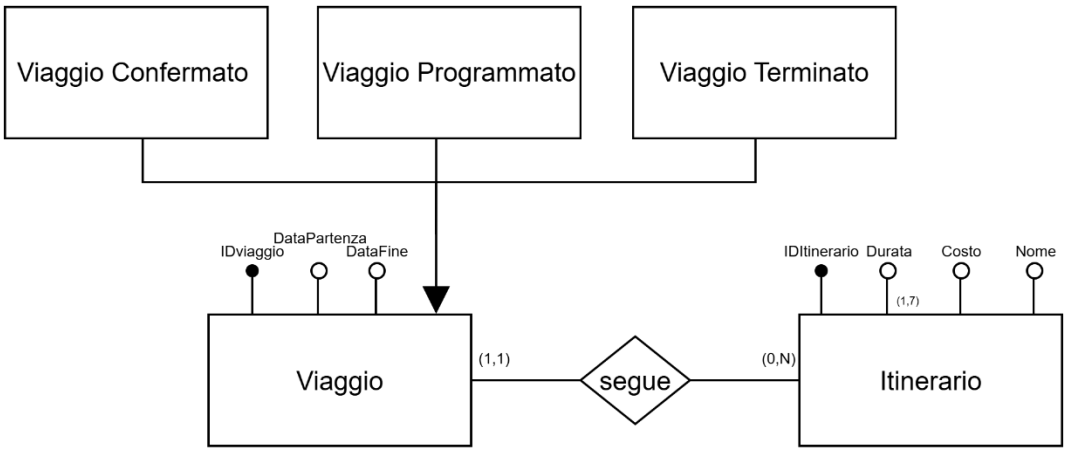
Questa entità rappresenta un viaggio organizzato specifico, con date di partenza e rientro definite. Vengono modellati anche gli stati di un viaggio, il suo ciclo di vita. Si parte da un viaggio *programmato* (quando mancano più di venti giorni alla data di partenza), allo scattare del 20 giorno, il viaggio passa allo stato di *incorso,* dove non sarà più possibile né effettuare delle prenotazioni né disdirne. Dopo la data di rientro il viaggio passa allo stato di *viaggio terminato*, a quel punto la segreteria potrà generare un report e vedere l'andamento economico del viaggio.



Il concetto di viaggio è fortemente legato ad un itinerario "Quando un piano di viaggio viene creato, questo è definito a partire dall'itinerario". L'entità **ITINERARIO** è fondamentale poiché rappresenta il "template" riutilizzabile per i viaggi. Separare l'itinerario dal viaggio specifico permette all'agenzia di riutilizzare percorsi consolidati per più viaggi in date diverse, ottimizzando il processo di pianificazione.

Nel contesto del modello concettuale, la relazione "SEGUE" lega le entità VIAGGIO e ITINERARIO con cardinalità (1,1) → (0,N) perché:

- Ogni viaggio deve necessariamente seguire uno e un solo itinerario predefinito (1,1)
- Un itinerario può essere utilizzato per zero viaggi (se appena creato o non più utilizzato) o per molti viaggi (0,N)



Ragionando sulla natura del minimondo di riferimento, si nota che un itinerario è composto da una sequenza di pernottamenti in località (italiane o europee) di durata differente e da un costo.

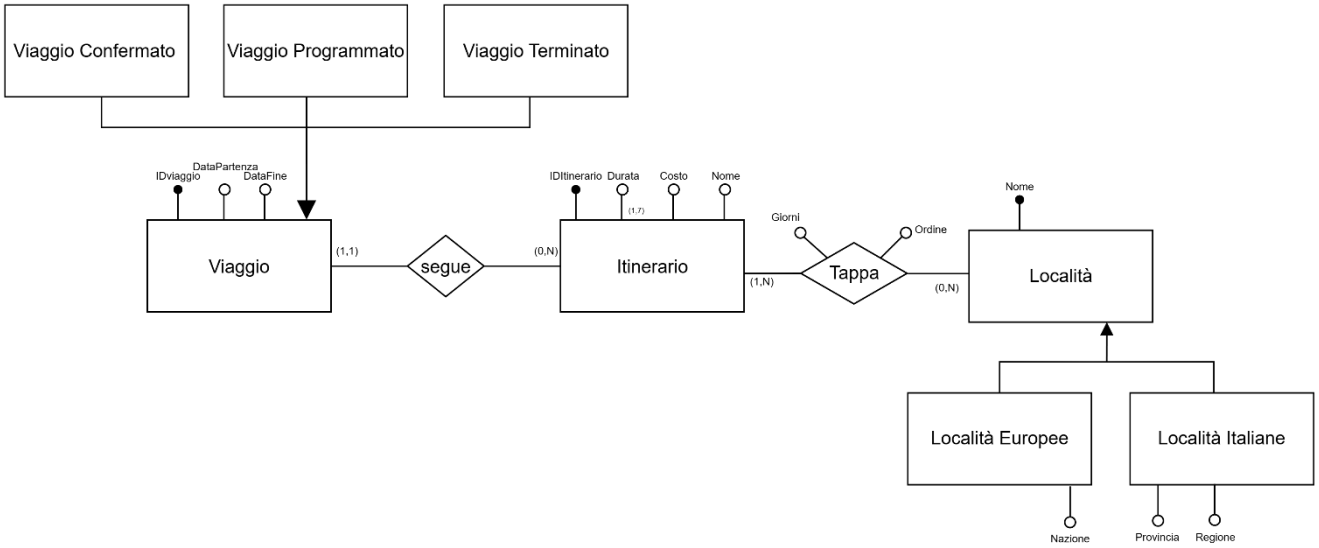
È importante sottolineare come un itinerario sia una sequenza di tappe in località diverse , dove ogni tappa quando il viaggio sarà confermato gli sarà associato un albergo e li diventerà un pernottamento. Un itinerario sarà solamente un susseguirsi di tappe con lunghezza differente.

Data questa relazione introduciamo l'entità **LOCALITA** e la specializziamo come dice la specifica in LOCALITA' EUROPPEE e NAZIONALI. Questa specializzazione è stata introdotta perché:

- Le località italiane richiedono la specificazione di provincia/regione
- Le località europee richiedono la specificazione dello stato
- Questa distinzione sarà magari rilevante nel futuro per aspetti logistici e normativi differenti

Relazione con Itinerario (TAPPA).La relazione TAPPA tra ITINERARIO e LOCALITÀ ha cardinalità (1,N) → (0,N) perché:

- Un itinerario deve comprendere almeno una località (altrimenti non avrebbe senso) e tipicamente ne include diverse (1,N)
- Una località può non essere inclusa in alcun itinerario o può essere parte di molteplici itinerari (0,N)



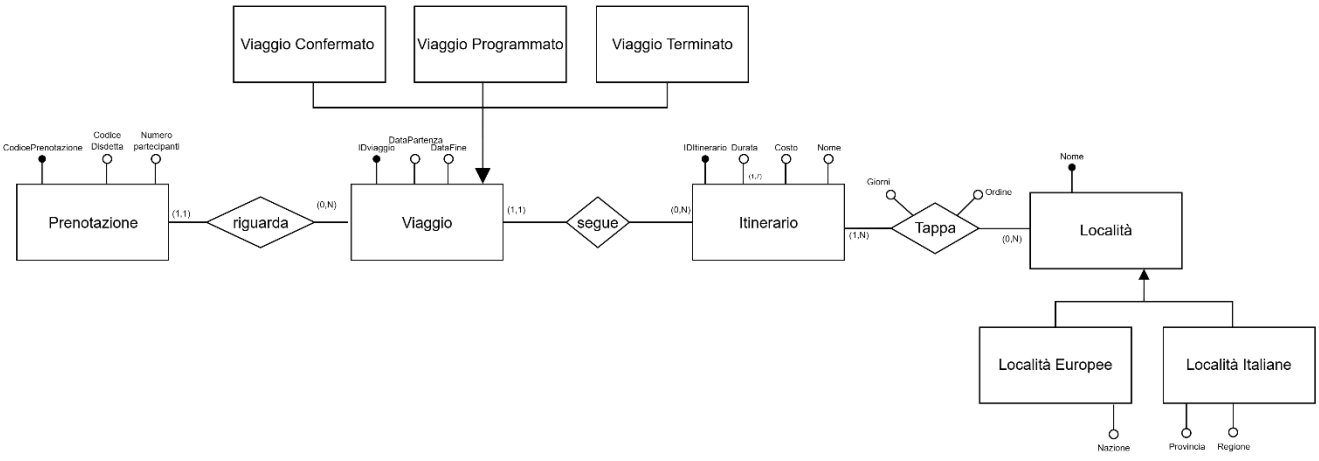
Introduciamo l'entità PRENOTAZIONE, rappresenta l'interazione fondamentale tra cliente e agenzia. È un'entità indispensabile per registrare le richieste dei clienti, gestire i posti disponibili e calcolare i ricavi per viaggio.

Una prenotazione sarà identificata da un codice prenotazione e avrà come altre informazioni il numero dei partecipanti e il codice di disdetta che l'agente di viaggio fornirà al cliente nel caso vorrà cancellare la sua prenotazione.

Relazione con VIAGGIO (PER): La relazione PER lega PRENOTAZIONE a VIAGGIO con cardinalità (1,1) → (0,N) perché:

- Ogni prenotazione è fatta per uno e un solo viaggio specifico (1,1)
- Un viaggio può esistere senza prenotazioni (quando appena creato) o avere molte prenotazioni (0,N)

La cardinalità minima (0) sul lato VIAGGIO permette di gestire i viaggi appena creati che non hanno ancora prenotazioni.



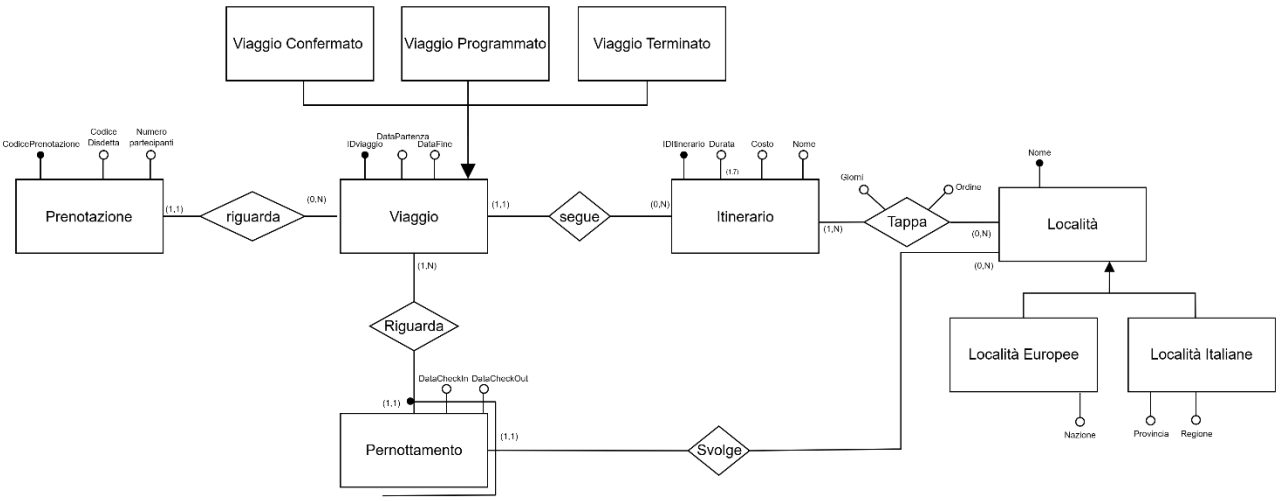
Dalla descrizione del minimondo notiamo che nel passaggio da Viaggio *programmato* a Viaggio *confermato* la segreteria assocerà per ogni località toccata dal viaggio una struttura ricettiva adeguata al numero di partecipanti (che conosciamo con precisione non essendo più possibile né prenotarsi né disdire), da qui nasce l'entità **Pernottamento** che collega tre concetti chiave: il viaggio, l'albergo dove si pernotta, e la località dove si trova l'albergo. Rappresenta l'effettiva implementazione dell'itinerario teorico in termini di strutture ricettive specifiche.

Relazione con VIAGGIO (RIGUARDA)**:** La relazione RIGUARDA lega VIAGGIO a PERNOTTAMENTO con cardinalità (1,N) → (1,1) perché:

- Un viaggio include sempre almeno un pernottamento (anche per viaggi di un giorno c'è tipicamente una struttura di appoggio) e generalmente più di uno (1,N)
- Ogni pernottamento fa parte di uno e un solo viaggio (1,1)

Relazione con LOCALITÀ (SVOLGE): La relazione SVOLGE lega PERNOTTAMENTO a LOCALITÀ con cardinalità (1,1) → (0,N) perché:

- Un pernottamento si svolge in una sola località specifica (1,1)
- Una località può ospitare uno o molti pernottamenti di viaggi diversi (0,N)



Un pernottamento andrà dunque fatto in una struttura recettiva (**ALBERGO**). La gestione degli alberghi è fondamentale per garantire un'adeguata sistemazione e per calcolare i costi di pernottamento.

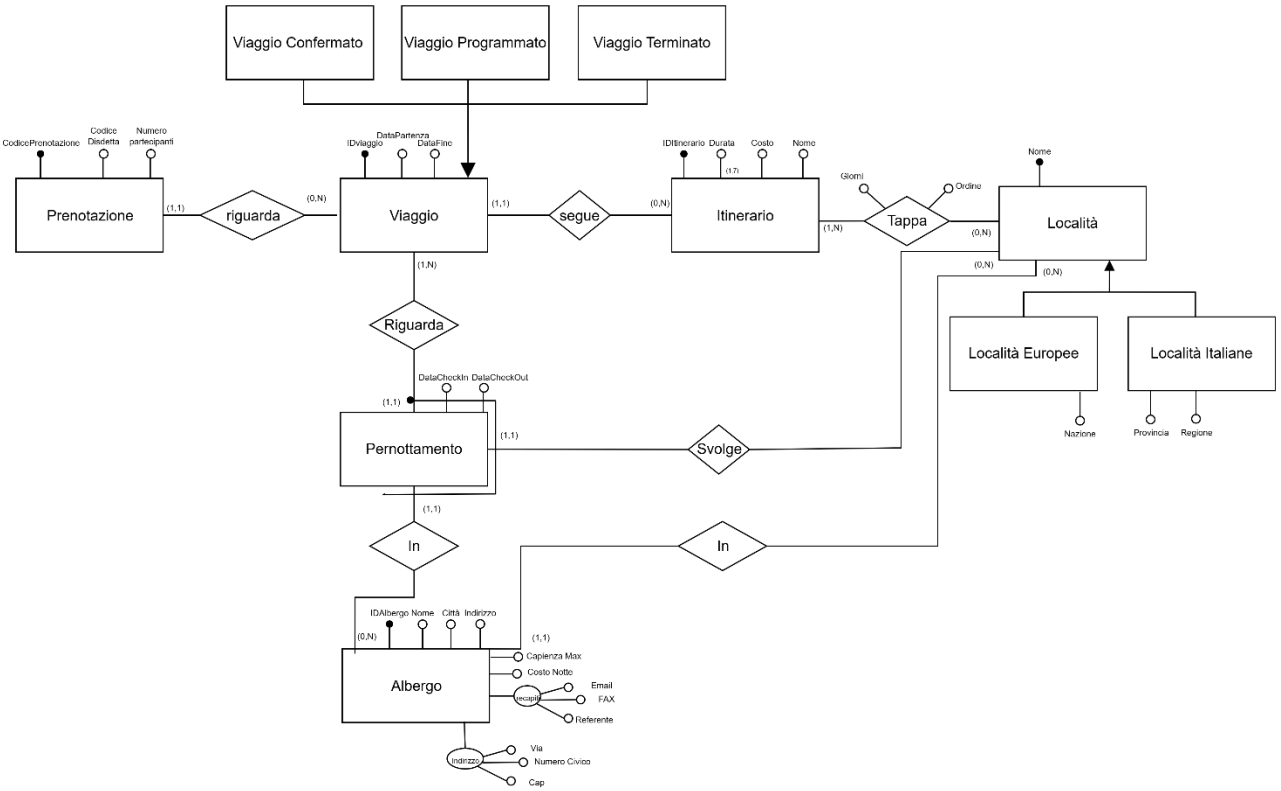
Gli attributi dettagliati di ALBERGO (nome, indirizzo, città, capienza, costo_notte, recapiti, referente) sono essenziali per la gestione operativa e amministrativa delle relazioni con le strutture ricettive.

Relazione con PERNOTTAMENTO (IN): La relazione IN lega PERNOTTAMENTO ad ALBERGO con cardinalità (1,1) → (0,N) perché:

- Un pernottamento avviene in uno e un solo albergo specifico (1,1)
- Un albergo può non essere mai utilizzato (se appena inserito nel database) o essere utilizzato per molti pernottamenti in viaggi diversi (0,N)

Relazione con LOCALITA' (RISIEDE): La relazione IN lega ALBERGO con LOCALITA' con cardinalità (1,1) → (0,N) perché:

- Un albergo risiede in una e una sola località (1,1)
- Una località potrà avere uno o più alberghi (1,N)

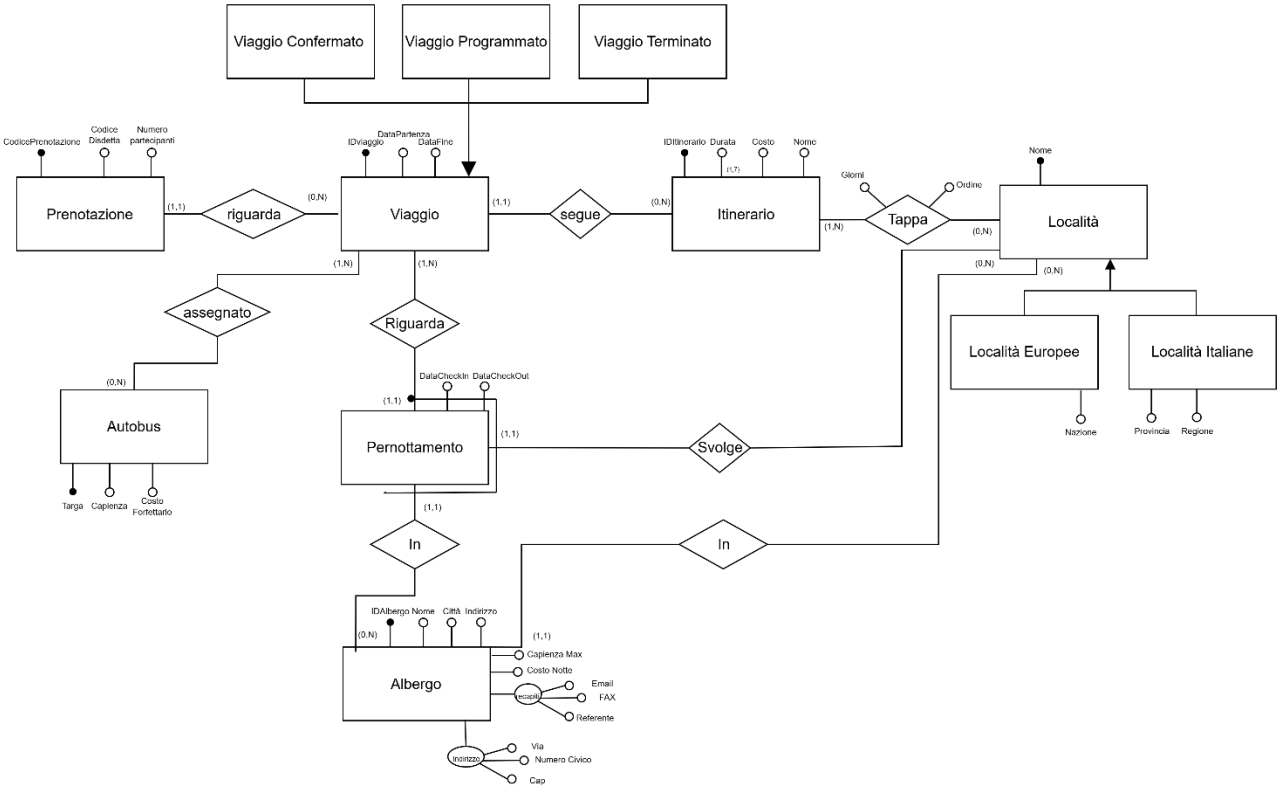


Come ci dice il la specifica "quando le prenotazioni sono concluse(ovverosia, 20 giorni prima della partenza), […] Assegnamo quindi un insieme di AUTOBUS …"

Introduciamo l'entità AUTOBUS, rappresenta i mezzi di trasporto che sono una risorsa critica per l'agenzia. La loro gestione è fondamentale per garantire capacità sufficiente per i partecipanti e per il calcolo dei costi operativi.

Relazione con VIAGGIO (ASSEGNATO)**:** La relazione ASSEGNATO lega AUTOBUS a VIAGGIO con cardinalità (0,N) → (1,N) perché:

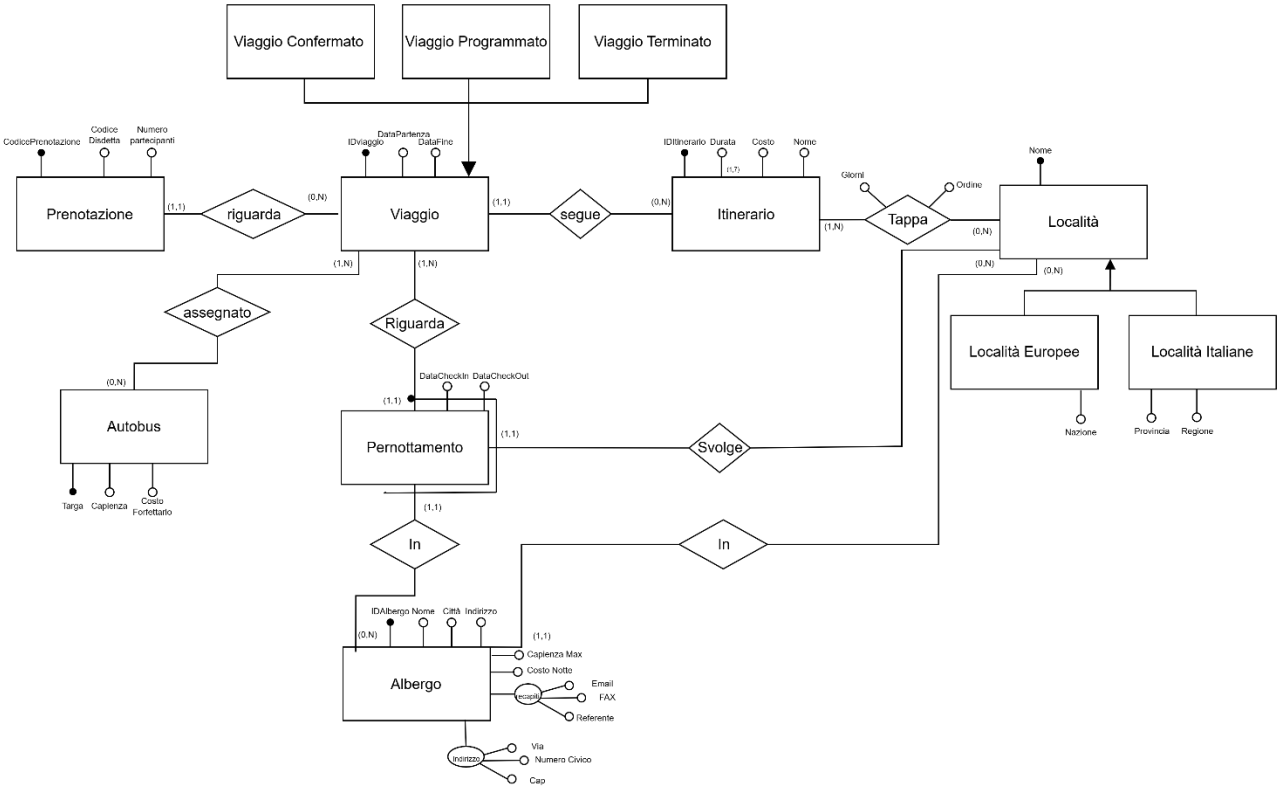
- Un autobus può non essere assegnato ad alcun viaggio (se temporaneamente non utilizzato) o essere assegnato a più viaggi in date diverse (0,N)
- Un viaggio necessita sempre di almeno un autobus, ma potrebbe richiederne più di uno in base al numero di partecipanti (1,N)



Integrazione finale

La strategia inside-out ha portato a creare un modello in cui ogni entità è collegata all'entità centrale VIAGGIO o a entità che lo supportano direttamente. Ciascuna entità è stata scelta per soddisfare requisiti specifici del dominio e le relazioni sono state definite per modellare accuratamente i legami logici e operativi tra esse.

Il sistema di relazioni permette di rappresentare fedelmente i processi dell'agenzia di viaggi, dalla creazione degli itinerari alla prenotazione dei clienti, fino alla gestione operativa di alberghi e autobus.



Regole aziendali

- Una prenotazione può essere effettuata solo se la data corrente è anteriore a (DataPartenza - 20 giorni) quindi se il viaggio è ancora nella fase *viaggio programmato*.
- La somma delle capienze degli autobus assegnati ad un viaggio deve essere maggiore o uguale al numero di partecipanti prenotati per quel viaggio.
- Verifica che gli autobus assegnati siano disponibili per le date del viaggio
- Un viaggio può avere una durata complessiva inferiore o uguale a 7 giorni.
- L'albergo assegnato ad una tappa (ossia, il pernottamento operativo in un viaggio) deve avere una capienza massima superiore al numero di partecipanti prenotati per quel viaggio.
- Un report può essere generato solo per viaggi che sono terminati (ovvero, dove la DataRientro è antecedente alla data corrente), quindi se il viaggio è nella fase *viaggio terminato*.

Dizionario dei dati

Entità	Descrizione	Attributi	Identificatori
Viaggio	Rappresenta l'effettiva esecuzione di un	DataPartenza	IDViaggio

	itinerario in date specifiche (con partenza e rientro).	DataFine	
Itinerario	Definisce la durata e il costo di una sequenza di tappe in diverse località	Durata, Costo	IDItinerario
Prenotazione	Registra la prenotazione di uno o più posti per un viaggio, con un codice univoco per eventuale disdetta entro i 20 giorni precedenti la partenza.	NumeroPartecipanti CodiceDisdetta	CodicePrenotazione
Autobus	Informazioni sui mezzi di trasporto	Capienza Costo Forfettario	Targa
Pernottamento	Associa una tappa (previsto dall'itinerario) a un albergo per un determinato viaggio, registrando informazioni operative (es. check-in/out, costi).	DataCheckIn, DataCheckOut	IDViaggio, NomeLocalità, CodiceAlbergo, DataCheckIn, DataCheckOut
Località	Rappresenta tutte le località europee e nazionali che sono nel sistema e che posso essere utilizzate per creare nuovi itinerari di viaggio	Provincia, Regione, Stato	Nome
Albergo	Informazioni sulla struttura alberghiera facente parte del pool di strutture ricettive per la località	Capienza Max Costo Notte Email Telefono Fax	Nome, Città, Indirizzo

4. Progettazione logica

Volume dei dati

Considero che il sistema non mantiene informazioni su prenotazioni e viaggi per più di 5 anni. L'agenzia è di piccola/media dimensione (5-8 viaggi organizzati al mese) e I viaggi hanno durata variabile da 1 a 7 giorni infine le località coperte sono sia italiane che europee

Concetto nello schema	Tipo[1]	Volume atteso
Località	E	200
Albergo	E	500
Viaggio	E	450
Prenotazione	E	9000
Itinerario	E	100
Pernottamento	E	2000
Autobus	E	60
Segue	R	450
Tappa	R	500
Per	R	9000
Riguarda	R	2000
In (pernottamento-albergo)	R	2000
Svolge	R	2000
Assegnato	R	900
In (albergo-località)	R	500

Veloce spiegazione della scelta dei volumi :

Località (200):~100 località italiane + ~100 località europee più frequentate.

Stima basata sulle principali località turistiche italiane ed europee che un'agenzia di piccole dimensioni potrebbe servire nel corso di 5 anni.

Albergo (500): ~2-3 alberghi convenzionati per ogni località frequentata

Considero che l'agenzia mantiene relazioni con 2-3 alberghi in ciascuna delle principali località, per avere alternative in base alla disponibilità e alla stagione.

Viaggio (450) ~7 viaggi/mese × 12 mesi × 5 anni + margine

Considerando una media di 7-8 viaggi al mese (maggiore frequenza in alta stagione, minore in bassa), per 5 anni.

Prenotazione (9000) ~20 prenotazioni medie × 450 viaggi

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

Ogni viaggio riceve mediamente 20 prenotazioni, considerando sia prenotazioni individuali che di gruppo.

Itinerario (100)~20 itinerari/anno × 5 anni (alcuni vengono rinnovati)

Gli itinerari vengono in parte riutilizzati e in parte rinnovati ogni anno; quindi, si stimano circa 20 itinerari nuovi all'anno.

Pernottamento(2000) ~4-5 notti medie per viaggio × 450 viaggi

Considerando che molti viaggi prevedono 3-5 pernottamenti, con alcuni viaggi più brevi (weekend) e altri più lunghi (fino a 7 giorni).

Autobus(60): Parco autobus limitato di compagnie partner

Assegnato (900): ~2 autobus medi per viaggio × 450 viaggi

Tappa (500): ~5 tappe medie per itinerario × 100 itinerari

Tavola delle operazioni

Cod.	Descrizione	Frequenza attesa
AV1	Registra Prenotazione	6/Giorno
AV2	Cancella Prenotazione	10/Mese
AV3	Ricerca Viaggi disponibili	50/Giorno
AM1	Generazione Report	2/Mese
AM2	Creazione Itinerario	10/Mese
AM3	Assegnazione Autobus	20/Giorno
AM4	Assegnazione Albergo	10/Giorno
AM5	Aggiunta Località	5/Mese
AM6	Aggiunta Albergo	5/Mese
AM7	Aggiunta Autobus	2/Mese
AM8	Creazione Viaggio	7/Mese
L1	Login	100/Giorno

Costo delle operazioni

AV1 - Registra Prenotazione: Legge informazioni su un viaggio e scrive una nuova prenotazione e la relazione con il viaggio

- Costo unitario dell'operazione: 1 lettura + 2 scritture × 2 = 5

AV2 – Cancella Prenotazione: Legge la prenotazione esistente e modifica il suo stato (cancellazione logica)

- Costo unitario dell'operazione: 1 lettura + 1 scrittura × 2 = 3

AV3- Ricerca viaggi disponibili: Lettura dei viaggi con stato *viaggio programmato*

- Costo unitario dell'operazione: 1 lettura

AM1- Generazione Report:

- Verifica viaggio terminato (1 lettura)
- Lettura del numero dei partecipanti (1 lettura)
- Lettura e calcolo del costo degli autobus (2 letture)
- Lettura e calcolo dei costi degli alberghi (2 letture)
- Lettura del costo dell'itinerario (1 lettura)

Costo unitario dell'operazione = 7 letture

AM2 – Creazione Itinerario

- Inserire ordine e giorni della tappa (1 scrittura)
- Cercare le località disponibili (1 lettura)

Questa operazione può essere fatta per n volte, ricordandoci del vincolo che un itinerario ha durata di al più sette giorni

- Creare il nuovo itinerario con tappe e costi (1 scrittutra)

Costo unitario dell'operazione = ((1 scritture) × 2 + (1 lettura) × 1) × n + (1 scrittura) × 2

AM3 – Assegnazione Autobus: Bisogna prima leggere il viaggio (1 lettura), contare i partecipanti del viaggio (2 letture), Ricercare gli autobus disponibili (2 letture) e assegnare degli autobus necessari (1 scrittura)

- Costo unitario dell'operazione: (5 letture × 1) + (1 scrittura × 2) = **7**

AM4- Assegnazione Albergo: Lettura del viaggio e dell'itinerario (2 letture), Ricerca degli alberghi disponibili nella località (2 letture), Verifica della capienza rispetto ai partecipanti (1 lettura) e Assegnazione dell'albergo al viaggio (1 scrittura)

- Costo unitario dell'operazione: (5 letture × 1) + (1 scrittura × 2) = **7**

AM5 – Aggiunta località: Verificare che la località non esista già (1 lettura) e inserire i dati della nuova località (1 scrittura)

- Costo unitario dell'operazione: 1 lettura + 1 scrittura × 2 = 3

AM6-Aggiunta albergo: Verificare che l'albergo non esista già (1 lettura), Verificare la località associata (1 lettura) e inserire i dati del nuovo albergo con recapiti e referente (1 scrittura)

- Costo unitario dell'operazione: (2 letture) × 1+ (1 scrittura) ×2 =4

AM7-Aggiunta autobus: Verificare che l'autobus non esista già (1 lettura) e inserire i dati del nuovo autobus (1 scrittura)

- Costo unitario dell'operazione: 1 lettura + 1 scritture × 2 = 3

AM8 – Creazione Viaggio: Cercare l'itinerario base (1 lettura), Verificare Se esiste già un viaggio con lo stesso itinerario e data (1 lettura) e creare il nuovo viaggio con le date specifiche

- Costo unitario dell'operazione: 2 letture + 1 scritture × 2 = 4

Ristrutturazione dello schema E-R

Lo schema E/R necessita ora di una fase di ristrutturazione, pur rimanendo evidente l'intento originario di limitare al minimo le ridondanze. Va sottolineato la presenza di un dato derivato nello schema ovvero *durata* dell'entità itinerario per ragioni prestazionali si decide di mantenerlo.
Un aspetto fondamentale di questa fase di ristrutturazione consisterà nella rimozione delle generalizzazioni, al fine di rendere il modello maggiormente compatibile con la successiva traduzione logica e fisica.

1. **Analisi delle ridondanze**

   Ridondanza nella relazione SVOLGE:

   Esaminando lo schema E-R, noto che la relazione SVOLGE che collega PERNOTTAMENTO a LOCALITÀ è potenzialmente ridondante. Infatti:

   - Un PERNOTTAMENTO è associato a un ALBERGO tramite la relazione IN
   - Un ALBERGO è situato in una LOCALITÀ (attributo CITTÀ)

   La decisione è quella di mantenere questa relazione essendo che migliora le prestazioni dell'operazione AM4 che ha frequenza elevata e il costo in termini di spazio è minimo rispetto al guadagno prestazionale

   L'attributo durata nell'entità itinerario: Può essere derivato contando il numero di pernottamenti associati all'itinerario o come differenza tra l'ultima e la prima data di tappa + 1.

   La decisone è quella di mantenere l'attributo per ottimizzare l'operazione AV3 con frequenza elevata e riduco la complessità computazionale durante le visualizzazioni

2. **Eliminazione delle generalizzazioni**

   Lo schema presenta una generalizzazione totale per LOCALITÀ in LOCALITÀ ITALIANE e LOCALITÀ EUROPEE.

   La decisione è di collassare i figli nel padre , quindi aggiungere gli attributi di LOCALITÀ ITALIANE e LOCALITÀ EUROPEE in LOCALITÀ rendendo gli attributi nullable per provincia e regione (nel caso fosse uno stato estero) e aggiungere alla chiave primaria Stato

Lo schema inoltre presenta una generalizzazione totale per VIAGGIO , la decisione è quella di far collassare i figli nel padre aggiungendo un attributo STATO, che ci dira in quale delle tre "fase di vita" si trova il viaggio.

3. **Partizionamento/accorpamento di concetti**

   Non è necessario partizionare entità o associazioni.

4. **Identificatori principali**

   Gli identificatori principali delle entità sono facilmente riconoscibili a livello grafico e risultano immediatamente individuabili grazie a una lettura attenta dello schema E/R, in linea con la notazione utilizzata.

   Essi sono stati scelti nel pieno rispetto delle regole di integrità del modello relazionale, assicurando in particolare il vincolo di non nullabilità.

   Si è data preferenza, ove possibile a identificatori "naturali" rispetto a identificatori virtuali, ad esempio l'identificatore dell'entità AUTOBUS l'identificatore scelto è Targa, essa identifica univocamente un mezzo di trasporto, mentre per altre entità si è usato un identificatore virtuale per non avere una chiave composta da diversi attributi.

   Per esempio, nell'entità Albergo si è scelto di identificarla CodiceAlbergo rispetto ad una chiave composta dal nome dell'albergo e dal suo indirizzo, per motivi di efficienza.

Trasformazione di attributi e identificatori

L'entità Pernottamento è identificata da una chiave composta che include gli identificatori esterni di Albergo, Viaggio e Località, a cui si aggiungono gli attributi check-in e check-out. Per facilitare la traduzione nello schema relazionale, questa chiave composta viene mantenuta come chiave primaria della tabella Pernottamento, e gli identificatori esterni vengono rappresentati come chiavi esterne che referenziano le rispettive tabelle.

Traduzione di entità e associazioni

VIAGGIO(IDViaggio, DataPartenza, DataRitorno, Stato, IDItinerario)

ITINERARIO(IDItinerario,Durata,Costo,NomeItinerario)

TAPPA(IDItinerario,NomeLocalità,Stato,Ordine,Giorni)

LOCALITA'(Nome, Stato ,Provincia,Regione)

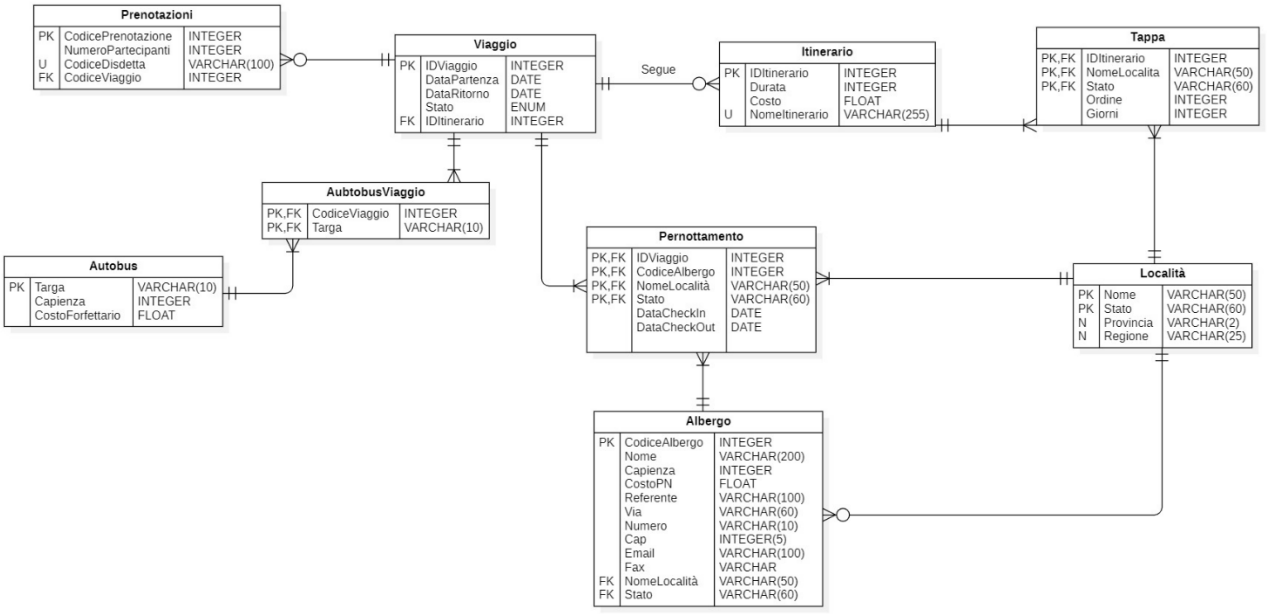
ALBERGO(CodiceAlbergo,Nome,Capienza,CostoPN,Referente,Via,NumeroCivico,CAP,email,fax, NomeLocalità,Stato)

PERNOTTAMENTO(IDViaggio,CodiceAlbergo,NomeLocalità,Stato,DataCheckIn,DataCheckOut)

PRENOTAZIONI (<u>CodicePrenotazione</u>,NumeroPartecipanti,CodiceDisdetta,CodiceViaggio)

AUTOBUS(<u>Targa</u>,Capienza,CostoForfettario)

AUTOBUSVIAGGO(<u>IDViaggio,Targa</u>)



Normalizzazione del modello relazionale

Il modello non è in forma normale, elenco di seguito dove non lo è e perché si è scelto di non normalizzarlo.

Tabella ~~*Tappa*~~ *Località* non è un terza forma normale essendo che esiste una relazione transitiva tra (nome, stato) provincia e regione. Per motivi di performance ed essendo che per le città europee l'attributo provincia e regione è presumibile sia null si decide di non creare una tabella *Provincie*

Tabella *Albergo* non è nella forma normale di Boyce-Codd essendo che esiste la dipendenza (Via,Numero,Cap) con Nomelocalità , Stato. Ma (Via,Numero,Cap) non è superchiave di albergo.
Per normalizzare il modello si dovrebbe inserire un'altra tabella *Indirizzi* collegata ad *Albergo* ma per motivi prestazionali si è deciso di non farlo e mantenere tutto in un'unica tabella.

Tabella *Pernottamento*, il codice dell'albergo identifica anche la località in cui si effettua il pernottamento. Tuttavia, per ragioni prestazionali, si decide di non normalizzare il modello.

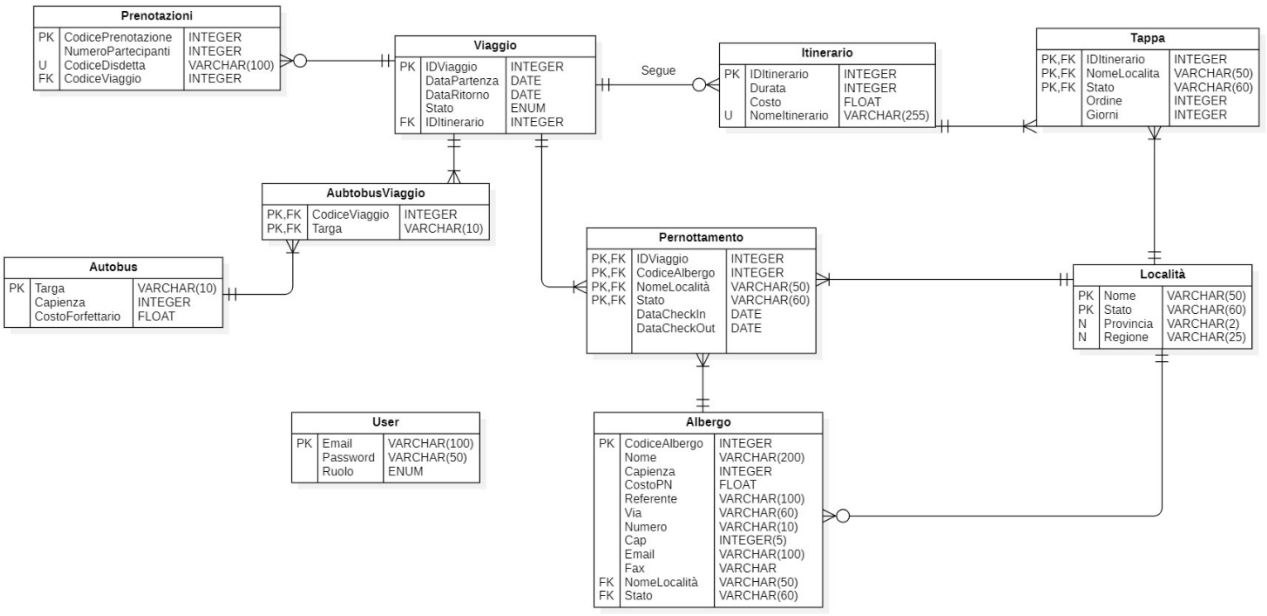
5. Progettazione fisica

Utenti e privilegi

Si decide di introdurre due tipi di utenti per l'applicazione:

- Segreteria
  - Grant in esecuzione sulle operazioni L1, AM1, AM2, AM3, AM4, AM5, AM6, AM7, AM8
- Agente di viaggio
  - Grant in esecuzione sulle operazioni L1,AG1,AG2,AG3

Per identificare gli utenti si aggiunge una tabella utenti , con attributi (email(PK),password,Ruolo)



Strutture di memorizzazione

Tabella Viaggio		
Colonna	**Tipo di dato**	**Attributi[2]**
IDViaggio	Integer	PK,AI
DataPartenza	Date	NN
DataRitorno	Date	NN
Stato	Enum('PROGRAMMATO','IN	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.

	CORSO’,’TERMINATO’)	
IDItenierario	Integer	NN

Tabella Itinerario		
Colonna	**Tipo di dato**	**Attributi**
IDItinerario	Integer	PK,AI
Durata	Integer	NN
Costo	Float	NN
NomeItinerario	Varchar(255)	UQ

Tabella Localita		
Colonna	**Tipo di dato**	**Attributi**
Nome	Varchar(50)	PK
Stato	Varchar(60)	PK
Provincia	Varchar(2)	
Regione	Varchar(25)	

Tabella Tappa		
Colonna	**Tipo di dato**	**Attributi**
Nome	Varchar(50)	PK
Stato	Varchar(60)	PK
IDIteneraio	Integer	PK
Ordine	Integer	NN
Giorni	Integer	NN

Tabella Prenotazione		
Colonna	**Tipo di dato**	**Attributi**
CodicePrenotazione	Integer	PK,AI
NumeroPartecipanti	Integer	NN
CodiceDisdetta	Varchar(20)	UQ
CodiceViaggio	Integer	NN

Tabella Autobus		
Colonna	**Tipo di dato**	**Attributi**
Targa	Varchar(10)	PK
Capienza	Integer	NN
CostoForfettario	Float	NN

Tabella AutobusViaggio		
Colonna	**Tipo di dato**	**Attributi**
Targa	Varchar(10)	PK

IDViaggio	Integer	PK

Tabella Albergo		
Colonna	**Tipo di dato**	**Attributi**
CodiceAlbergo	Integer	PK,AI
Nome	Varchar(200)	NN
Capienza	Integer	NN
CostoPN	Float	NN
Referente	Varchar(100)	NN
Via	Varchar(60)	NN
Numero	Varchar(10)	NN
Cap	Varchar(20)	
Email	Varchar(100)	NN
Fax	Varchar(20)	NN
NomeLocalità	Varchar(50)	NN
Stato	Varchar(60)	NN

Tabella Pernottamento		
Colonna	**Tipo di dato**	**Attributi**
CodiceAlbergo	Integer	PK
IDViaggio	Integer	PK
Nome	Varchar(50)	PK
Stato	Varchar(60)	PK
DataCheckIn	Date	NN
DataCheckOut	Date	NN

Tabella Utente		
Colonna	**Tipo di dato**	**Attributi**
Email	Varchar(100)	PK
Password	Varchar(50)	NN
Ruolo	Enum(‘AV’,’AM’)	NN

Indici

Tabella Albergo	
Indice Località_Albergo	**Tipo[3]:**
NomeLocalità,Stato	IDX

Motivazione: Chiave esterna verso Località. Ottimizza la ricerca di alberghi disponibili in una specifica località, riducendo drasticamente il costo dell'operazione di assegnazione albergo (AM4).

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

Tabella Utente	
Indice Login	**Tipo:**
Email, Password	IDX

Tabella Tappa	
Indice Itinerario_Tappa	**Tipo:**
IDItinerario	IDX

Motivazione: Essenziale per recuperare velocemente tutte le tappe che compongono un itinerario, operazione fondamentale durante la creazione (AM2) e la consultazione di un itinerario (AM4).

Tabella Prenotazione	
Indice CodiceDisdettaPrenotazione	**Tipo:**
CodiceDisdetta	UQ

Motivazione: Garantisce l'unicità del codice di disdetta e rende la ricerca per la cancellazione (AV2) quasi istantanea, riducendo il costo da una scansione completa a una singola lettura mirata.

Tabella Pernottamento	
Indice Pernottamento_Viaggio	**Tipo:**
IDViaggio	IDX

Motivazione: Accelera la ricerca di tutti i pernottamenti (e quindi degli alberghi) associati a un viaggio, operazione cruciale per il calcolo dei costi (AM1) e la gestione logistica (AM4).

Tabella Viaggio	
Indice StatoViaggio	**Tipo:**
Stato	IDX

Motivazione : La colonna Stato è usata frequentemente per verificare lo stato di vita del viaggio

Trigger

Descrivere quali trigger sono stati implementati, mostrando il codice SQL per la loro instanziazione. Si faccia riferimento al fatto che il DBMS di riferimento richiede di utilizzare trigger anche per realizzare vincoli di check ed asserzioni.

Trigger per la tabella *autobusviaggio* per controllare che l'autobus scelto sia disponibile per i giorni del viaggio

```
1. create definer = root@localhost trigger check_autobus_disponibile
2.     before insert
3.     on autobusviaggio
4.     for each row
5. BEGIN
```

```
    DECLARE viaggio_start DATE;
    DECLARE viaggio_end DATE;

    -- Ottieni le date del viaggio che si vuole assegnare all'autobus
    SELECT DataPartenza, DataRientro
    INTO viaggio_start, viaggio_end
    FROM viaggio
    WHERE IDViaggio = NEW.CodiceViaggio;
    -- Verifica se l'autobus è già assegnato ad altri viaggi con date sovrapposte
    IF EXISTS (
        SELECT 1
        FROM autobusviaggio av
                 JOIN viaggio v ON av.CodiceViaggio = v.IDViaggio
        WHERE av.Targa = NEW.Targa
          AND (
            (v.DataPartenza <= viaggio_end AND v.DataRientro >= viaggio_start)
            )
    )
    THEN
        SIGNAL SQLSTATE '45001'
            SET MESSAGE_TEXT = 'Autobus non disponibile per le date del viaggio selezionato!';
    END IF;
END;

```

Trigger per la tabella *pernottamento* dove andiamo a controllare prima di inserire il pernottamento che l'albergo scelto abbia un numero di posti maggiore o uguale al numero di partecipanti del viaggio

```
create definer = root@localhost trigger check_albergo_capienza
    before insert
    on pernottamento
    for each row
BEGIN
    DECLARE tot_partecipanti INT;
    DECLARE albergo_capienza INT;

    -- Calcola il numero totale di partecipanti al viaggio
    SELECT IFNULL(SUM(NumeroPartecipanti),0)
    INTO tot_partecipanti
    FROM prenotazioni
    WHERE CodiceViaggio = NEW.IDViaggio;

    -- Prendi la capienza dell'albergo selezionato
    SELECT Capienza
    INTO albergo_capienza
    FROM albergo
    WHERE CodiceAlbergo = NEW.CodiceAlbergo;

    -- Se l'albergo non ha abbastanza posti, blocca l'inserimento
    IF albergo_capienza < tot_partecipanti THEN
        SIGNAL SQLSTATE '45001'
```

```
24.                 SET MESSAGE_TEXT = 'La capienza dell\'albergo selezionato non è sufficiente per i
partecipanti al viaggio!';
25.     END IF;
26. END;
27.
```

Trigger sulla tabella *prenotazione* dove si controlla prima di inserire una nuova prenotazione, che non venga fatta nei 20 giorni antecedenti alla partenza

```
 1. create definer = root@localhost trigger check_prenotazione
 2.     before insert
 3.     on prenotazioni
 4.     for each row
 5. BEGIN
 6.     DECLARE var_StartDate DATE;
 7.
 8.     -- Recupera la data di partenza del viaggio associato
 9.     SELECT DataPartenza
10.     INTO var_StartDate
11.     FROM viaggio
12.     WHERE IDViaggio = NEW.CodiceViaggio
13.     LIMIT 1;
14.
15.     -- Se la partenza è nei prossimi 20 giorni, blocca l'inserimento
16.     IF DATEDIFF(var_StartDate, CURDATE()) <= 20 THEN
17.         SIGNAL SQLSTATE '45001'
18.             SET MESSAGE_TEXT = 'Impossibile effettuare una prenotazione nei venti giorni prima della
partenza';
19.     END IF;
20. END;
21.
```

Trigger sulla tabella *prenotazione* dove si controlla prima di cancellare una prenotazione che non venga fatto nei venti giorni prima della partenza

```
 1. create definer = root@localhost trigger check_cancellation
 2.     before delete
 3.     on prenotazioni
 4.     for each row
 5. begin
 6.     declare var_StartDate date;
 7.     declare var_status  varchar(20);
 8.     -- Recupero la data e lo stato del viaggio legato alla prenotazione che sto cancellando
 9.     SELECT v.DataPartenza, v.Stato
10.     INTO var_StartDate, var_status
```

```
11.     FROM viaggio v
12.     WHERE v.IDViaggio = OLD.CodiceViaggio
13.     LIMIT 1;
14.
15.     IF var_status = 'INCORSO' AND DATEDIFF(var_StartDate, NOW()) <= 20 THEN
16.         SIGNAL SQLSTATE '45001'
17.             SET MESSAGE_TEXT = 'Impossibile cancellare una prenotazione nei venti giorni prima della
partenza';
18.     ELSEIF var_status = 'TERMINATO' THEN
19.         SIGNAL SQLSTATE '45001'
20.             SET MESSAGE_TEXT = 'Impossibile cancellare una prenotazione di un viaggio terminato';
21.     END IF;
22. end;
23.
```

Trigger sulla tabella *viaggio* dove prima di inserire un nuovo viaggio si esegue un controllo sulla coerenza delle date

```
 1. create definer = root@localhost trigger check_date_viaggio
 2.     before insert
 3.     on viaggio
 4.     for each row
 5. begin
 6.         if NEW.DataRientro<NEW.DataPartenza then signal sqlstate '45001' set message_text = 'La data
del rientro non può
 7. precedere la data di partenza';
 8.  end if;
 9.         end;
10.
```

Eventi

Per automatizzare il ciclo di vita di un viaggio utilizzo gli eventi per verificare quanti giorni manchino alla data di partenza, cosi da aggiornare lo stato di vita di viaggio.

Il primo evento che inserisco è per modificare lo stato di un viaggio quando mancano venti giorni alla data della partenza.

```
 1. create event aggiorna_viaggi_confermati on schedule
 2.     every '1' DAY
 3.         starts '2025-08-13 17:36:33'
 4.     enable
 5.     do
 6.     UPDATE viaggio
 7.     SET Stato = 'INCORSO'
 8.     WHERE Stato = 'PROGRAMMATO'
 9.       AND DATEDIFF(DataPartenza, CURDATE()) = 20;
10.
```

Inoltre, creo un evento con periodicità giornaliera dove viene aggiornato l'attributo stato per tutti i viaggi dove la data corrente è dopo la data di rientro del viaggio

```
create event aggiorna_viaggi_terminati on schedule
    every '1' DAY
        starts '2025-08-13 17:36:33'
    enable
    do
    UPDATE viaggio
    SET Stato = 'TERMINATO'
    WHERE Stato = 'INCORSO'
      AND DATEDIFF(DataPartenza, CURDATE()) = -1;

```

Per policy aziendale inseriamo un evento che elimina tutti i viaggi terminati da più di 5 anni e di conseguenza tutto ciò che è collegato ad un viaggio come: prenotazioni, pernottamenti e autobus associati al viaggio

```
create definer = root@localhost event elimina_viaggi_vecchi on schedule
    every '1' DAY
        starts '2025-08-23 15:55:26'
    enable
    do
    DELETE FROM viaggio
    WHERE DataRientro < DATE_SUB(CURDATE(), INTERVAL 5 YEAR);

```

Viste

Non sono state utilizzate viste.

Stored Procedures e transazioni

1. Aggiungi tappa itinerario

```
create
    definer = root@localhost procedure aggiungiTappaItinerario(IN p_IDItinerario
int, IN p_NomeLocalita varchar(50),
                                                               IN p_Stato
varchar(60), IN p_Ordine int, IN p_Giorni int)
BEGIN
        DECLARE v_DurataItinerario INT;
        DECLARE v_SommaGiorni INT;
        DECLARE v_MaxOrdine INT;

        DECLARE EXIT HANDLER FOR SQLEXCEPTION
        BEGIN
            ROLLBACK;
            RESIGNAL;
        END;

        SET TRANSACTION ISOLATION LEVEL SERIALIZABLE ;
        START TRANSACTION;

        -- Controllo esistenza itinerario
        IF NOT EXISTS (SELECT 1 FROM itinerario WHERE IDItinerario =
p_IDItinerario) THEN
```

```
            SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Itinerario non trovato!';
        -- Controllo esistenza località
        ELSEIF NOT EXISTS (SELECT 1 FROM localita WHERE Nome= p_NomeLocalita AND
Stato = p_Stato) THEN
            SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Località non trovata!';
        -- Controllo ordine consecutivo
        ELSE
            SELECT IFNULL(MAX(Ordine), 0) INTO v_MaxOrdine
            FROM tappa
            WHERE IDItinerario = p_IDItinerario;

            -- Se l'ordine non è consecutivo rispetto all'ultimo inserito
            IF p_Ordine <> v_MaxOrdine + 1 THEN
                SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Ordine non
consecutivo!';
            ELSE
                -- Controllo durata itinerario
                SELECT Durata INTO v_DurataItinerario
                FROM itinerario
                WHERE IDItinerario = p_IDItinerario;

                IF p_Giorni > v_DurataItinerario THEN
                    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Durata tappa
superiore alla durata itinerario!';
                ELSE
                    -- Controllo somma giorni tappe
                    SELECT IFNULL(SUM(Giorni), 0) INTO v_SommaGiorni
                    FROM tappa
                    WHERE IDItinerario = p_IDItinerario;

                    -- Se la somma dei giorni delle tappe supera la durata
dell'itinerario
                    -- o se la somma dei giorni delle tappe più i giorni della
nuova tappa
                    -- supera 7 giorni, l'inserimento non è consentito
                    IF p_Giorni=0 THEN
                        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'I giorni
della tappa non possono essere zero!';
                    ELSEIF v_SommaGiorni + p_Giorni > 7 OR v_SommaGiorni +
p_Giorni > v_DurataItinerario THEN
                        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Somma giorni
tappe superiore alla durata itinerario!';
                    ELSE
                        INSERT INTO tappa (IDItinerario, NomeLocalita, Stato,
Ordine, Giorni)
                        VALUES (p_IDItinerario, p_NomeLocalita, p_Stato,
p_Ordine, p_Giorni);
                        COMMIT;
                    END IF;
                END IF;
            END IF;
        END IF;
    END;
```

Livello di isolamento: SERIALIZABLE

La procedura esegue molteplici controlli di coerenza, tra cui la verifica dell'esistenza dell'itinerario, della località, dell'ordine consecutivo e dei vincoli di durata imposti dalla specifica.

La scelta del livello di isolamento massimo è dettata dalla necessità che tutti questi controlli avvengano in un contesto completamente isolato, prevenendo letture fantasma che potrebbero verificarsi se un'altra transazione modificasse l'itinerario o le tappe durante l'esecuzione.

Funzionalità operativa della procedura: Dopo aver creato un nuovo itinerario con tutte le informazioni accessorie (costo , durata e nome), la segreteria aggiunge le tappe dell'itinerario. Nella procedura vengono fatti inizialmente i controlli per accertarsi dell'esistenza della località inserita, se questa è presente si passa al controllo della sequenzialità della tappa e all'accertamento che il totale dei giorni delle tappe precedenti insieme a quella che si sta per aggiungere non superi il limite di 7 giorni.

2. Associa autobus al viaggio

```
create
    definer = root@localhost procedure associaAutobusViaggio(IN p_CodiceViaggio
int, IN p_Targa varchar(10),
                                                              OUT p_Messaggio
varchar(255))
BEGIN


    DECLARE EXIT HANDLER FOR SQLEXCEPTION
        BEGIN
            ROLLBACK;
            RESIGNAL;
        END;
    SET TRANSACTION ISOLATION LEVEL REPEATABLE READ ;
    START TRANSACTION;

    -- Controllo se il viaggio esiste
    IF NOT EXISTS (SELECT 1 FROM viaggio WHERE IDViaggio = p_CodiceViaggio) THEN
        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Viaggio non trovato!';
    ELSEIF (SELECT viaggio.Stato from viaggio WHERE viaggio.IDViaggio =
p_CodiceViaggio) <> 'INCORSO' THEN
        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Impossibile associare un autobus ad un viaggio se
non mancano meno di 20 giorni alla data di partenza!';
    -- Controllo se l'autobus esiste
    ELSEIF NOT EXISTS (SELECT 1 FROM autobus WHERE Targa = p_Targa) THEN
        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Autobus non trovato con questa targa!';
    -- Controllo se l'autobus è già associato al viaggio
    ELSEIF EXISTS (SELECT 1 FROM autobusviaggio WHERE CodiceViaggio =
p_CodiceViaggio AND Targa = p_Targa) THEN
```

```
        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Autobus già associato a questo viaggio!';
        -- Controllo se la somma dei posti degli autobus assocciati è almeno
uguale  al numero di partecipanti
    ELSEIF  (SELECT sum(NumeroPartecipanti) FROM prenotazioni WHERE CodiceViaggio
= p_CodiceViaggio) > (SELECT COALESCE(SUM(Capienza),0) +

(SELECT Capienza FROM autobus WHERE Targa = p_Targa)

FROM autobus join agenziadiviaggi.autobusviaggio a on autobus.Targa = a.Targa
join agenziadiviaggi.viaggio v on v.IDViaggio = a.CodiceViaggio

WHERE a.CodiceViaggio = p_CodiceViaggio)
       THEN
        -- Associo l'autobus al viaggio
        INSERT INTO autobusviaggio (CodiceViaggio, Targa)
        VALUES (p_CodiceViaggio, p_Targa);
        SET p_Messaggio = 'La somma dei posti degli autobus associati è inferiore
al numero di partecipanti, aggiungi un altro autobus!';
        COMMIT;

    ELSE
        -- Associo l'autobus al viaggio
        INSERT INTO autobusviaggio (CodiceViaggio, Targa)
        VALUES (p_CodiceViaggio, p_Targa);
        SET p_Messaggio = 'Autobus associato al viaggio con successo!';
        COMMIT;
    END IF;
END;
```

Livello di isolamento : REPETABLE READ;

La procedura verifica se esiste l'autobus selezionato e se la somma del numero dei posti degli autobus 'allocati' sommata alla capienza di quello che si sta inserendo sia maggiore o uguale al numero dei partecipanti al viaggio. REPEATABLE READ previene che altre transazioni possano modificare i dati letti durante l'esecuzione, garantendo decisioni basate su uno stato coerente del database.

Funzionalità operativa: Associa un autobus ad un viaggio specifico. Inoltre, controlla se il viaggio è nella sua fase "INCORSO" ovvero se mancano meno di venti giorni dalla data di partenza, la modifica del ciclo di vita di un viaggio è gestita dagli eventi inseriti, che ogni giorno verificano per ogni viaggio quanti giorni mancano alla data di partenza.

3. Associa un pernottamento al viaggio

```
create
    definer = root@localhost procedure associaPernottamento(IN p_IDViaggio int,
IN p_CodiceAlbergo int)
BEGIN
```

```
    DECLARE v_NomeLocAlbergo varchar(50);
    DECLARE v_StatoAlbergo varchar(60);
    DECLARE v_DataCheckIn DATE;
    DECLARE v_DataCheckOut DATE;
    DECLARE v_Durata INT;
    DECLARE v_GiorniPrec INT;
    -- Dichiaro il gestore di eccezioni per la transazione
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
        BEGIN
            ROLLBACK;
            RESIGNAL;
        END;
    SET TRANSACTION ISOLATION LEVEL SERIALIZABLE ;
    START TRANSACTION;

    -- Recupero il nome località dell'albergo
    SELECT albergo.NomeLocalita, albergo.Stato INTO v_NomeLocAlbergo,
v_StatoAlbergo
    FROM albergo
    WHERE albergo.CodiceAlbergo = p_CodiceAlbergo;

    -- Controllo se il viaggio esiste
    IF NOT EXISTS (SELECT 1 FROM viaggio WHERE IDViaggio = p_IDViaggio) THEN
        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Viaggio non trovato!';
    ELSEIF (SELECT viaggio.Stato from viaggio WHERE viaggio.IDViaggio =
p_IDViaggio) <> 'INCORSO' THEN
        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Impossibile associare un Pernottamento ad un
viaggio se non mancano meno di 20 giorni alla data di partenza!';

        -- Controllo se l'albergo esiste
    ELSEIF NOT EXISTS (SELECT 1 FROM albergo WHERE CodiceAlbergo =
p_CodiceAlbergo) THEN
        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Albergo non trovato con questo codice!';
    -- Controllo se la località esiste
    ELSEIF NOT EXISTS (SELECT 1 FROM localita WHERE Nome = v_NomeLocAlbergo AND
Stato = v_StatoAlbergo) THEN
        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Località non trovata!';
    -- Controllo se il pernottamento esiste già
    ELSEIF EXISTS (SELECT 1 FROM pernottamento WHERE IDViaggio = p_IDViaggio  AND
NomeLocalita = v_NomeLocAlbergo AND Stato = v_StatoAlbergo) THEN
        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Pernottamento già associato a questo viaggio!';
    ELSE


       -- Calcolo la somma dei giorni delle tappe precedenti
        SELECT IFNULL(SUM(Giorni), 0) INTO v_GiorniPrec
        FROM tappa
        WHERE IDItinerario = (
            SELECT IDItinerario FROM viaggio WHERE IDViaggio = p_IDViaggio
        )
        AND Ordine < (
            SELECT Ordine FROM tappa
            WHERE IDItinerario = (
                SELECT IDItinerario FROM viaggio WHERE IDViaggio = p_IDViaggio
            )
            AND NomeLocalita = v_NomeLocAlbergo AND Stato = v_StatoAlbergo
```

```
        );

        -- Calcolo la durata della tappa corrente
        SELECT Giorni INTO v_Durata
        FROM tappa
        WHERE IDItinerario = (
            SELECT IDItinerario FROM viaggio WHERE IDViaggio = p_IDViaggio
        )
        AND NomeLocalita = v_NomeLocAlbergo AND Stato = v_StatoAlbergo;

        -- Calcolo le date di check-in e check-out
        SELECT DataPartenza + INTERVAL v_GiorniPrec DAY INTO v_DataCheckIn
        FROM viaggio WHERE IDViaggio = p_IDViaggio;

        SET v_DataCheckOut = v_DataCheckIn + INTERVAL v_Durata DAY;

        -- Inserisco il pernottamento
        INSERT INTO pernottamento (IDViaggio, CodiceAlbergo, NomeLocalita, Stato,
DataCheckIn, DataCheckOut)
        VALUES (p_IDViaggio, p_CodiceAlbergo, v_NomeLocAlbergo, v_StatoAlbergo,
v_DataCheckIn, v_DataCheckOut);
        COMMIT;
    END IF;

end;
```

Livello di isolamento:SERIALIZABLE

La procedura esegue oltre ai controlli di esistenza dei dati inseriti anche i calcoli rispetto alle date di checkIn e checkOut. È stato scelto di rendere così complessa computazionalmente questa procedura così da evitare problemi di correttezza nelle date. SERIALIZABLE garantisce che questi calcoli avvengano in un contesto completamente isolato, prevenendo interferenze che potrebbero compromettere la correttezza dei risultati.

4. Cancella prenotazione

```
create
    definer = root@localhost procedure cancellazionePrenotazione(IN
p_codiceDisdetta varchar(20))
BEGIN
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
        BEGIN
            ROLLBACK;
            RESIGNAL;
        END;
    SET TRANSACTION ISOLATION LEVEL REPEATABLE READ ;
    START TRANSACTION;


    IF NOT EXISTS (
        SELECT 1 FROM prenotazioni WHERE CodiceDisdetta = p_codiceDisdetta
```

```
    ) THEN
        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Prenotazione non trovata con questo codice di
disdetta!';
    ELSE
        DELETE FROM prenotazioni WHERE CodiceDisdetta = p_codiceDisdetta;
        COMMIT;
    END IF;
END;
```

Livello di isolamento: REPEATABLE READ

Anche se l'operazione è relativamente semplice, REPEATABLE READ garantisce che la prenotazione identificata per la cancellazione non venga modificata da altre transazioni durante l'esecuzione.

Funzionalità operativa: Cancella una prenotazione esistente utilizzando il suo codice di disdetta univoco.

5. Inserisci Prenotazione

```
create
    definer = root@localhost procedure inserisciPrenotazione(IN
p_NumeroPartecipanti int, IN p_IDViaggio int,
                                                        OUT p_CodiceDisdetta
varchar(100))
BEGIN
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
        BEGIN
            ROLLBACK;
            RESIGNAL;
        END;

    SET TRANSACTION ISOLATION LEVEL REPEATABLE READ ;
    START TRANSACTION;

    -- Controllo se il viaggio esiste
    IF NOT EXISTS (SELECT 1 FROM viaggio WHERE IDViaggio = p_IDViaggio) THEN
        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Viaggio non trovato!';
    ELSEIF p_NumeroPartecipanti <= 0 THEN
        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Il numero di partecipanti deve essere maggiore di
zero!';
    ELSE
        -- Genero un codice di disdetta unico
        SET p_CodiceDisdetta = CONCAT('DIS', LPAD(FLOOR(RAND() * 1000000), 6,
'0'));
        -- Controllo se il codice di disdetta esiste già
        WHILE EXISTS (SELECT 1 FROM prenotazioni WHERE CodiceDisdetta =
p_CodiceDisdetta) DO
            SET p_CodiceDisdetta = CONCAT('DIS', LPAD(FLOOR(RAND() * 1000000), 6,
'0'));
        END WHILE;

        -- Inserisco la prenotazione
        INSERT INTO prenotazioni (NumeroPartecipanti, CodiceDisdetta,
```

```
CodiceViaggio)
        VALUES (p_NumeroPartecipanti, p_CodiceDisdetta, p_IDViaggio);
        COMMIT;
    END IF;
end;
```

Livello di isolamento: REPEATABLE READ

REPEATABLE READ garantisce che il codice di disdetta generato sia unico e che il viaggio esista durante tutta la transazione.

6. Crea itinerario

```
create
    definer = root@localhost procedure creaItinerario(IN p_Durata int, IN p_Costo
float,
                                                       IN p_NomeItinerario
varchar(255), OUT itinerarioID int)
BEGIN

    -- Dichiaro il gestore di eccezioni per la transazione
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
        BEGIN
            ROLLBACK;
            RESIGNAL;
        END;

    -- Imposto il livello di isolamento della transazione
    SET TRANSACTION ISOLATION LEVEL REPEATABLE READ ;
    START TRANSACTION;

    -- Controllo se l'itinerario esiste già
    IF EXISTS (SELECT 1 FROM itinerario WHERE NomeItinerario = p_NomeItinerario)
THEN
        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Itinerario già presente con questo nome!';
    ELSEIF p_Durata>7 OR p_Durata<1 THEN
        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Durata non valida! Deve essere compresa tra 1 e 7
giorni.';
    ELSEIF p_Costo<0 THEN
        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Costo non valido! Deve essere un valore
positivo.';
    ELSE
        -- Inserisco l'itinerario nella tabella
         INSERT INTO itinerario (Durata, Costo, NomeItinerario)
        VALUES (p_Durata, p_Costo, p_NomeItinerario);
        -- Recupero l'ID dell'itinerario appena inserito
        SET itinerarioID = LAST_INSERT_ID();
    END IF;
    -- Commit della transazione
    COMMIT;
end;
```

Livello di isolamento: REPEATABLE READ

La procedura verifica l'unicità del nome dell'itinerario. REPEATABLE READ garantisce che questa condizione rimanga valida tra la verifica e l'inserimento, prevenendo race condition.

Funzionalità operativa: Crea un nuovo itinerario dopo aver verificato che non ne esista già uno con lo stesso nome e che i parametri di durata e costo siano validi.

7. Crea Viaggio

```
create
    definer = root@localhost procedure creaViaggio(IN p_DataPartenza date, IN
p_IDItinerario int, OUT p_IDViaggio int,
                                                   OUT p_DataRientro date)
BEGIN
    DECLARE v_Durata INT;
    DECLARE v_DataRientro DATE;
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
        BEGIN
            ROLLBACK;
            RESIGNAL;
        END;

    SET TRANSACTION ISOLATION LEVEL REPEATABLE READ ;
    START TRANSACTION;

    -- Controllo se l'itinerario esiste
    IF NOT EXISTS (SELECT 1 FROM itinerario WHERE IDItinerario = p_IDItinerario)
THEN
        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Itinerario non trovato!';
    ELSE
        -- Controllo se esiste già un viaggio programmato per l'itinerario
        IF EXISTS (SELECT 1 FROM viaggio WHERE IDItinerario = p_IDItinerario AND
DataPartenza = p_DataPartenza) THEN
            SIGNAL SQLSTATE '45000'
                SET MESSAGE_TEXT = 'Esiste già un viaggio programmato per questo
itinerario con questa data di partenza!';
        ELSEIF
            -- Controllo se la data di partenza è nel passato
            DATEDIFF(p_DataPartenza,CURDATE()) <= 20 THEN
            SIGNAL SQLSTATE '45000'
                SET MESSAGE_TEXT = 'La data di partenza deve essere almeno 20
giorni nel futuro!';
        ELSE
            -- Calcolo la data di rientro aggiungendo la durata dell'itinerario
            SELECT Durata INTO v_Durata FROM itinerario WHERE IDItinerario =
p_IDItinerario;
             SET v_DataRientro = DATE_ADD(p_DataPartenza, INTERVAL v_Durata DAY);
            -- Inserisco il nuovo viaggio
            INSERT INTO viaggio (DataPartenza, DataRientro, Stato, IDItinerario)
            VALUES (p_DataPartenza, v_DataRientro, 'PROGRAMMATO',
p_IDItinerario);

            -- Recupero l'ID del nuovo viaggio
            SET p_IDViaggio = LAST_INSERT_ID();
            SET p_DataRientro = v_DataRientro;
            COMMIT;
        END IF;
```

```
    END IF;
end;
```

Livello di isolamento: REPEATABLE READ

I controlli sulla combinazione itinerario/data richiedono REPEATABLE READ per garantire che nessun'altra transazione possa inserire un viaggio con gli stessi parametri tra la verifica e l'inserimento.

Funzionalità operativa: Programma un nuovo viaggio per un dato itinerario in una specifica data di partenza, calcolando la data di rientro.

8. Genera Report Viaggio

```
create
    definer = root@localhost procedure generaReportViaggio(IN p_IDViaggio int,
OUT p_Report text)
BEGIN
    DECLARE v_NomeItinerario VARCHAR(255);
    DECLARE v_DataPartenza DATE;
    DECLARE v_DataRientro DATE;
    DECLARE v_CostoAlberghi FLOAT;
    DECLARE v_CostoAutobus FLOAT;
    DECLARE v_CostoTotale FLOAT;
    DECLARE v_CostoPerPartecipante FLOAT;
    DECLARE v_NumeroPartecipanti INT;
    DECLARE v_EntratePartecipanti FLOAT;
    DECLARE v_Esito ENUM('Profitto','Perdita');
    DECLARE v_Bilancio FLOAT;

    -- Dichiaro il gestore di eccezioni per la transazione
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
        BEGIN
            ROLLBACK;
            RESIGNAL;
        END;
    SET TRANSACTION ISOLATION LEVEL REPEATABLE READ ;
    START TRANSACTION;

    -- Controllo se il viaggio esiste
    IF NOT EXISTS (SELECT 1 FROM viaggio WHERE IDViaggio = p_IDViaggio) THEN
        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Viaggio non trovato!';
    ELSEIF
        -- Controllo se il viaggio è già concluso
        (SELECT Stato FROM viaggio WHERE IDViaggio = p_IDViaggio) <> 'TERMINATO'
THEN
        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Il viaggio non è ancora concluso!';
    ELSE
        -- Mi ricavo il numero di partecipanti
        SELECT sum(NumeroPartecipanti) INTO v_NumeroPartecipanti
        FROM prenotazioni
        WHERE CodiceViaggio = p_IDViaggio;

        -- Mi ricavo il nome dell'itinerario e il costo totale
```

```
        SELECT i.NomeItinerario, i.Costo INTO v_NomeItinerario,
v_CostoPerPartecipante
        FROM itinerario i
        JOIN viaggio v ON i.IDItinerario = v.IDItinerario
        WHERE v.IDViaggio = p_IDViaggio;


        -- Mi ricavo le date di partenza e rientro
        SELECT DataPartenza, DataRientro INTO v_DataPartenza, v_DataRientro
        FROM viaggio
        WHERE IDViaggio = p_IDViaggio;

        -- Calcolo il numero di notti
        SET @numNotti = DATEDIFF(v_DataRientro, v_DataPartenza) + 1;

        -- Calcolo il costo totale degli alberghi (tutte le strutture, tutte le
notti, tutti i partecipanti)
        SELECT COALESCE(SUM(a.CostoPN),0) * @numNotti *
COALESCE(v_NumeroPartecipanti,0) INTO v_CostoAlberghi
        FROM pernottamento p
                JOIN albergo a ON p.CodiceAlbergo = a.CodiceAlbergo
        WHERE p.IDViaggio = p_IDViaggio;

        -- Calcolo il costo totale degli autobus
        SELECT SUM(a.CostoForfettario)*(DATEDIFF(v_DataRientro, v_DataPartenza))
INTO v_CostoAutobus
        FROM autobus a
        JOIN autobusviaggio av ON a.Targa = av.Targa
        WHERE av.CodiceViaggio = p_IDViaggio;

        -- Calcolo il costo totale del viaggio
        SET v_CostoTotale = v_CostoAlberghi + v_CostoAutobus;

        -- Calcolo le entrate totali dai partecipanti
        SET v_EntratePartecipanti = v_CostoPerPartecipante *
v_NumeroPartecipanti;

        -- Calcolo il bilancio
        SET v_Bilancio = v_EntratePartecipanti - v_CostoTotale;
        -- Determino l'esito del viaggio
        IF v_Bilancio > 0 THEN
            SET v_Esito = 'Profitto';
        ELSEIF v_Bilancio < 0 THEN
            SET v_Esito = 'Perdita';
        ELSE
            SET v_Esito = 'Pareggio';
        END IF;
        -- Creo il report
        SET p_Report = CONCAT(
            'Report Viaggio ID: ', p_IDViaggio, '\n',
            'Nome Itinerario: ', v_NomeItinerario, '\n',
            'Data Partenza: ', v_DataPartenza, '\n',
            'Data Rientro: ', v_DataRientro, '\n',
            'Numero Partecipanti: ', v_NumeroPartecipanti, '\n',
            'Costo per Partecipante: ', v_CostoPerPartecipante, '\n',
            'Costo Totale Alberghi: ', v_CostoAlberghi, '\n',
            'Costo Totale Autobus: ', v_CostoAutobus, '\n',
            'Costo Totale Viaggio: ', v_CostoTotale, '\n',
            'Entrate Totali Partecipanti: ', v_EntratePartecipanti, '\n',
            'Bilancio: ', v_Bilancio, '\n',
            'Esito: ', v_Esito
```

```
        );
        COMMIT;
    END IF;
END;
```

Livello di isolamento: REPETABLE READ

La generazione del report richiede letture consistenti di molteplici tabelle correlate. REPEATABLE READ garantisce che i dati letti siano consistenti durante tutta l'esecuzione della procedura, essenziale per calcoli finanziari accurati.

Funzionalità operativa: Calcola e restituisce un report testuale riassuntivo (costi, entrate, bilancio) per un viaggio che deve essere già concluso.

9. Inserisci località

```
create
    definer = root@localhost procedure inserisciLocalita(IN p_Nome varchar(50),
IN p_Stato varchar(60),
                                                         IN p_Provincia
varchar(2), IN p_Regione varchar(25))
BEGIN
    -- Dichiaro il gestore di eccezioni per la transazione
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
        BEGIN
            ROLLBACK;
            RESIGNAL;
        END;

    -- Imposto il livello di isolamento della transazione
    SET TRANSACTION ISOLATION LEVEL REPEATABLE READ ;
    START TRANSACTION;

    -- Controllo se la località esiste già
    IF EXISTS (SELECT 1 FROM localita WHERE Nome = p_Nome AND Stato = p_Stato)
THEN
        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Località già presente con questo nome e stato!';
    ELSE
        INSERT INTO localita (Nome, Stato, Provincia, Regione)
        VALUES (p_Nome, p_Stato, p_Provincia, p_Regione);
    END IF;

    COMMIT;
end;
```

Livello di isolamento: REPETABLE READ

Il livello scelto garantisce che la condizione da verificare rimanda vera fino al momento dell'inserimento.

Funzionalità operativa: Procedura di inserimento di una località

10. Inserisci Autobus

```
create
    definer = root@localhost procedure inserisciAutobus(IN p_Targa varchar(10),
IN p_Capienza int, IN p_CostoForfettario float)
BEGIN
    -- Dichiaro il gestore di eccezioni per la transazione
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
        BEGIN
            ROLLBACK;
            RESIGNAL;
        END;

    -- Imposto il livello di isolamento della transazione
    SET TRANSACTION ISOLATION LEVEL READ COMMITTED ;
    START TRANSACTION;

    -- Controllo se la targa esiste già
    IF EXISTS (SELECT 1 FROM autobus WHERE Targa = p_Targa) THEN
        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Autobus già presente con questa targa!';
    ELSE
        INSERT INTO autobus (Targa, Capienza, CostoForfettario)
        VALUES (p_Targa, p_Capienza, p_CostoForfettario);
    END IF;

    COMMIT;
END;
```

Livello di isolamento: READ COMMITTED

Poiché la verifica di unicità si basa su una chiave primaria (Targa), il rischio di anomalie è molto basso. Il database stesso garantirebbe l'unicità.

Funzionalità operativa: Procedura di inserimento di una autobus

11. Inserisci albergo

```
create
    definer = root@localhost procedure inserisciAlbergo(IN p_Nome varchar(200),
IN p_Capienza int, IN p_CostoPN float,
                                                        IN p_Referente
varchar(100), IN p_Via varchar(60),
                                                        IN p_Numero varchar(10),
IN p_Cap varchar(20),
                                                        IN p_Email varchar(100),
IN p_Fax varchar(20),
                                                        IN p_NomeLocalita
varchar(50), IN p_Stato varchar(60))
BEGIN
    -- Dichiaro il gestore di eccezioni per la transazione
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
        BEGIN
            ROLLBACK;
            RESIGNAL;
        END;

    -- Imposto il livello di isolamento della transazione
    SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;
    START TRANSACTION;
```

```
    -- Controllo se la località esiste
    IF NOT EXISTS (SELECT 1 FROM localita WHERE Nome = p_NomeLocalita AND Stato =
p_Stato) THEN
        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Località non esistente! Inserire prima la
località.';
    ELSEIF EXISTS (SELECT 1 FROM albergo WHERE Nome = p_Nome AND Via = p_Via AND
Numero = p_Numero AND Cap = p_Cap) THEN
        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Albergo già esistente con lo stesso nome e
indirizzo!';
    ELSE
        -- Inserisco l'albergo
        INSERT INTO albergo (
            Nome, Capienza, CostoPN, Referente, Via, Numero, Cap, Email, Fax,
NomeLocalita, Stato
        ) VALUES (
                        p_Nome, p_Capienza, p_CostoPN, p_Referente, p_Via, p_Numero,
p_Cap, p_Email, p_Fax, p_NomeLocalita, p_Stato
                    );
    END IF;

    COMMIT;
END;
```

Livello di isolamento: REPETABLE READ

Il livello scelto garantisce che la condizione da verificare rimanda vera fino al momento dell'inserimento.

Funzionalità operativa: Procedura di inserimento di una Albergo

12. Login

```
create
    definer = root@localhost procedure login(IN p_email varchar(100), IN
p_password varchar(100),
                                             OUT p_ruolo enum ('AM', 'AV'))
BEGIN
    DECLARE v_count INT DEFAULT 0;

    -- Dichiaro il gestore di eccezioni per la transazione
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
        BEGIN
            ROLLBACK;
            RESIGNAL;
        END;

    -- Imposto il livello di isolamento della transazione
    SET TRANSACTION ISOLATION LEVEL READ COMMITTED ;
    START TRANSACTION;

    -- Controllo se l'utente esiste e recupero il ruolo
    SELECT COUNT(*), Ruolo INTO v_count, p_ruolo
    FROM user
    WHERE Email = p_email AND Password = p_password;

    IF v_count = 0 THEN
        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Email o password errati!';
    ELSE
```

```
        COMMIT;
    END IF;
end;
```

Livello di isolamento; READ COMMITTED

Essendo che si fa solamente una select su dei dati che presumibilmente cambiano raramente è stato scelto di usare READ COMMITTED come livello di isolamento.

Funzionalità operativa: Procedura di login che restituisce il ruolo dell'utente inserito, che verrà gestito nel client per aprire la corretta connessione del db.

Le procedure che verranno mostrate in seguito sono procedure di sola lettura che sono utilizzate dal client per la visualizzazione dei dati e sono state create per una maggiore fruibilità del sistema, queste utilizzano tutte il livello di isolamento READ COMMITTED, che rappresenta un buon compromesso tra consistenza e performance per operazioni di sola lettura.

Il livello scelto:

- Previene letture sporche
- Consente maggiore velocità rispetto a livelli più restrittivi
- È sufficiente per procedure che non richiedono inserimenti ma solo selezione dei dati

13. Stato del viaggio

```
create
    definer = root@localhost procedure statoVIaggio(IN codiceViaggio int, OUT
stato varchar(20))
BEGIN
    -- Dichiaro il gestore di eccezioni per la transazione
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
        BEGIN
            ROLLBACK;
            RESIGNAL;
        END;

    -- Imposto il livello di isolamento della transazione
    SET TRANSACTION ISOLATION LEVEL READ COMMITTED ;
    START TRANSACTION;

    SELECT Stato
    INTO stato
    FROM viaggio
    WHERE IDViaggio = codiceViaggio;

    IF stato IS NULL THEN
        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Viaggio non trovato';
    END IF;
    COMMIT;
END;
```

Funzionalità operativa: Utilizzata nel client come verifica preliminare dello stato di un viaggio, se lo stato non è almeno "INCORSO" non permetto di entrare nel sotto menu per effettuare le operazioni: di associazione di un autobus, di inserimento di un pernottamento e di generazione del report.
Poteva essere anche evitata ma creare una barriera preliminare fa risparmiare sia tempo alla segreteria che aggiunge un livello di sicurezza maggiore dei dati.

14. Verifica esistenza viaggio

```
create
    definer = root@localhost procedure verificaEsistenzaViaggio(IN codiceViaggio
int, OUT esiste tinyint(1), OUT Stato varchar(20))
BEGIN
    DECLARE var_count INT;
    -- Dichiaro il gestore di eccezioni per la transazione
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
        BEGIN
            ROLLBACK;
            RESIGNAL;
        END;

    -- Imposto il livello di isolamento della transazione
    SET TRANSACTION ISOLATION LEVEL READ COMMITTED ;
    START TRANSACTION;

    SELECT COUNT(*), viaggio.Stato INTO var_count, Stato
    FROM viaggio
    WHERE IDViaggio = codiceViaggio;

    SET esiste = (var_count > 0);
    IF NOT esiste THEN
        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Viaggio non trovato';
    END IF;
    COMMIT;
END;
```

Funzionalità operativa: Anche questa procedura è stata creata per effettuare una verifica preliminare dell'esistenza del viaggio

15. Visualizza alberghi per località

```
create
    definer = root@localhost procedure visualizzaAlberghiPerLocalita(IN
p_idViaggio int, IN p_nomeLocalita varchar(50),
                                                                    IN p_stato
varchar(60))
BEGIN
    -- Dichiaro il gestore di eccezioni per la transazione
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
        BEGIN
            ROLLBACK;
            RESIGNAL;
        END;

    -- Imposto il livello di isolamento della transazione
    SET TRANSACTION ISOLATION LEVEL READ COMMITTED ;
    START TRANSACTION;
```

```
    SELECT a.CodiceAlbergo, a.Nome, a.Capienza, a.CostoPN, a.Referente, a.Via,
a.Numero, a.Cap, a.Email, a.Fax, t.NomeLocalita, t.Stato
    FROM albergo a
    JOIN tappa t ON a.NomeLocalita = t.NomeLocalita AND a.Stato = t.Stato
    JOIN viaggio v ON t.IDItinerario = v.IDItinerario
    WHERE v.IDViaggio = p_idViaggio
      AND a.NomeLocalita = p_nomeLocalita
      AND a.Stato = p_stato and Capienza > (SELECT
IFNULL(SUM(p.NumeroPartecipanti),0)
                            FROM prenotazioni p
                            WHERE p.CodiceViaggio = v.IDViaggio);

END;
```

Funzionalità operativa: Vengono selezionati solo gli alberghi della tappa e che abbiano una capienza maggiore del numero di partecipanti al viaggio

16. Visualizza autobus disponibili

```
create
    definer = root@localhost procedure visualizzaAutobusDisponibili(IN
p_idViaggio int)
BEGIN
    DECLARE v_dataPartenza DATE;
    DECLARE v_dataRientro DATE;

    -- Dichiaro il gestore di eccezioni per la transazione
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
        BEGIN
            ROLLBACK;
            RESIGNAL;
        END;

    -- Imposto il livello di isolamento della transazione
    SET TRANSACTION ISOLATION LEVEL READ COMMITTED ;
    START TRANSACTION;

    -- Verifico che il viaggio esista
    IF NOT EXISTS (SELECT 1 FROM viaggio WHERE IDViaggio = p_idViaggio) THEN
        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Il viaggio richiesto non esiste.';
    END IF;

    -- Prendi le date del viaggio richiesto
    SELECT DataPartenza, DataRientro
    INTO v_dataPartenza, v_dataRientro
    FROM viaggio
    WHERE IDViaggio = p_idViaggio;

    -- Seleziona autobus non impegnati in altri viaggi nelle stesse date
    SELECT a.Targa, a.CostoForfettario, a.Capienza
    FROM autobus a
    WHERE a.Targa NOT IN (
        SELECT av.Targa
        FROM autobusviaggio av
                 JOIN viaggio v ON av.CodiceViaggio = v.IDViaggio
        WHERE (v.DataPartenza <= v_dataRientro AND v.DataRientro >=
v_dataPartenza)
    );
    COMMIT;
end;
```

Funzionalità operativa: vengono visualizzati tutti gli autobus disponibili per le date del viaggio

17. Visualizza itinerari possibili

```
create
    definer = root@localhost procedure visualizzaItinerariDisponibili()
BEGIN
    -- Dichiaro il gestore di eccezioni per la transazione
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
        BEGIN
            ROLLBACK;
            RESIGNAL;
        END;

    -- Imposto il livello di isolamento della transazione
    SET TRANSACTION ISOLATION LEVEL READ COMMITTED ;
    START TRANSACTION;

   SELECT itinerario.IDItinerario, itinerario.NomeItinerario, itinerario.Costo,
t.NomeLocalita,
          t.Stato, t.Giorni,t.Ordine
    From itinerario join agenziadiviaggi.tappa t on itinerario.IDItinerario =
t.IDItinerario
    ORDER BY itinerario.IDItinerario, t.Ordine;
    COMMIT;
END;
```

Funzionalità operativa: Vengono visualizzati gli itinerari con le tappe associate

18. Visualizzano itinerario con tappe

```
create
    definer = root@localhost procedure visualizzaItinerarioConTappe(IN idViaggio
int)
BEGIN
    DECLARE var_count INT;
    DECLARE var_idItinerario INT;
    -- Controllo se il viaggio esiste

    -- Dichiaro il gestore di eccezioni per la transazione
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
        BEGIN
            ROLLBACK;
            RESIGNAL;
        END;

    -- Imposto il livello di isolamento della transazione
    SET TRANSACTION ISOLATION LEVEL READ COMMITTED ;
    START TRANSACTION;

    SELECT COUNT(*) INTO var_count
    FROM agenziadiviaggi.viaggio
    WHERE IDViaggio = idViaggio;
    IF var_count = 0 THEN
        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Viaggio non trovato';
    END IF;

   SELECT  i.IDItinerario, Durata, Costo, NomeItinerario, NomeLocalita, t.Stato,
Ordine, Giorni
       from viaggio join agenziadiviaggi.itinerario i on i.IDItinerario =
viaggio.IDItinerario join agenziadiviaggi.tappa t on i.IDItinerario =
```

```
t.IDItinerario
         where viaggio.IDViaggio = idViaggio;
    COMMIT;
end;
```

Funzionalità operativa: Vengono visualizzati gli itinerari con le tappe associate

19. Visualizza località disponibili

```
create
    definer = root@localhost procedure visualizzaLocalitaDisponibili()
BEGIN

    -- Dichiaro il gestore di eccezioni per la transazione
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
        BEGIN
            ROLLBACK;
            RESIGNAL;
        END;

    -- Imposto il livello di isolamento della transazione
    SET TRANSACTION ISOLATION LEVEL READ COMMITTED ;
    START TRANSACTION;

    SELECT Nome, Stato, Regione, Provincia
    FROM agenziadiviaggi.localita
    ORDER BY Nome;
    COMMIT;
END;
```

Funzionalità operativa: vengono visualizzate le località disponibili in fase di creazione di un itinerario

20. Visualizza viaggi disponibili

```
create
    definer = root@localhost procedure visualizzaViaggiDisponibili()
BEGIN

    -- Dichiaro il gestore di eccezioni per la transazione
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
        BEGIN
            ROLLBACK;
            RESIGNAL;
        END;

    -- Imposto il livello di isolamento della transazione
    SET TRANSACTION ISOLATION LEVEL READ COMMITTED ;
    START TRANSACTION;

    SELECT
        v.IDViaggio,
        i.NomeItinerario,
        v.DataPartenza,
        v.Stato,
        t.Ordine,
        t.NomeLocalita,
        t.Stato AS StatoLocalita,
        t.Giorni
    FROM viaggio v
             JOIN itinerario i ON v.IDItinerario = i.IDItinerario
```

```
            JOIN tappa t ON t.IDItinerario = i.IDItinerario
    ORDER BY v.IDViaggio, t.Ordine;
    COMMIT;
END;
```

Funzionalità operativa: vengono visualizzati tutti i viaggi, che poi nel cliente verranno filtrati in base a chi chiama la procedura

```
grant execute on procedure agenziadiviaggi.creaViaggio to Segreteria;
grant execute on procedure agenziadiviaggi.inserisciLocalita to Segreteria;
grant execute on procedure agenziadiviaggi.inserisciAutobus to Segreteria;
grant execute on procedure agenziadiviaggi.inserisciAlbergo to Segreteria;
grant execute on procedure agenziadiviaggi.creaItinerario to Segreteria;
grant execute on procedure agenziadiviaggi.aggiungiTappaItinerario to Segreteria;
grant execute on procedure agenziadiviaggi.associaPernottamento to Segreteria;
grant execute on procedure agenziadiviaggi.login to Segreteria;
grant execute on procedure agenziadiviaggi.associaAutobusViaggio to Segreteria;
grant execute on procedure agenziadiviaggi.generaReportViaggio to Segreteria;
grant execute on procedure agenziadiviaggi.visualizzaViaggiDisponibili to Segreteria;
grant execute on procedure agenziadiviaggi.visualizzaItinerariDisponibili to Segreteria;
grant execute on procedure agenziadiviaggi.visualizzaLocalitaDisponibili to Segreteria;
grant execute on procedure agenziadiviaggi.visualizzaAutobusDisponibili to Segreteria;
grant execute on procedure agenziadiviaggi.visualizzaItinerarioConTappe to Segreteria;
grant execute on procedure agenziadiviaggi.visualizzaAlberghiPerLocalita to Segreteria;
grant execute on procedure agenziadiviaggi.cancellazionePrenotazione to Agente;
grant execute on procedure agenziadiviaggi.inserisciPrenotazione to Agente;
grant execute on procedure agenziadiviaggi.login to Agente;
grant execute on procedure agenziadiviaggi.visualizzaViaggiDisponibili to Agente;
```