

Hotel Statistics

Università degli studi di Roma Tor Vergata



Gabriele Biscetti

Simone Caruso

07 02 2023

Sommario

Il presente documento rappresenta un progetto puramente accademico, senza pretese di completezza nella descrizione del dominio di interesse. Come metodologia di progettazione, verrà utilizzata quella proposta dal testo [\[1\]](#) alla quale verranno aggiunte le varianti suggerite dalla professoressa Loredana Vigliano. Tale metodologia è articolata in tre fasi principali da effettuare in cascata: raccolta e analisi dei requisiti, progettazione concettuale e progettazione fisica.

Indice

1	Raccolta e analisi dei requisiti	2
1.1	Requisiti espressi in linguaggio naturale	2
1.2	Strutturazione dei requisiti	4
1.3	Glossario dei termini	7
1.4	Requisiti software	7
2	Progettazione concettuale	8
2.1	Diagramma E-R (Entity Relationship)	8
2.2	Dizionario dei dati	9
2.3	Regole aziendali	9
3	Progettazione logica	13
3.1	Le principali operazioni	13
3.2	Ristrutturazione del diagramma E-R	15
3.3	Diagramma E-R normalizzato	15
3.4	Traduzione verso il modello relazione	15
3.5	Normalizzazione	20
3.6	SQL DDL (Data Definition Language)	21
3.7	SQL DML (Data Manipulation Language)	24
3.8	SQL DQL (Data Query Language)	25
3.9	Viste	32
3.10	Stored procedure	33
3.11	Triggers	36
3.12	Cursori	39
3.13	Funzioni analitiche	41
3.14	Gestione utenti e privilegi	43
3.15	Algebra relazionale	44
3.16	Calcolo relazionale	45
4	Progettazione fisica	46
4.1	Storage Engines	47
4.2	Indici e performance	48
4.3	Deploy e scripting	58
5	Un diverso modello dei dati	60
5.1	MongoDB	60
5.2	Query con MongoDB	61

Raccolta e analisi dei requisiti

L'attività di raccolta dei requisiti, che precede la fase di progettazione, fornisce una specifica informale ma completa di tutte le proprietà e le funzionalità del sistema software o base di dati che si vuole progettare. In sintesi, si descrive il dominio di interesse.

Generalmente i requisiti che compongono la specifica vengono acquisiti dall'interazione con gli utenti, dalla documentazione esistente e/o eventuali realizzazioni preesistenti. In questo caso il fatto che non ci siano dei possibili utenti rende qualsiasi attività della raccolta legata all'interazione con gli utenti impossibile.

Quindi assumeremo per semplicità che il testo seguente sia una specifica completa e informale ottenuta da una raccolta dei requisiti preliminare per la quale non documenteremo le attività di intervista con gli utenti e procedendo dunque all'analisi dei requisiti.

1.1 Requisiti espressi in linguaggio naturale

Si vuole realizzare una base di dati per un software statistico che opera su una catena alberghiera. Tale catena alberghiera è composta da diversi hotel. Per ognuno di essi siamo interessati a tenere traccia di tutti i clienti degli hotel, del loro soggiorno e delle informazioni relative all'utilizzo dei servizi offerti da ciascun hotel insieme ai corrispondenti costi di gestione e possibili guadagni.

I clienti, ognuno con un codice identificativo, possono essere o delle società delle quali interessa sapere il nome e la partita IVA, o delle persone per le quali si memorizza il codice fiscale, nome, cognome ed età. Per ogni cliente si mantiene anche il numero di telefono e un indirizzo mail (opzionale). Le persone di cui si memorizzano i dati possono aver prenotato una stanza, aver alloggiato in una camera senza averla prenotata a loro nome, o aver partecipato a un convegno.

Ogni hotel è caratterizzato oltre che dal nome, dalla città e dalla via in cui si trova (attributi identificativi), anche dal numero di stelle e da un numero di telefono. In tutti gli hotel le stanze sono numerate univocamente per ognuna delle quali si memorizza il numero di posti e il prezzo associato. Le stanze di un hotel si suddividono in camere per il soggiorno per le quali il numero di posti rappresenta il numero di posti letto, e in sale convegno, caratterizzate da un nome e dove vengono presentati eventi. Naturalmente per le stanze che sono delle sale convegno il numero di posti riportato rappresenta il numero di persone che è in grado di ospitare. Il prezzo delle camere rappresenta il prezzo di pernottamento di quella camera per una singola persona, mentre il prezzo di una sala convegno rappresenta il suo prezzo di prenotazione. Ogni convegno viene presentato in una sala convegno di un hotel appositamente prenotata

da un cliente. La durata di un convegno è di un giorno. Inoltre per ogni convegno presentato in un hotel si vuole sapere il nome del convegno, e la data di presentazione.

Ovviamente qualunque cliente può prenotare una stanza, ma una camera può ospitare solo persone e non una società. Al momento della prenotazione un cliente prenota una determinata stanza specificando la data di arrivo, la data di partenza e il numero di ospiti, cioè il numero totale di persone che la stanza prenotata dovrà ospitare. Naturalmente per un qualsiasi periodo di permanenza una stanza non può avere di al più di una prenotazione attiva per essa.

Si vogliono rappresentare tutte le prenotazioni passate e presenti. Ogni prenotazione è a nome di un solo cliente ed è relativa a una sola stanza. (Un cliente deve fare una prenotazione diversa per ogni stanza che vuole prenotare a suo nome.) Per ciascuna prenotazione viene generato un codice identificativo, e si vuole memorizzare il periodo di permanenza (la data di arrivo e di partenza) e il numero di ospiti.

Per ogni prenotazione di una stanza a nome di una persona, risulta che quella persona occupa o ha occupato tale stanza. Inoltre per ogni prenotazione di una stanza si vogliono conoscere anche le informazioni relative alle persone che hanno occupato o occupano tale stanza senza averla prenotata a loro nome.

A ogni singola prenotazione corrisponde un importo che il cliente deve pagare. Per ogni prenotazione di una sala convegno l'importo da pagare è semplicemente il costo di prenotazione della sala. Invece per ogni prenotazione di una camera l'importo viene calcolato moltiplicando il costo di pernottamento della camera prenotata per il numero di notti soggiornate e il numero di posti prenotati. Quando il pagamento viene saldato, l'hotel rilascia una fattura di prenotazione, ognuna identificata da un codice e sulla quale si riporta la data del pagamento e il totale pagato. Inoltre interessa conoscere anche le modalità di pagamento (contanti, carta di credito, bonifico).

Ciascun hotel offre vari servizi e per ogni hotel si vogliono sapere tutti i servizi offerti. Naturalmente ogni servizio di un hotel può essere usato da una persona solamente durante il periodo in cui questa viene ospitata in quell'hotel. Inoltre si è interessati a tenere traccia di tutti gli utilizzi che una persona fa dei servizi dell'hotel in cui è ospitata, e di sapere quando tale servizio viene utilizzato.

In ciascuna stanza di un hotel possono verificarsi guasti di varia natura. Ognuno di questi guasti richiede una specifica attività di manutenzione per essere risolto. Tutte le attività di manutenzione vengono fornite da società terze delle quali si vuole memorizzare il nome, la P.IVA, il numero di telefono e un indirizzo e-mail. Ogni attività di manutenzione eseguita su una stanza è caratterizzata dalla categoria (Idraulica, Elettrica, Pulizia, Edile, Arredo) del guasto che viene riparato, da una breve descrizione dell'attività svolta, dalla data di manutenzione e dal prezzo richiesto dalla società all'hotel per l'attività fornita.

Ad ogni servizio afferisce un insieme (non vuoto) di dipendenti dei quali si rappresentano il codice fiscale, il nome, il cognome, lo stipendio, la data di nascita e la data di assunzione. Ciascun dipendente può afferire a un solo servizio e per un solo hotel.

Per ciascun servizio dell'hotel vengono rappresentati tutti i prodotti necessari a rifornire quel servizio. Ogni prodotto rifornisce un solo servizio ed è caratterizzato dal nome, dalla marca e dal codice (identificativo). Naturalmente vengono acquistati solamente i prodotti necessari per rifornire i servizi. Al fine di rintracciare ogni spesa, si vuole memorizzare per ogni prodotto la fattura relativa all'acquisto con cui il prodotto è stato acquistato. Tutte le fatture sono emesse dopo l'acquisto di una fornitura di prodotti che può essere composta da prodotti per servizi diversi e in quantità variabili. Per ogni fattura si vuole sapere il codice, la data del pagamento, l'importo totale pagato e il numero totale di prodotti acquistati. Inoltre si vuole tenere traccia della

variazione del prezzo dei prodotti acquistati. Infatti al fine di gestire la contabilità degli hotel si vuole sapere con frequenza mensile i possibili rincari dovuti all'acquisto di forniture di ogni servizio.

1.2 Strutturazione dei requisiti

Un primo livello di analisi dei requisiti è offerto dalla strutturazione; si decompone il testo precedente in gruppi di frasi relative agli stessi concetti. In questo modo è possibile evidenziare proprietà significative utili alla rappresentazione concettuale dei dati.

Frasi di carattere generale

Si vuole realizzare una base di dati per un software che raccoglie statistiche su una catena alberghiera. Tale catena alberghiera è composta da diversi hotel. Per ognuno di essi siamo interessati a tenere traccia di tutti i clienti degli hotel, del loro soggiorno e delle informazioni relative all'utilizzo dei servizi offerti da ciascun hotel insieme ai corrispondenti costi di gestione e possibili guadagni.

Frasi relative a Hotel

Ogni hotel è caratterizzato oltre che dal nome, dalla città e dalla via in cui si trova (attributi identificativi), anche dal numero di stelle e da un numero di telefono. In tutti gli hotel le stanze sono numerate univocamente per ognuna delle quali si memorizza il numero di posti e il prezzo associato. Le stanze di un hotel si suddividono in camere per il soggiorno per le quali il numero di posti rappresenta il numero di posti letto, e in sale convegno, caratterizzate da un nome e dove vengono presentati eventi. Naturalmente per le stanze che sono delle sale convegno il numero di posti riportato rappresenta il numero di persone che è in grado di ospitare. Il prezzo delle camere rappresenta il prezzo di pernottamento di quella camera per una singola persona, mentre il prezzo di una sala convegno rappresenta il suo prezzo di prenotazione. Ogni convegno viene presentato in una sala convegno di un hotel appositamente prenotata da un cliente. La durata di un convegno è di un giorno. Inoltre per ogni convegno presentato in un hotel si vuole sapere il nome del convegno, e la data di presentazione. Ovviamente qualunque cliente può prenotare una stanza, ma una camera può ospitare solo persone e non una società. Al momento della prenotazione un cliente prenota una determinata stanza specificando la data di arrivo, la data di partenza e il numero di ospiti, cioè il numero totale di persone che la stanza prenotata dovrà ospitare. Naturalmente per un qualsiasi periodo di permanenza una stanza non può avere di al più di una prenotazione attiva per essa.

Frase relative ai clienti

I clienti, ognuno con un codice identificativo, possono essere o delle società delle quali interessa sapere il nome e la partita IVA, o delle persone per le quali si memorizza il codice fiscale, nome, cognome ed età. Per ogni cliente si mantiene anche il numero di telefono e un indirizzo mail (opzionale). Le persone di cui si memorizzano i dati possono aver prenotato una stanza, aver alloggiato in una camera senza averla prenotata a loro nome, o aver partecipato a un convegno.

Frase relative alle prenotazioni

Si vogliono rappresentare tutte le prenotazioni passate e presenti. Ogni prenotazione è a nome di un solo cliente ed è relativa a una sola stanza. (Un cliente deve fare una prenotazione diversa per ogni stanza che vuole prenotare a suo nome.) Per ciascuna prenotazione viene generato un codice identificativo, e si vuole memorizzare il periodo di permanenza (la data di arrivo e di partenza) e il numero di ospiti. Per ogni prenotazione di una stanza a nome di una persona, risulta che quella persona occupa o ha occupato tale stanza. Inoltre per ogni prenotazione di una stanza si vogliono conoscere anche le informazioni relative alle persone che hanno occupato o occupano tale stanza senza averla prenotata a loro nome. A ogni singola prenotazione corrisponde un importo che il cliente deve pagare. Per ogni prenotazione di una sala convegno l'importo da pagare è semplicemente il costo di prenotazione della sala. Invece per ogni prenotazione di una camera l'importo viene calcolato moltiplicando il costo di pernottamento della camera prenotata per il numero di notti soggiornate e il numero di posti prenotati. Quando il pagamento viene saldato, l'hotel rilascia una fattura di prenotazione, ognuna identificata da un codice e sulla quale si riporta la data del pagamento e il totale pagato. Inoltre interessa conoscere anche le modalità di pagamento (contanti, carta di credito, bonifico).

Frase relative ai servizi

Ciascun hotel offre vari servizi e per ogni hotel si vogliono sapere tutti i servizi offerti. Naturalmente ogni servizio di un hotel può essere usato da una persona solamente durante il periodo in cui questa viene ospitata in quell'hotel. Inoltre si è interessati a tenere traccia di tutti gli utilizzi che una persona fa dei servizi dell'hotel in cui è ospitata, e di sapere quando tale servizio viene utilizzato.

Frazi relative alle manutenzioni

In ciascuna stanza di un hotel possono verificarsi guasti di varia natura. Ognuno di questi guasti richiede una specifica attività di manutenzione per essere risolto. Tutte le attività di manutenzione vengono fornite da società terze delle quali si vuole memorizzare il nome, la P.IVA, il numero di telefono e un indirizzo e-mail. Ogni attività di manutenzione eseguita su una stanza è caratterizzata dalla categoria (Idraulica, Elettrica, Pulizia, Edile, Arredo) del guasto che viene riparato, da una breve descrizione dell'attività svolta, dalla data di manutenzione e dal prezzo richiesto dalla società all'hotel per l'attività fornita.

Frazi relative ai dipendenti

Ad ogni servizio afferisce un insieme (non vuoto) di dipendenti dei quali si rappresentano il codice fiscale, il nome, il cognome, lo stipendio, la data di nascita e la data di assunzione. Ciascun dipendente può afferire a un solo servizio e per un solo hotel.

Frazi relative ai prodotti

Per ciascun servizio dell'hotel vengono rappresentati tutti i prodotti necessari a rifornire quel servizio. Ogni prodotto rifornisce un solo servizio ed è caratterizzato dal nome, dalla marca e dal codice (identificativo). Naturalmente vengono acquistati solamente i prodotti necessari per rifornire i servizi. Al fine di rintracciare ogni spesa, si vuole memorizzare per ogni prodotto la fattura relativa all'acquisto con cui il prodotto è stato acquistato. Tutte le fatture sono emesse dopo l'acquisto di una fornitura di prodotti che può essere composta da prodotti per servizi diversi e in quantità variabili. Per ogni fattura si vuole sapere il codice, la data del pagamento, l'importo totale pagato e il numero totale di prodotti acquistati. Inoltre si vuole tenere traccia della variazione del prezzo dei prodotti acquistati. Infatti al fine di gestire la contabilità degli hotel si vuole sapere con frequenza mensile i possibili rincari dovuti all'acquisto di forniture di ogni servizio.

1.3 Glossario dei termini

Per la comprensione e la precisazione dei termini utilizzati nella descrizione del dominio si utilizza un *glossario* che ad ogni termine associa una breve descrizione.

Termine	Descrizione
Codice fiscale	Codice che identifica in modo univoco le persone fisiche
Convegno	Riunione per la discussione di argomenti di comune interesse
Cliente	Persona maggiorenne o società
Dipendente	Persona maggiorenne che lavora presso l'hotel
Fattura	Ricevuta di pagamento
Guasto	Malfunzionamento che rende la stanza inservibile
Hotel	Struttura alberghiera
Prenotazione	I clienti riservano stanze o tavoli
Software	Insieme di programmi con documentazione associata
Manutenzione	Attività per la risoluzione o prevenzione di guasti
Partita IVA	Codice che identifica univocamente una società

Tabella 1.1: Il glossario dei termini per il progetto *Hotel Statistics*

1.4 Requisiti software

Seguendo il modello relazionale sono disponibili numerosi [RDBMS](#) (Relational Database Management System) sul mercato. Per via della semplicità del progetto proposto si è scelto l'utilizzo di [MariaDB](#). Questa scelta è dettata soprattutto dalla completa natura open source di questo software. Non segue più la stessa strada il vecchio [MySQL](#), ormai divenuto un prodotto in parte [proprietario](#) dopo la sua acquisizione da parte di Oracle. Interessante notare come MariaDB offra sempre più funzionalità ed una maggiore adesione allo standard SQL. Ulteriori informazioni sul confronto possono essere trovate in questa [pagina](#).

Progettazione concettuale

2.1 Diagramma E-R (Entity Relationship)

Il prodotto di questa fase è lo *schema concettuale* e fa riferimento a un *modello concettuale* dei dati indipendente dai criteri di rappresentazione utilizzati nel DBMS. Il modello concettuale utilizzato è il modello E-R (Entità-Relazione) come formalizzato da Peter Chen.

Grazie alla precedente strutturazione dei requisiti, è possibile descrivere ogni concetto con il suo relativo schema E-R. Lo schema E-R finale sarà ottenuto mediante passi di integrazione con i precedenti (come descritto da una strategia *inside out*). Si riporta il diagramma E-R concettuale finale in Figura 2.1. Non utilizzando una notazione standard riconosciuta come [UML](#), di seguito è riportata una legenda nella Tabella 2.1 utile alla lettura del diagramma E-R.




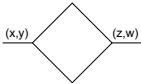

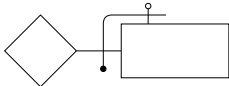

Costrutto	Rappresentazione
Entità	
Relazione	
Attributo	
Cardinalità di relazione	
Identificatore	
Identificatore esterno	
Generalizzazione	

Tabella 2.1: Legenda relativa al diagramma E-R

2.2 Dizionario dei dati

Uno schema E-R deve essere corredato da una documentazione di supporto per facilitarne la comprensione e descrivere ulteriori proprietà dei dati. Un *dizionario dei dati* è composto da due tabelle: la prima descrive le entità dello schema mentre la seconda le relazioni. Queste sono riportate nella Tabella 2.2 e nella Tabella 2.3

2.3 Regole aziendali

Uno schema concettuale, in particolare uno schema E-R, è completo nell'esprimere il contenuto informativo ma non i vincoli sui dati. È necessaria una documentazione a supporto per esprimere ulteriori vincoli sui dati. Questo può essere realizzato mediante le regole aziendali espresse in un linguaggio informale. Una regola aziendale può essere una regola di vincolo o derivazione. La prima, esprime quel vincolo sui dati non esprimibile da uno schema E-R, mentre la seconda ci dice come un concetto possa essere ottenuto da altri concetti. Si riporta in seguito questo tipo di regole per il progetto *HotelStatistics*.

Regole di vincolo

1. Una stanza non deve avere più prenotazioni con periodo di permanenza sovrapponibili.
2. Il numero di posti della prenotazione di una stanza deve essere minore o uguale al numero di posti della stanza prenotata.
3. Il numero di ospiti di una stanza prenotata deve essere uguale al numero di posti per la relativa prenotazione della stanza.
4. Una persona ospite di un hotel deve usare un servizio dell'hotel in cui è ospitato e in una data compresa nel periodo di permanenza nell'hotel.

Regole di derivazione

1. Il costo di prenotazione di una camera si ottiene moltiplicando il numero di ospiti della prenotazione per il prezzo di pernottamento della camera prenotata e il numero di notti soggiornate

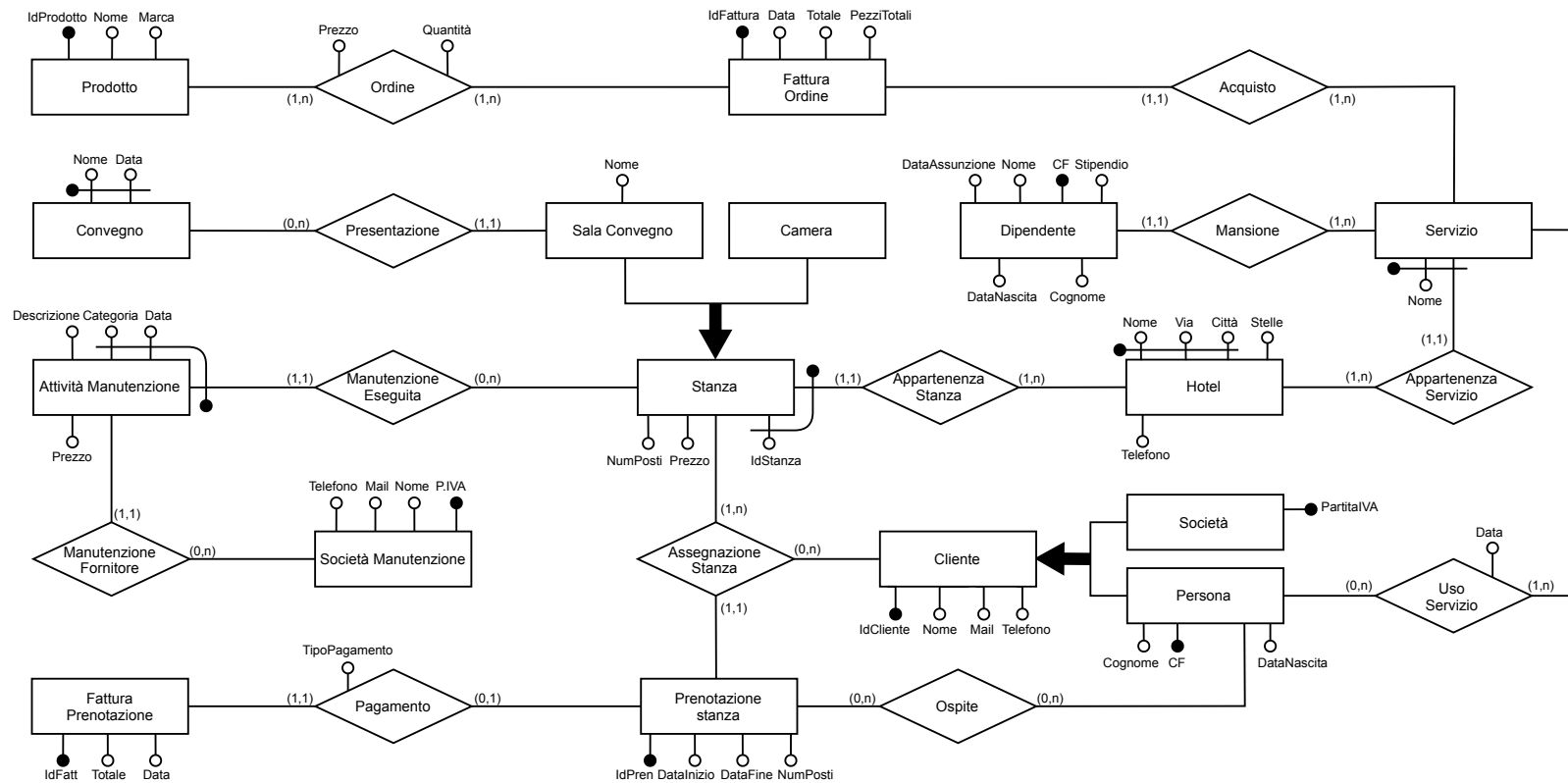


Figura 2.1: Il diagramma E-R dell'intero progetto *Hotel Statistics*

Entità	Descrizione	Attributi	Identificatore
Attività Manutenzione	Risoluzione guasti	Descrizione, Prezzo	Categoria, Data, Stanza
Camera	Camera di un hotel	NumPosti, Prezzo,	IdStanza,Hotel
Cliente	Persona o Società in grado di prenotare	Codice	
Convegno	Evento o riunione in un preciso giorno	Nome, Data, Edizione	Nome, Data, Edizione
Dipendente	Lavoratore stipendiato di un hotel	Nome, Cognome, Stipendio, DataNascita, DataAssunzione	CF
Fattura Ordine	Ricevuta pagamento di acquisto prodotti	Data, Totale, PezziTotali	IdFattura
Fattura Prenotazione	Ricevuta di pagamento del soggiorno	Totale, Data	IdFatt
Hotel	Struttura alberghiera	Cap, Stelle, Telefono	Nome, Via, Città
Servizio	Servizi per soddisfare i clienti	Nome	Nome, Hotel
Società	Azienda in grado di prenotare	Telefono, Mail, Nome	P.IVA
Società Manutenzione	Società addetta allla manutenzione	Telefono, Mail, Nome	P.IVA
Stanza	Stanza di un hotel	Categoria, Nome, Prezzo, NumPosti	IdStanza, Hotel
Sala Convegno	Sala messa a disposizione da un hotel	Nome	
Persona	Cliente di un hotel	Cognome, Nome, Mail, Telefono, Età	CF
Prenotazione Stanza	Prenotazione del soggiorno	DataInizio, DataFine, NumPosti	IdPren
Prodotto	Prodotto necessario ad un hotel	Marca, Nome	IdProdotto

Tabella 2.2: Il dizionario dei dati per le entità dello schema concettuale

Relazione	Descrizione	Entità coinvolte	Attributi
Appartenenza Servizio	Associa un servizio ad un hotel	Hotel,Servizio	
Appartenenza Stanza	Associa una stanza ad un hotel	Hotel, Stanza	
Assegnazione Stanza	Associa cliente, stanza e prenotazione	Stanza, PrenotazioneStanza, Cliente	
Ordine	Associa una Fattura di un ordine ad un prodotto ordinato	Fattura Ordine, Prodotto	Prezzo, Quantità
Ospite	Associa una persona ad una prenotazione attiva	Persona, PrenotazioneStanza	
Mansione	Associa un Dipendente ad un Servizio	Dipendente, Servizio	
Manutenzione Eseguita	Associa una manutenzione ad una stanza	AttivitàManutenzione, Stanza	
Manutenzione Fornitore	Associa una manutenzione ad una società	AttivitàManutenzione, Società	
Pagamento	Associa una prenotazione di una stanza la sua fattura	FatturaPrenotazione, PrenotazioneStanza	
Presentazione	Associa un convegno ad una sala convegni	Stanza, Convegno	
Uso Servizio	Associa una persona ad un servizio utilizzato	Persona, Servizio	Data

Tabella 2.3: Il dizionario dei dati per le relazioni dello schema concettuale

Progettazione logica

Questa fase consiste nella traduzione dello schema E-R normalizzato, in termini del modello di rappresentazione dei dati adottato dal DBMS scelto. Il prodotto di questa fase viene chiamato *Schema logico* della base di dati e fa riferimento a un modello logico dei dati. Il modello logico dei dati utilizzato è il modello *relazionale*. Questa fase inoltre prevede l'utilizzo della *normalizzazione*, tecnica per il controllo qualità dello schema logico prodotto.

3.1 Le principali operazioni

Per poter tradurre in modo efficiente ed efficace il diagramma E-R, serve una previsione del possibile carico applicativo (dati e operazioni). Sono riportate in seguito le principali operazioni. Si evidenzia come non sia prevedibile il carico applicativo per via della natura didattica del progetto; non si ha a disposizione una previsione frutto di una analisi dei requisiti o studio di una precedente implementazione.

1. Stampare nome, cognome e numero di telefono di tutte le persone che sono o sono stati ospiti in una stanza d'hotel.
2. Stampa le informazioni di tutti i convegni, il numero di partecipanti, il nome della sala convegno e dell'hotel dove sono stati presentati.
3. Data una stanza di un hotel e un periodo di permanenza trovare se esiste una prenotazione con periodo di permanenza sovrapponibile per tale stanza.
4. Dato un hotel e un periodo di permanenza trovare tutte le stanze non prenotate per quel periodo.
5. Dato un anno, stampare per ogni mese la somma degli importi pagati dai clienti in quel mese.
6. Dato un anno, stampare per ogni mese la somma degli importi pagati dai clienti in quel mese, il numero di pagamenti totali e il numero di pagamenti per ogni tipo di pagamento.
7. Dato un anno, stampare per ogni mese il numero di persone che hanno alloggiato in una camera in quel mese.
8. Dato il codice di una prenotazione di una stanza stampare le informazioni di tutte le persone che sono stati ospitati nella stanza senza aver prenotato a loro nome e per ognuna di esse le informazioni relative alla prenotazione.

9. Stampare il codice e il nominativo di tutte le prenotazioni ancora non pagate.
10. Stampare per ogni ricevuta di prenotazione il codice e il costo di permanenza della prenotazione e il valore delle spese aggiuntive pagato.
11. Dato un anno, stampare per ogni mese la somma degli importi pagati dai clienti in quel mese, il numero di pagamenti totali e il numero di pagamenti per ogni tipo di pagamento.
12. Stampare le informazioni di tutti gli hotel e dei servizi offerti da ciascun hotel.
13. Stampare il nome, cognome e telefono di tutte le persone che hanno fatto un ordine a una lavanderia e il costo del loro ordine.
14. Stampare il nome, cognome e telefono di tutte le persone che hanno partecipato a un convegno e prenotato un tavolo di un ristorante nell'hotel dove il convegno è stato presentato.
15. Stampare le informazioni di tutte le persone che hanno prenotato un tavolo in un qualsiasi hotel e non sono mai stati ospiti di un hotel della catena.
16. Dato un anno e un hotel, stampare per ogni mese il numero di prenotazioni ricevute e la somma degli importi pagati per le prenotazioni in quel mese, il numero di ordini di lavanderia e la somma dei costi degli ordini, il numero di prenotazioni di ristorante e la somma dei costi delle consumazioni, il numero di acquisti di forniture e la somma dei loro costi.
17. Stampare per ogni attività di manutenzione il nome, la città e la via dell'hotel e il numero della stanza in cui si è verificato il guasto, la descrizione del guasto, le spese di manutenzione e il tipo di manutenzione.
18. Stampare per ciascun servizio di ciascun hotel il numero di dipendenti afferenti a quel servizio e la loro età media.
19. Per ogni servizio di ogni hotel si vuole sapere lo stipendio medio dei dipendenti afferenti a quel servizio, e la somma degli stipendi per ogni servizio.
20. Stampare sulla stessa riga per ogni hotel il numero di manutenzioni per categoria eseguite in quell'hotel.
21. Stampare sulla stessa riga per ogni hotel il numero di pagamenti per tipo di pagamento e l'incasso totale per tutte le fatture di prenotazione.
22. Le camere dell'hotel sheraton che hanno un numero di prenotazioni sopra la media.
23. Tutti i convegni ordinati per numero di posti prenotati e data, tenuti negli hotel diversi dall' hotel casino e che hanno avuto un numero di posti prenotati maggiore di almeno uno dei convegni tenuti nell'hotel casino.
24. Tutte le persone che sono state più di due volte clienti della catena alberghiera.
25. Stampare per ogni servizio di ogni hotel e per ogni mese la somma delle spese per l'acquisto dei prodotti di quel servizio.

3.2 Ristrutturazione del diagramma E-R

Prima di passare allo schema logico, Lo schema E-R deve essere ristrutturato per soddisfare due esigenze: quella di semplificare la traduzione e quella di ottimizzare il progetto. La semplificazione si rende necessaria perché non tutti i costrutti del modello E-R hanno una traduzione naturale nei modelli logici corrispondenti (come ad esempio le generalizzazioni). Inoltre, diverse scelte di traduzione sono in funzione delle tavole dei volumi e delle operazioni illustrate precedentemente. Il diagramma E-R ristrutturato è riportato in Figura 3.1

3.3 Diagramma E-R normalizzato

Questo diagramma permette di definire un ordinamento topologico (*cosa dipende da cosa*) delle relazioni, raccoglie gli identificatori principali e evidenzia le relazioni che dovranno essere definite insieme ai vincoli di integrità referenziale. Il diagramma E-R normalizzato è riportato in Figura 3.2.

3.4 Traduzione verso il modello relazione

In questa sezione le corrispondenti entità e associazioni del diagramma E-R normalizzato verranno tradotte negli equivalenti costrutti del modello relazionale. In seguito verranno proposti schemi di relazione dove la sottolineatura indica il vincolo di chiave mentre il *corsivo* indica il vincolo di integrità referenziale e il * un possibile valore nullo.

AttivitàManutenzione(Stanza, *NomeHotel*, *ViaHotel*, *CittàHotel*, Data, Categoria,
Prezzo, Descrizione, *Società*)

Convegno(Data, Nome, *Stanza*, *NomeHotel*, *ViaHotel*, *CittàHotel*)

Dipendente(CF, Nome, Cognome, DataNascita, Stipendio, DataAssunzione,
NomeServizio, *NomeHotel*, *ViaHotel*, *CittàHotel*)

FatturaOrdine(IdFattura, Data, Totale, PezziTotali,
NomeServizio, *NomeHotel*, *ViaHotel*, *CittàHotel*)

FatturaPrenotazione(IdFattura, Totale, Data, *PrenotazioneStanza*)

Hotel(Nome, Via, Città, Stelle, Telefono)

Ordine(FatturaOrdine, Prodotto, Quantità, Prezzo)

Ospite(PrenotazioneStanza, Persona)

Persona(CF, Nome, Cognome, Mail, Telefono, DataNascita)

PrenotazioneStanza(IdPrenotazione, DataInizio, DataFine, NumPosti,
Stanza, *NomeHotel*, *ViaHotel*, *CittàHotel*, *Società**, *Persona**)

Prodotto(IdProdotto, Nome, Marca)

Servizio(Nome, *NomeHotel*, *ViaHotel*, *CittàHotel*)

Società(PartitaIVA, Nome, Mail, Telefono)

Stanza(IdStanza, *NomeHotel*, *ViaHotel*, *CittàHotel*, Prezzo,
NumPosti, *Nome**)

UsoServizio(Persona, *NomeServizio*, *NomeHotel*, *ViaHotel*, *CittàHotel*, Data)

Una rappresentazione grafica dell'intero schema della basi di dati è riportata in Figura 3.3 dove le frecce indicano i vincoli di integrità referenziale.

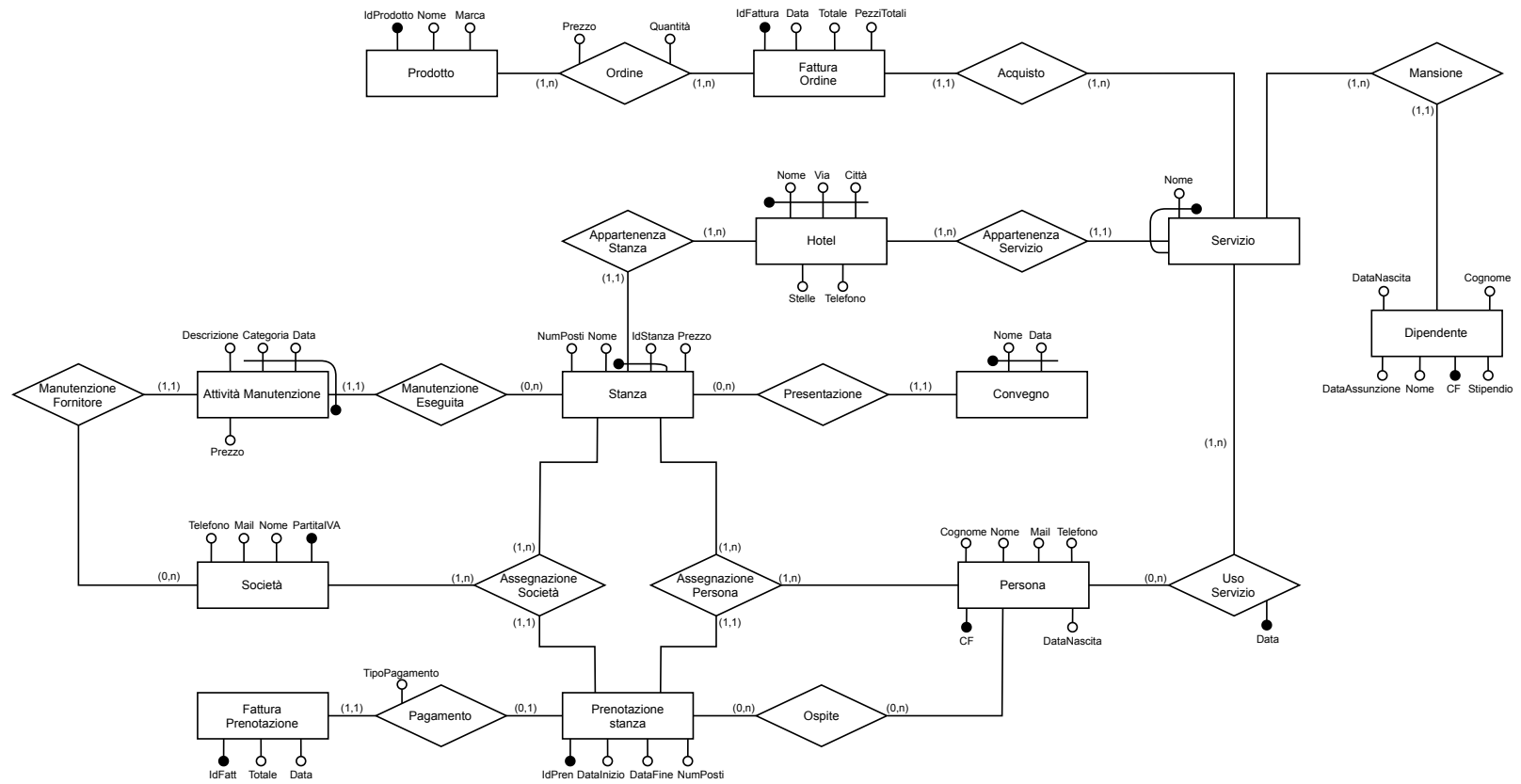


Figura 3.1: Il diagramma E-R ristrutturato dell'intero progetto *Hotel Statistics*

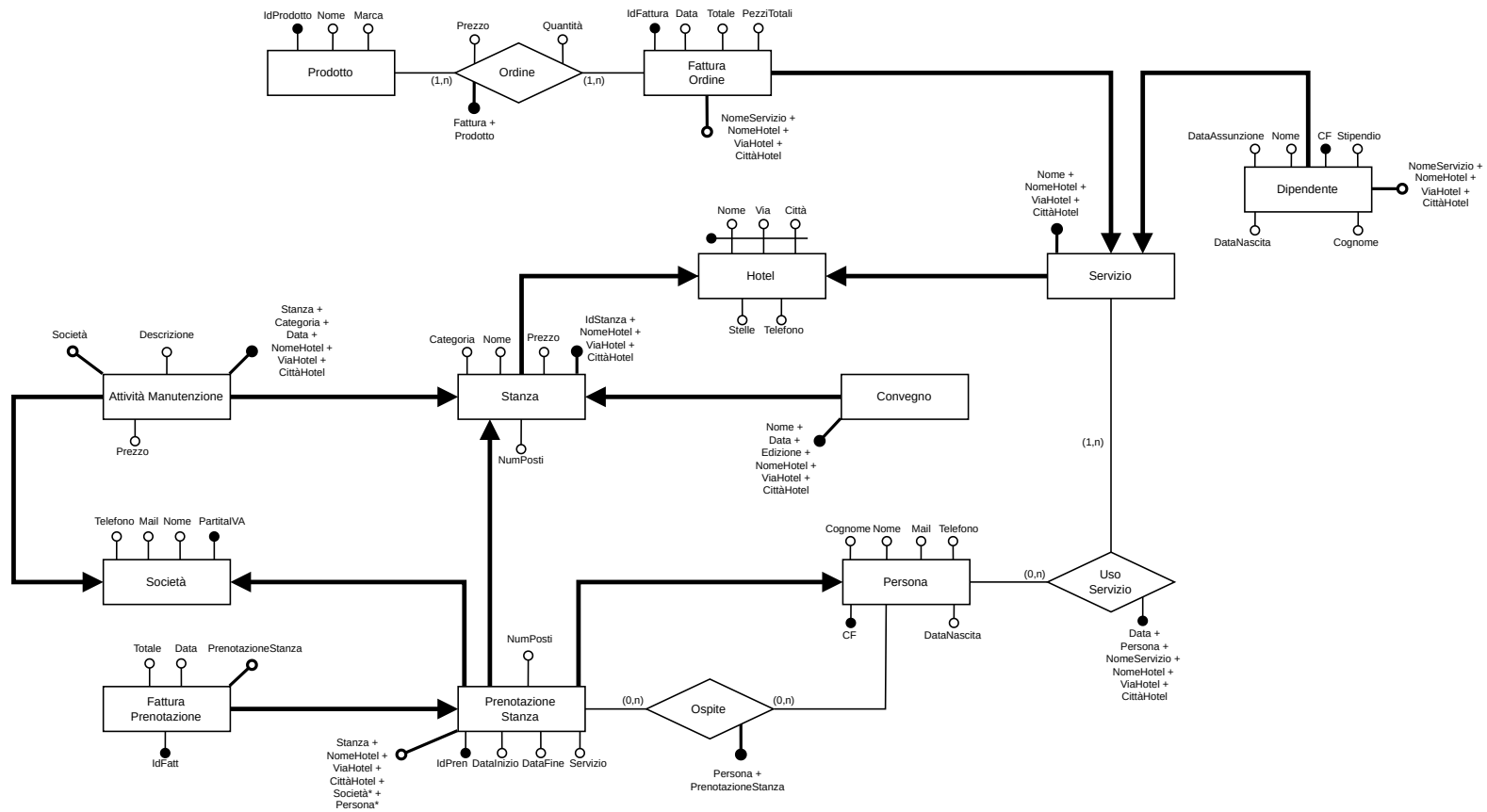


Figura 3.2: Il diagramma E-R normalizzato dell'intero progetto *Hotel Statistics*

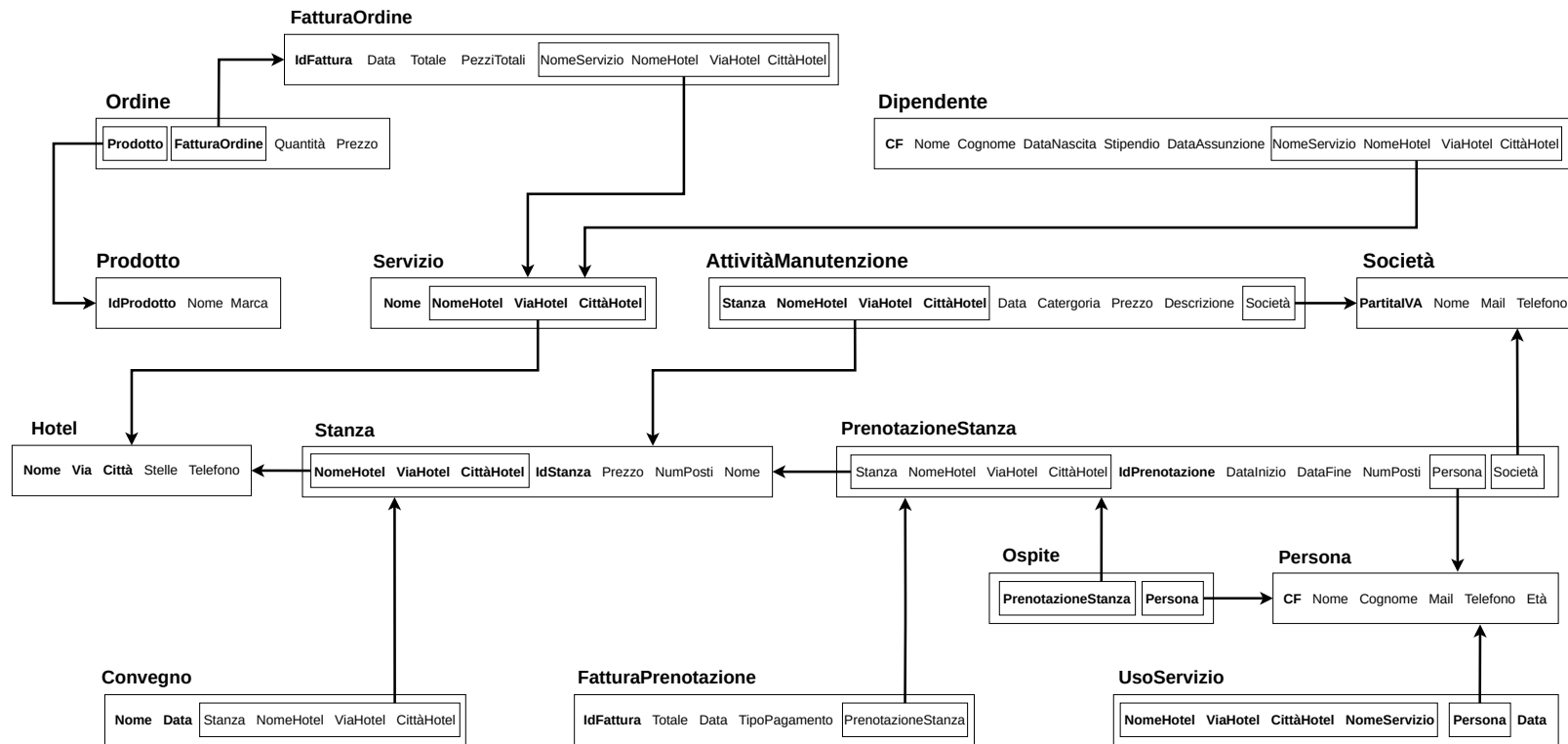


Figura 3.3: Rappresentazione grafica della traduzione nel modello relazionale

3.5 Normalizzazione

Le forme normali ci permettono di certificare la qualità dello schema logico di una base dati. Di tutte le forme normali, quella che garantisce il giusto compromesso tra ridondanze e anomalie è la terza. Una relazione è in **3NF** (Terza Forma Normale) se per ogni dipendenza funzionale (non banale) $X \rightarrow A$ definita su di essa, almeno una delle seguenti condizioni è verificata:

- X contiene una chiave K della relazione
- A appartiene ad almeno una chiave della relazione

Si nota come il vincolo di dipendenza funzionale generalizzi il vincolo di chiave. La dipendenza funzionale $X \rightarrow A$ che rappresenta il vincolo di chiave (quando $X \cup A$ è l'insieme di tutti gli attributi della relazione) soddisfa automaticamente la terza forma formale. In particolare, se ogni relazione della basi dati ha solamente questo tipo di dipendenza funzionale, la 3NF è banalmente soddisfatta. Questo è il caso di *Hotel Statistics*, dove ogni relazione possiede solamente la dipendenza funzionale relativa alla chiave. Le dipendenze funzionali individuate riportate in Tabella 3.1 sono tutti vincoli di chiave. La base di dati *Hotel Statistics* è così in 3NF.

Relazione	Dipendenza funzionale
AttivitàManutenzione	Stanza, NomeHotel, ViaHotel, CittàHotel, Data, Categoria \rightarrow Prezzo, Descrizione, Società
Convegno	Data, Nome \rightarrow Stanza, NomeHotel, ViaHotel, CittàHotel
Dipendente	CF \rightarrow Nome, Cognome, DataNascita, Stipendio, DataAssunzione, NomeServizio, NomeHotel, ViaHotel, CittàHotel
FatturaOrdine	IdFattura \rightarrow Data, Totale, PezziTotali, NomeServizio, NomeHotel, ViaHotel, CittàHotel
FatturaPrenotazione	IdFattura \rightarrow Totale, Data, PrenotazioneStanza
Hotel	Nome, Via, Città \rightarrow Stelle, Telefono
Ordine	FatturaOrdine, Prodotto \rightarrow Quantità, Prezzo
Ospite	PrenotazioneStanza, Persona \rightarrow PrenotazioneStanza, Persona
PrenotazioneStanza	IdPrenotazione \rightarrow DataInizio, DataFine, NumPosti, Stanza, NomeHotel, ViaHotel, CittàHotel, Società, Persona
Persona	CF \rightarrow Nome, Cognome, Mail, Telefono, DataNascita
Prodotto	IdProdotto \rightarrow Nome, Marca
Servizio	Nome, NomeHotel, ViaHotel, CittàHotel \rightarrow Nome, NomeHotel, ViaHotel, CittàHotel
Società	PartitaIVA \rightarrow Nome, Mail, Telefono
Stanza	IdStanza \rightarrow NomeHotel, ViaHotel, CittàHotel, Prezzo, NumPosti, Nome
UsoServizio	Persona, NomeServizio, NomeHotel, ViaHotel, CittàHotel, Data \rightarrow Persona, NomeServizio, NomeHotel, ViaHotel, CittàHotel, Data

Tabella 3.1: Dipendenze funzionali dello schema di base di dati *Hotel Statistics*

3.6 SQL DDL (Data Definition Language)

Utilizzando il diagramma E-R Normalizzato e la traduzione nel modello relazionale è possibile definire l'ordinamento topologico nella definizione delle tabelle. In seguito i comandi SQL DDL di ogni tabella rispettando tale ordine, dove ogni comando è scritto in MAIUSCOLO. Il file completo si trova presso questo [link](#).

```
CREATE TABLE Hotel
(
    Nome VARCHAR(50),
    Via VARCHAR(50),
    Citta VARCHAR(30),
    Stelle TINYINT UNSIGNED CHECK (STELLE > 0 AND STELLE <= 5),
    Telefono VARCHAR(15) NOT NULL,

    PRIMARY KEY (Nome,Via,Citta)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

```
CREATE TABLE Persona
(
    CF CHAR(16) PRIMARY KEY,
    Nome VARCHAR(30),
    Cognome VARCHAR(30),
    Mail VARCHAR(100) NOT NULL,
    Telefono VARCHAR(15) NOT NULL,
    DataNascita DATE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

```
CREATE TABLE Societa
(
    PartitaIVA CHAR(11) PRIMARY KEY,
    Nome VARCHAR(50),
    Mail VARCHAR(100) NOT NULL,
    Telefono VARCHAR(15) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

```
CREATE TABLE Prodotto
(
    IdProdotto VARCHAR(15) PRIMARY KEY,
    Nome VARCHAR(50),
    Marca VARCHAR(50)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

```
CREATE TABLE Stanza
(
    IdStanza INT UNSIGNED,
    NomeHotel VARCHAR(50),
    ViaHotel VARCHAR(50),
    CittaHotel VARCHAR(30),
    Prezzo DECIMAL(7,2),
    NumPosti INT UNSIGNED,
    Nome VARCHAR(30),

    PRIMARY KEY (IdStanza, NomeHotel, ViaHotel, CittaHotel),
    FOREIGN KEY (NomeHotel, ViaHotel, CittaHotel)
        REFERENCES Hotel(Nome, Via, Citta)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

```

CREATE TABLE Servizio
(
  Nome VARCHAR(30),
  NomeHotel VARCHAR(50),
  ViaHotel VARCHAR(50),
  CittaHotel VARCHAR(30),

  PRIMARY KEY (Nome, NomeHotel, ViaHotel, CittaHotel),
  FOREIGN KEY (NomeHotel, ViaHotel, CittaHotel)
    REFERENCES Hotel(Nome, Via, Citta)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

```

```

CREATE TABLE FatturaOrdine
(
  IdFattura CHAR(14),
  Data DATE,
  Totale DECIMAL(8,2),
  PezziTotali INT UNSIGNED,
  NomeServizio VARCHAR(30),
  NomeHotel VARCHAR(50),
  ViaHotel VARCHAR(50),
  CittaHotel VARCHAR(30),

  PRIMARY KEY (IdFattura),
  FOREIGN KEY (NomeServizio, NomeHotel, ViaHotel, CittaHotel)
    REFERENCES Servizio(Nome, NomeHotel, ViaHotel, CittaHotel)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

```

```

CREATE TABLE Ordine
(
  Fattura CHAR(14),
  Prodotto VARCHAR(15),
  Quantita INT UNSIGNED,
  Prezzo DECIMAL (7,2),

  PRIMARY KEY (Fattura,Prodotto),
  FOREIGN KEY (Fattura) REFERENCES FatturaOrdine(IdFattura),
  FOREIGN KEY (Prodotto) REFERENCES Prodotto(IdProdotto)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

```

```

CREATE TABLE AttivitaManutenzione
(
  Stanza INT UNSIGNED,
  NomeHotel VARCHAR(50),
  ViaHotel VARCHAR(50),
  CittaHotel VARCHAR(30),
  Data DATE,
  Categoria VARCHAR(50)
    CHECK (Categoria IN ('Idraulica','Elettrica',
                        'Pulizia','Edile','Arredo')),
  Prezzo DECIMAL(6,2),
  Descrizione TEXT,
  Societa CHAR(11),

  PRIMARY KEY (Stanza, NomeHotel, CittaHotel, ViaHotel, Data, Categoria),
  FOREIGN KEY (Societa) REFERENCES Societa(PartitaIVA),
  FOREIGN KEY (Stanza, NomeHotel, ViaHotel, CittaHotel)
    REFERENCES Stanza(IdStanza, NomeHotel, ViaHotel, CittaHotel)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

```



```

CREATE TABLE Convegno
(
    Data DATE,
    Nome VARCHAR(100),
    Stanza INT UNSIGNED,
    NomeHotel VARCHAR(50),
    ViaHotel VARCHAR(50),
    CittaHotel VARCHAR(30),

    PRIMARY KEY (Data,Nome),
    FOREIGN KEY (Stanza, NomeHotel, ViaHotel, CittaHotel)
        REFERENCES Stanza(IdStanza,NomeHotel, ViaHotel, CittaHotel)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

```

```

CREATE TABLE Dipendente
(
    CF CHAR(16) PRIMARY KEY,
    Nome VARCHAR(30),
    Cognome VARCHAR(30),
    DataNascita DATE,
    Stipendio DECIMAL(6,2),
    DataAssunzione DATE,
    NomeServizio VARCHAR(50),
    NomeHotel VARCHAR(50),
    ViaHotel VARCHAR(50),
    CittaHotel VARCHAR(30),

    FOREIGN KEY (NomeServizio, NomeHotel, ViaHotel, CittaHotel)
        REFERENCES Servizio(Nome, NomeHotel, ViaHotel, CittaHotel)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

```

```

CREATE TABLE PrenotazioneStanza
(
    IdPrenotazione CHAR(10) PRIMARY KEY,
    DataInizio DATE,
    DataFine DATE,
    NumPosti INT UNSIGNED,
    Stanza INT UNSIGNED,
    NomeHotel VARCHAR(50),
    ViaHotel VARCHAR(50),
    CittaHotel VARCHAR(30),
    Societa CHAR(11),
    Persona CHAR(16),

    FOREIGN KEY (Stanza, NomeHotel, ViaHotel, CittaHotel)
        REFERENCES Stanza(IdStanza, NomeHotel, ViaHotel, CittaHotel),
    FOREIGN KEY (Societa) REFERENCES Societa(PartitaIVA),
    FOREIGN KEY (Persona) REFERENCES Persona(CF)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

```

```

CREATE TABLE FatturaPrenotazione
(
    IdFattura CHAR(14) PRIMARY KEY,
    Totale DECIMAL(8,2),
    Data DATE,
    TipoPagamento VARCHAR(50)
        CHECK (TipoPagamento IN ('Contanti','Carta','Bonifico')),
    PrenotazioneStanza CHAR(10) REFERENCES PrenotazioneStanza(IdPrenotazione)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

```

```
CREATE TABLE Ospite
(
    PrenotazioneStanza CHAR(10),
    Persona CHAR(16),

    PRIMARY KEY (PrenotazioneStanza, Persona),
    FOREIGN KEY (PrenotazioneStanza)
        REFERENCES PrenotazioneStanza(IdPrenotazione),
    FOREIGN KEY (Persona) REFERENCES Persona(CF)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

```
CREATE TABLE UsoServizio
(
    NomeServizio VARCHAR(30),
    Persona CHAR(16),
    NomeHotel VARCHAR(50),
    ViaHotel VARCHAR(50),
    CittaHotel VARCHAR(30),
    Data DATE,

    PRIMARY KEY (NomeServizio, Persona, NomeHotel, ViaHotel, CittaHotel, Data),
    FOREIGN KEY (NomeServizio, NomeHotel, ViaHotel, CittaHotel)
        REFERENCES Servizio(Nome, NomeHotel, ViaHotel, CittaHotel),
    FOREIGN KEY (Persona) REFERENCES Persona(CF)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

3.7 SQL DML (Data Manipulation Language)

Per ogni relazione definita precedentemente, si riportano esempi di inserimento. Tutti i file completi relativi ad ogni relazione, si trovano presso questo [link](#).

```
INSERT INTO FatturaOrdine VALUES
("AEA-6261271061", "2021-01-18", 11062.00, 998,
"Discoteca", "Hotel Sheraton", "Via Marsala 20", "Roma");
```

```
INSERT INTO Persona VALUES
("BSCGRL94P26G894M", "Gabriele", "Biscetti",
"gabriele.biscetti@gmail.com", "0343708430", "1994-09-16");
```

```
INSERT INTO Societa VALUES
("00006572649", "Anderson Halvorson and Feil",
"andersonhalvorsonandfeil@gmail.com", "062352142");
```

```
INSERT INTO Prodotto VALUES
("0012516355545", "Tovaglie", "Moroccan Home Decor");
```

```
INSERT INTO Stanza VALUES
(1, "Hotel Casino", "Corso degli Inglesi 18", "Sanremo", 229.00, 4, NULL);
```

```
INSERT INTO Servizio VALUES
("Bar", "Hotel Casino", "Corso degli Inglesi 18", "Sanremo");
```

```
INSERT INTO Ordine VALUES
("AWD-4877251625", "0012516355545", 300, 2209.00);
```

```
INSERT INTO AttivitaManutenzione VALUES
(3, "Hotel Overlook", "Via Cavour 333", "Torino",
"2021-06-06", "Elettrica", 149.00, NULL, "85375573224");
```

```
INSERT INTO Convegno VALUES
("2021-01-03", "Home Furnishings", 1000,
"Hotel Sheraton", "Via Marsala 20", "Roma");
```

```
INSERT INTO Dipendente VALUES
("ACRJFV41N20C113Z", "Sisely", "Clough", "1982-09-06", 1476.00, "2021-01-01",
"Ristorante", "Hotel Sheraton", "Via Marsala 20", "Roma");
```

```
INSERT INTO PrenotazioneStanza VALUES
("0000095737", "2021-08-09", "2021-08-10", 3, 234, "Hotel Overlook",
"Via Cavour 333", "Torino", NULL, "XKMEHU54W58J257I")
```

```
INSERT INTO FatturaPrenotazione VALUES
("AAA-3081197946", 1152.00, "2021-08-08", "Bonifico", "4216656165");
```

```
INSERT INTO UsoServizio VALUES
("Bar", "AAACNY32D07Y758R", "Hotel Sheraton",
"Via Marsala 20", "Roma", "2021-03-07");
```

```
INSERT INTO Ospite VALUES ("0307939079", "AAABCY20M02H477Z");
```

3.8 SQL DQL (Data Query Language)

Le operazioni riportate precedentemente sono ora tradotte in SQL. Il file completo si trova presso questa [pagina](#).

1. Stampare nome, cognome e numero di telefono di tutte le persone che sono o sono stati ospiti in una stanza d'hotel.

```
SELECT DISTINCT p.Nome, p.Cognome, p.Telefono
FROM Ospite o JOIN Persona p ON o.Persona = p.CF;
```

2. Stampare le informazioni di tutti i convegni, il numero di partecipanti, il nome della sala convegno e dell'hotel dove sono stati presentati.

```
SELECT C.Nome as Convegno, C.Data as Data, C.NomeHotel as Hotel,
S.Nome as NomeSala, count(*) as NumeroOspiti
FROM Convegno C, Stanza S, PrenotazioneStanza PS, Ospite O
WHERE C.Stanza = S.IdStanza AND
C.NomeHotel = S.NomeHotel AND
C.ViaHotel = S.ViaHotel AND
C.CittaHotel = S.CittaHotel AND
PS.Stanza = S.IdStanza AND
PS.NomeHotel = S.NomeHotel AND
PS.ViaHotel = S.ViaHotel AND
PS.CittaHotel = S.CittaHotel AND
PS.Stanza = S.IdStanza AND
PS.NomeHotel = S.NomeHotel AND
PS.ViaHotel = S.ViaHotel AND
```

```

        PS.CittaHotel = S.CittaHotel AND
        PS.IdPrenotazione = O.PrenotazioneStanza AND
        C.Data = PS.DataInizio
GROUP BY C.Nome,C.Data;

```

3. Stampare nome, cognome e mail di tutti i partecipanti del/dei convegno/i con nome 'Database Administrator I'

```

SELECT P.Nome,P.Cognome,P.Mail
FROM Convegno C
JOIN PrenotazioneStanza PS
    ON C.Stanza = PS.Stanza AND
       C.NomeHotel = PS.NomeHotel AND
       C.ViaHotel = PS.ViaHotel AND
       C.CittaHotel = PS.CittaHotel
JOIN Ospite O
    ON O.PrenotazioneStanza = PS.IdPrenotazione
JOIN Persona P
    ON P.CF = O.Persona
WHERE C.Nome = "Database Administrator I";

```

4. Data la stanza N.186 dell'Hotel Overlook trovare (in ordine di DataInizio) se esistono le possibili prenotazioni con periodo di permanenza sovrapponibile al periodo con DataInizio 2021-10-08 e DataFine 2021-10-13.

```

SELECT PS.IdPrenotazione, PS.DataInizio, PS.DataFine
FROM PrenotazioneStanza PS
WHERE PS.Stanza = 186 AND
       PS.NomeHotel = "Hotel Overlook" AND
       PS.ViaHotel = "Via Cavour 333" AND
       PS.CittaHotel = "Torino" AND
       ((PS.DataInizio BETWEEN "2021-10-08" AND "2021-10-13") OR
        (PS.DataFine BETWEEN "2021-10-08" AND "2021-10-13"))
ORDER BY PS.DataInizio;

```

5. Dato l'Hotel Overlook e il periodo di permanenza DataInizio = 2022-01-08 e DataFine = 2022-01-13 e trovare tutte le stanze non prenotate per quel periodo nell'hotel dato

```

SELECT S.IdStanza
FROM Stanza S
WHERE S.NomeHotel = "Hotel Overlook" AND
       S.ViaHotel = "Via Cavour 333" AND
       S.CittaHotel = "Torino" AND
       S.IdStanza NOT IN
           (SELECT PS.Stanza
            FROM PrenotazioneStanza PS
            WHERE PS.NomeHotel = "Hotel Overlook" AND
                  PS.ViaHotel = "Via Cavour 333" AND
                  PS.CittaHotel = "Torino" AND
                  ((PS.DataInizio BETWEEN "2021-10-08" AND "2021-10-13") OR
                   (PS.DataFine BETWEEN "2021-10-08" AND "2021-10-13")));

```

6. Dato l'anno 2021, stampare per ogni mese la somma degli importi pagati dai clienti in quel mese e il numero di fatture emesse.

```

SELECT MONTHNAME(FP.Data) as Mese, sum(FP.Totale) as ImportiPagati,
       count(*) as NumeroFattureEmesse
FROM FatturaPrenotazione FP
WHERE YEAR(FP.Data) = "2021"
GROUP BY MONTHNAME(FP.Data)
ORDER BY MONTHNAME(FP.Data);

```

7. Dato l'anno, stampare per ogni mese il numero di persone che hanno alloggiato in ciascun hotel in quel mese.

```

SELECT MONTHNAME(PS.DataFine), PS.NomeHotel, PS.ViaHotel,
       PS.CittaHotel, count(*) as NumeroOspiti
FROM Ospite O
JOIN PrenotazioneStanza PS
  ON O.PrenotazioneStanza = PS.IdPrenotazione
JOIN Stanza S
  ON PS.Stanza = S.IdStanza AND
     PS.NomeHotel = S.NomeHotel AND
     PS.ViaHotel = S.ViaHotel AND
     PS.CittaHotel = S.CittaHotel
WHERE YEAR(PS.DataFine) = "2021"
AND S.Nome IS NULL
GROUP BY MONTHNAME(PS.DataFine), PS.NomeHotel,
         PS.ViaHotel, PS.CittaHotel;

```

8. Dato il codice di prenotazione 2594469299 stampare le informazioni di tutte le persone che sono stati ospitati nella stanza senza aver prenotato a loro nome e per ognuna di esse il codice della relativa prenotazione.

```

SELECT PS.IdPrenotazione, P.CF CF, P.Nome Nome, P.Cognome Cognome,
       P.Mail Mail, P.Telefono Telefono, P.DataNascita DataNascita
FROM PrenotazioneStanza PS
JOIN Ospite O
  ON IdPrenotazione = PrenotazioneStanza
JOIN Persona P
  ON O.Persona = P.CF
WHERE IdPrenotazione = "2594469299" AND
     PS.Persona IS NOT NULL AND
     O.Persona <> PS.Persona;

```

9. Stampare il codice e il nominativo di tutte le prenotazioni ancora non pagate

```

SELECT PS.IdPrenotazione,
       COALESCE(PS.Persona, PS.Societa) as Nominativo
FROM PrenotazioneStanza PS
LEFT JOIN FatturaPrenotazione FP
  ON PS.IdPrenotazione = FP.PrenotazioneStanza
WHERE FP.IdFattura IS NULL;

-- Soluzione alternativa

SELECT PS.IdPrenotazione,
       COALESCE(PS.Persona, PS.Societa) as Nominativo
FROM PrenotazioneStanza PS
WHERE PS.IdPrenotazione NOT IN (SELECT FP.PrenotazioneStanza
                                FROM FatturaPrenotazione FP);

```

10. Stampare per ogni fattura di prenotazione il codice, il costo di prenotazione e la data del pagamento. */

```
SELECT IdFattura, Totale, Data as "Data pagamento"
FROM FatturaPrenotazione;
```

11. Stampare le informazioni di tutti gli hotel e dei servizi offerti da ciascun hotel.

```
SELECT H.Nome, H.Via, H.Citta, S.Nome
FROM Hotel H
JOIN Servizio S
  ON H.Nome = S.NomeHotel AND
     H.Via = S.ViaHotel AND
     H.Citta = S.CittaHotel;
```

12. Stampare il nome, cognome e telefono di tutte le persone che hanno fatto uso di almeno un servizio, e il nome del servizio da loro utilizzato con la data d'utilizzo.

```
SELECT P.Nome Nome, P.Cognome Cognome,
       P.Telefono Telefono, US.NomeServizio, US.Data Data
FROM UsoServizio US
JOIN Persona P
  ON US.Persona = P.CF;
```

13. Stampare il nome, cognome e telefono di tutte le persone che hanno partecipato a un convegno e prenotato un camera nell'hotel dove il convegno è stato presentato.

```
SELECT P.Nome Nome, P.Cognome Cognome, P.Telefono Telefono
FROM Stanza S
JOIN PrenotazioneStanza PS
  ON PS.Stanza = S.IdStanza AND
     PS.NomeHotel = S.NomeHotel AND
     PS.ViaHotel = S.ViaHotel AND
     PS.CittaHotel = S.CittaHotel
JOIN Ospite O
  ON O.PrenotazioneStanza = PS.IdPrenotazione
JOIN Persona P
  ON P.CF = O.Persona
JOIN PrenotazioneStanza PS1
  ON PS1.Persona = P.CF
JOIN Stanza S1
  ON PS1.Stanza = S1.IdStanza AND
     PS1.NomeHotel = S1.NomeHotel AND
     PS1.ViaHotel = S1.ViaHotel AND
     PS1.CittaHotel = S1.CittaHotel
WHERE S.Nome IS NOT NULL AND
      S1.Nome IS NULL AND
      PS.NomeHotel = PS1.NomeHotel AND
      PS.ViaHotel = PS1.ViaHotel AND
      PS.CittaHotel = PS1.CittaHotel;
```

14. Stampare codice fiscale, nome, cognome, numero di telefono di tutte le persone che sono stati ospiti della catena alberghiera più di due volte.

```
SELECT P.CF, P.Nome Nome, P.Cognome Cognome, P.Telefono Telefono
FROM Persona P
JOIN Ospite O
```

```
ON P.CF = O.Persona
GROUP BY P.CF
HAVING count(*) > 2;
```

15. Dato l'anno 2021 e l'Hotel Casino, Corso degli Inglesi 18, Sanremo stampare per ogni mese il numero di prenotazioni ricevute e la somma degli importi pagati per le prenotazioni in quel mese

```
SELECT "2021" as Anno , MONTHNAME(PS.DataFine),
       count(*) NumeroPrenotazioni, sum(FP.Totale) Incasso
FROM PrenotazioneStanza PS
JOIN FatturaPrenotazione FP
ON PS.IdPrenotazione = FP.PrenotazioneStanza
WHERE YEAR(PS.DataFine) = "2021" AND
PS.NomeHotel = "Hotel Casino" AND
PS.ViaHotel = "Corso degli Inglesi 18" AND
PS.CittaHotel = "Sanremo"
GROUP BY MONTHNAME(PS.DataFine);
```

16. Stampare per ogni attività di manutenzione il nome, la città e la via dell'hotel e il numero della stanza in cui si è verificato il guasto, la descrizione del guasto, le spese di manutenzione, il tipo di manutenzione e il nome della società che ha fornito la manutenzione

```
SELECT AM.Stanza, AM.NomeHotel, AM.ViaHotel,
       AM.CittaHotel, AM.Descrizione, AM.Prezzo, AM.Categoria, S.Nome
FROM AttivitaManutenzione AM, Societa S
WHERE AM.Societa = S.PartitaIVA;
```

17. Stampare il numero delle attività di manutenzioni eseguite e il totale delle spese per categoria e per hotel in cui vengono eseguite.

```
SELECT AM.NomeHotel, AM.ViaHotel, AM.CittaHotel, AM.Categoria,
       count(*) as "Numero Manutenzioni",
       sum(AM.Prezzo) as "Totale Spese per Categoria di Manutenzione"
FROM AttivitaManutenzione AM, Societa S
WHERE AM.Societa = S.PartitaIVA
GROUP BY AM.NomeHotel, AM.ViaHotel, AM.CittaHotel, AM.Categoria;
```

18. Stampare per ciascun servizio di ciascun hotel il numero di dipendenti afferenti a quel servizio e la loro età media.

```
SELECT S.Nome, S.NomeHotel, S.ViaHotel, S.CittaHotel, count(*),
       avg(TIMESTAMPDIFF(YEAR, D.DataNascita, CURDATE())) as EtaMedia
FROM Servizio S
JOIN Dipendente D
ON D.NomeServizio = S.Nome AND
   D.NomeHotel = S.NomeHotel AND
   D.ViaHotel = S.ViaHotel AND
   D.CittaHotel = S.CittaHotel
GROUP BY S.Nome, S.NomeHotel, S.ViaHotel, S.CittaHotel;
```

19. Per ogni servizio di ogni hotel si vuole sapere lo stipendio medio dei dipendenti afferenti a quel servizio, e la somma degli stipendi per ogni servizio.

```

SELECT S.Nome, S.NomeHotel, S.ViaHotel, S.CittaHotel,
       avg(D.Stipendio) as "Stipendio Mensile Medio",
       sum(D.Stipendio) as "Somma Annuale Stipendi"
FROM Servizio S
JOIN Dipendente D
ON D.NomeServizio = S.Nome AND
   D.NomeHotel = S.NomeHotel AND
   D.ViaHotel = S.ViaHotel AND
   D.CittaHotel = S.CittaHotel
GROUP BY S.Nome, S.NomeHotel, S.ViaHotel, S.CittaHotel;

```

20. Stampare sulla stessa riga per ogni hotel il numero di manutenzioni per categoria eseguite in quell'hotel.

```

SELECT AM.NomeHotel, AM.ViaHotel, AM.CittaHotel,
       SUM(CASE WHEN AM.Categoria = "Idraulica"
                THEN 1 ELSE 0 END) "Manutenzioni Idrauliche",
       SUM(CASE WHEN AM.Categoria = "Elettrica"
                THEN 1 ELSE 0 END) "Manutenzioni Elettriche",
       SUM(CASE WHEN AM.Categoria = "Pulizia"
                THEN 1 ELSE 0 END) "Manutenzioni Pulizie",
       SUM(CASE WHEN AM.Categoria = "Edile"
                THEN 1 ELSE 0 END) "Manutenzioni Edili",
       SUM(CASE WHEN AM.Categoria = "Arredo"
                THEN 1 ELSE 0 END) "Manutenzioni Arredo"
FROM AttivitaManutenzione AM
GROUP BY AM.NomeHotel, AM.ViaHotel, AM.CittaHotel;

```

21. Stampare sulla stessa riga per ogni hotel il numero di pagamenti per tipo di pagamento e l'incasso totale per tutte le fatture di prenotazione.

```

SELECT PS.NomeHotel, PS.ViaHotel, PS.CittaHotel,
       SUM(CASE WHEN FP.TipoPagamento = "Bonifico"
                THEN 1 ELSE 0 END) "Numero Pagamenti con Bonifico",
       SUM(CASE WHEN FP.TipoPagamento = "Carta"
                THEN 1 ELSE 0 END) "Numero Pagamenti con Carta",
       SUM(CASE WHEN FP.TipoPagamento = "Contanti"
                THEN 1 ELSE 0 END) "Numero Pagamenti con Contanti",
       SUM(FP.Totale) "Incasso Totale"
FROM FatturaPrenotazione FP
JOIN PrenotazioneStanza PS
ON FP.PrenotazioneStanza = PS.IdPrenotazione
GROUP BY PS.NomeHotel, PS.ViaHotel, PS.CittaHotel;

```

22. Le camere dell'hotel sheraton che hanno un numero di prenotazioni sopra la media.

```

SELECT PS.Stanza, count(*)
FROM PrenotazioneStanza PS
JOIN Stanza S
ON PS.Stanza = S.IdStanza AND
   PS.NomeHotel = S.NomeHotel AND
   PS.CittaHotel = S.CittaHotel AND
   PS.ViaHotel = S.ViaHotel
WHERE PS.NomeHotel = "Hotel Sheraton" AND
      PS.ViaHotel = "Via Marsala 20" AND
      PS.CittaHotel = "Roma" AND
      S.Nome IS NULL

```



```

GROUP BY PS.Stanza
HAVING count(*) > ALL
( SELECT count(*)/ (SELECT count(*)
                     FROM Stanza S
                     WHERE S.Nome IS NULL AND
                           S.NomeHotel = "Hotel Sheraton" AND
                           S.ViaHotel = "Via Marsala 20" AND
                           S.CittaHotel = "Roma")
  FROM PrenotazioneStanza PS
  JOIN Stanza S
  ON PS.Stanza = S.IdStanza AND
  PS.NomeHotel = S.NomeHotel AND
  PS.CittaHotel = S.CittaHotel AND
  PS.ViaHotel = S.ViaHotel
  WHERE PS.NomeHotel = "Hotel Sheraton" AND
  PS.ViaHotel = "Via Marsala 20" AND
  PS.CittaHotel = "Roma" AND
  S.Nome IS NULL
);

```

23. Tutti i convegni ordinati per numero di posti prenotati e data, tenuti negli hotel diversi dall' hotel casino e che hanno avuto un numero di posti prenotati maggiore di almeno uno dei convegni tenuti nell'hotel casino.

```

SELECT PS.NomeHotel, PS.DataInizio, PS.NumPosti, C.Nome
FROM PrenotazioneStanza PS
JOIN Stanza S
ON PS.Stanza = S.IdStanza AND
  PS.NomeHotel = S.NomeHotel AND
  PS.CittaHotel = S.CittaHotel AND
  PS.ViaHotel = S.ViaHotel
JOIN Convegno C
ON C.Stanza = PS.Stanza AND
  C.NomeHotel = PS.NomeHotel AND
  C.CittaHotel = PS.CittaHotel AND
  C.ViaHotel = PS.ViaHotel AND
  C.Data = PS.DataInizio
WHERE S.Nome IS NOT NULL AND
  PS.NomeHotel <> "Hotel Casino" AND
  PS.ViaHotel <> "Corso degli Inglesi 18" AND
  PS.CittaHotel <> "Sanremo" AND
  PS.NumPosti > ANY
( SELECT PS.NumPosti
  FROM PrenotazioneStanza PS
  JOIN Stanza S
  ON PS.Stanza = S.IdStanza AND
  PS.NomeHotel = S.NomeHotel AND
  PS.CittaHotel = S.CittaHotel AND
  PS.ViaHotel = S.ViaHotel
  WHERE S.Nome is not null AND
  PS.NomeHotel = "Hotel Casino" AND
  PS.ViaHotel = "Corso degli Inglesi 18" AND
  PS.CittaHotel = "Sanremo" )
ORDER BY PS.NumPosti, PS.DataInizio;

```

24. Tutte le persone che sono state più di due volte clienti della catena alberghiera

```

SELECT SQL_NO_CACHE P.*
FROM Persona P

```

```
WHERE P.CF IN
      (SELECT O.Persona
       FROM Ospite O
       GROUP BY O.Persona
       HAVING count(*) > 2);
```

25. Stampare per ogni servizio di ogni hotel e per ogni mese la somma delle spese per l'acquisto dei prodotti di quel servizio.

```
SELECT MONTHNAME(FO.Data), S.Nome Servizio, S.NomeHotel,
       S.ViaHotel, S.CittaHotel, sum(O.Prezzo) "Spese per Servizio"
FROM Servizio S
JOIN FatturaOrdine FO
ON FO.NomeServizio = S.Nome AND
   FO.NomeHotel = S.NomeHotel AND
   FO.ViaHotel = S.ViaHotel AND
   FO.CittaHotel = S.CittaHotel
JOIN Ordine O
ON O.Fattura = FO.IdFattura
GROUP BY MONTHNAME(FO.Data), S.Nome,
         S.NomeHotel, S.ViaHotel, S.CittaHotel;
```

3.9 Viste

Le viste ci permettono di definire delle relazioni “Virtuali” il cui contenuto dipende da altre tabelle. Possono essere utilizzate come meccanismo per semplificare query complesse, come meccanismo di autorizzazione per limitare l'accesso a determinate tabelle o specifici campi e per aggregare dati utili consultati frequentemente.

A differenza delle tabelle, le viste non sono memorizzate (ad eccezione di quelle *materializzate*, non disponibili in MariaDB e nemmeno in MySQL). Questo limita le operazioni di aggiornamento possibili, ma non pone alcuna limitazione alla loro interrogazione. Sono definite per mezzo di una query SQL.

In seguito alcuni esempi di possibili viste utili a future applicazioni. Il file completo si trova presso questa [pagina](#).

1. Vista con il nome, cognome e telefono di tutte le persone attualmente ospitate nella catena alberghiera.

```
CREATE VIEW OspitiAttuali(Nome,Cognome,Telefono) AS
SELECT P.Nome,P.Cognome,P.Telefono
FROM Ospite O
JOIN Persona P
ON O.Persona = P.CF
JOIN PrenotazioneStanza PS
ON O.PrenotazioneStanza = PS.IdPrenotazione
WHERE (CURDATE() BETWEEN PS.DataInizio AND PS.DataFine);
```

2. Una vista dove per ogni anno per ogni hotel, si vogliono sapere il numero di prenotazioni

```
CREATE VIEW NumeroPrenotazioniAnnuali (Anno,NomeHotel,ViaHotel,
                                       CittaHotel,NumeroPrenotazioni) AS
SELECT YEAR(DataFine),NomeHotel, ViaHotel,
```

```

        CittaHotel, count(*) as NumeroPrenotazioni

FROM PrenotazioneStanza PS

GROUP BY YEAR(DataFine),NomeHotel, ViaHotel, CittaHotel ;

```

3. Una vista per oscurare il numero di telefono e mail di tutte le Persone

```

CREATE VIEW DataPrivacyPersona(CF,Nome,Cognome,Mail,
                                Telefono,DataNascita) AS

SELECT CF,Nome,Cognome,
       concat(substr(Mail,1,2), '*****', substr(Mail,-4)) Mail,
       concat(substr(Telefono,1,2), '*****') Telefono, DataNascita
FROM Persona ;

```

4. Una vista dove ad ogni società è associato il numero di prenotazioni fatte nell'intera catena alberghiera

```

CREATE VIEW NumeroPrenotazioniSocieta(PartivaIVA, Nome, Mail,
                                       Telefono, NumeroPrenotazioni) AS

SELECT S.PartitaIVA, S.Nome, S.Mail,
       S.Telefono, count(*) NumeroPrenotazioni
FROM PrenotazioneStanza PS JOIN Societa S ON PS.Societa = S.PartitaIVA
WHERE PS.Societa IS NOT NULL
GROUP BY S.PartitaIVA, S.Nome, S.Mail, S.Telefono ;

```

5. Una vista dove all'anno attuale è associata la spesa mensile totale di fornitura di ogni servizio

```

CREATE VIEW SpeseFornitureMensili (NomeHotel,ViaHotel,CittaHotel,
                                   NomeServizio,Mese,SpesaTotale) AS

SELECT S.NomeHotel, S.ViaHotel, S.CittaHotel,
       S.Nome, MONTH(FO.Data), SUM(O.Prezzo)

FROM Servizio S
  JOIN FatturaOrdine FO
    ON S.Nome = FO.NomeServizio AND S.NomeHotel = FO.NomeHotel AND
       S.CittaHotel = FO.CittaHotel AND S.ViaHotel = FO.ViaHotel
  JOIN Ordine O
    ON O.Fattura = FO.IdFattura

WHERE YEAR(FO.Data) = YEAR(CURDATE())

GROUP BY S.NomeHotel, S.ViaHotel, S.CittaHotel, S.Nome, MONTH(FO.Data);

```

3.10 Stored procedure

Le stored procedure sono sicuramente utili per realizzare il principio della separazione degli interessi (separation of concerns) fra client e il DBMS ; la logica applicativa viene implementata nel backend e non nel client che invece deve solamente richiamare procedure con gli opportuni parametri. Questo facilita la manutenzione delle query. Inoltre, l'utilizzo di linguaggi strutturati all'interno delle procedure, permette di realizzare elaborazioni non possibili con il solo uso di SQL. Un altro motivo per utilizzare le

stored procedure sono le prestazioni; la query presente viene compilata e salvata nel DBMS fissando il piano di esecuzione (execution plan).

L'obiettivo di questa sezione è di fornire delle API (Application programming interface) alle future applicazioni le quali non dovranno includere query SQL al loro interno, ma solamente chiamate a stored procedure. In seguito alcune stored procedure utili al progetto *Hotel Statistics*. Il file completo si trova presso questo [link](#).

1. Dato il codice di una prenotazione di una stanza stampare le informazioni di tutte le persone che sono stati ospitati nella stanza senza aver prenotato a loro nome e per ognuna di esse il codice della relativa prenotazione.

```
DELIMITER //
```

```
CREATE PROCEDURE dati_prenotazione( IN id_prenotazione CHAR(10) )
```

```
BEGIN
```

```
SELECT PS.IdPrenotazione, P.CF CF, P.Nome Nome, P.Cognome Cognome,
```

```
       P.DataNascita DataNascita, P.Mail Mail, P.Telefono Telefono
```

```
FROM PrenotazioneStanza PS
```

```
     JOIN Ospite O
```

```
       ON IdPrenotazione = PrenotazioneStanza
```

```
     JOIN Persona P
```

```
       ON O.Persona = P.CF
```

```
WHERE IdPrenotazione = id_prenotazione AND
```

```
       PS.Persona IS NOT NULL AND
```

```
       O.Persona <> PS.Persona;
```

```
END //
```

```
DELIMITER ;
```

2. Eliminare le fatture di ordine più vecchie di X anni

```
DELIMITER //
```

```
SET AUTOCOMMIT = 0;
```

```
CREATE PROCEDURE elimina_vecchi_ordini( IN var_anni INT UNSIGNED )
```

```
BEGIN
```

```
    DECLARE var_rownum INT UNSIGNED DEFAULT 0;
```

```
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
```

```
    BEGIN
```

```
        ROLLBACK;
```

```
        RESIGNAL;
```

```
    END;
```

```
    SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
```

```
    START TRANSACTION;
```

```
    DELETE FROM Ordine
```

```
    WHERE Fattura IN (SELECT IdFattura
```

```
                      FROM FatturaOrdine
```

```

WHERE TIMESTAMPDIFF(YEAR, Data,
                    CURDATE()) >= var_anni);

DELETE FROM FatturaOrdine
WHERE TIMESTAMPDIFF(YEAR, Data, CURDATE()) >= var_anni;

SELECT count(*) INTO var_rownum
FROM FatturaOrdine
WHERE TIMESTAMPDIFF(YEAR, Data, CURDATE()) >= var_anni;

IF var_rownum > 0 THEN
    SIGNAL SQLSTATE '45001' SET MESSAGE_TEXT="Errore cancellazione";
END IF;

COMMIT;

END //

DELIMITER ;

```

3. Stampare sulla stessa riga per ogni hotel il numero di manutenzioni per categoria eseguite in quell'hotel per un dato anno.

```

DELIMITER //

CREATE PROCEDURE print_report_manutenzioni(IN var_anno INT UNSIGNED)

BEGIN

SELECT YEAR(AM.Data) Anno, AM.NomeHotel, AM.ViaHotel, AM.CittaHotel,
    SUM(CASE WHEN AM.Categoria = "Idraulica"
        THEN 1 ELSE 0 END) "Manutenzioni Idrauliche",
    SUM(CASE WHEN AM.Categoria = "Elettrica"
        THEN 1 ELSE 0 END) "Manutenzioni Elettriche",
    SUM(CASE WHEN AM.Categoria = "Pulizia"
        THEN 1 ELSE 0 END) "Manutenzioni Pulizie",
    SUM(CASE WHEN AM.Categoria = "Edile"
        THEN 1 ELSE 0 END) "Manutenzioni Edili",
    SUM(CASE WHEN AM.Categoria = "Arredo"
        THEN 1 ELSE 0 END) "Manutenzioni Arredo"
FROM AttivitaManutenzione AM
WHERE YEAR(AM.Data) = var_anno
GROUP BY YEAR(AM.Data), AM.NomeHotel, AM.ViaHotel, AM.CittaHotel;

END //

DELIMITER ;

```

4. Stampare sulla stessa riga per ogni hotel il numero di pagamenti per tipo di pagamento e l'incasso totale per tutte le fatture di preotazione per un dato anno.

```

DELIMITER //

CREATE PROCEDURE print_report_pagamenti(IN var_anno INT UNSIGNED)

BEGIN

SELECT YEAR(FP.Data) Anno, PS.NomeHotel, PS.ViaHotel, PS.CittaHotel,
    SUM(CASE WHEN FP.TipoPagamento = "Bonifico"

```

```

        THEN 1 ELSE 0 END) "Numero Pagamenti con Bonifico",
SUM(CASE WHEN FP.TipoPagamento = "Carta"
        THEN 1 ELSE 0 END) "Numero Pagamenti con Carta",
SUM(CASE WHEN FP.TipoPagamento = "Contanti"
        THEN 1 ELSE 0 END) "Numero Pagamenti con Contanti",
SUM(FP.Totale) "Incasso Totale"
FROM FatturaPrenotazione FP
JOIN PrenotazioneStanza PS
ON FP.PrenotazioneStanza = PS.IdPrenotazione
WHERE YEAR(FP.Data) = var_anno
GROUP BY YEAR(FP.Data), PS.NomeHotel, PS.ViaHotel, PS.CittaHotel;

END //

DELIMITER ;

```

3.11 Triggers

I triggers permettono di definire regole attive. Queste seguono il paradigma ECA (*Evento-Condizione-Azione*); quando si verifica l'evento, se la condizione è soddisfatta allora viene svolta l'azione. I triggers permettono di implementare le regole aziendali, convalidare dati in input, verificare integrità dei dati o reagire a modifiche (come annullare effetti indesiderati o registrare le attività di una specifica tabella). Permettono di realizzare *l'indipendenza della conoscenza*; la conoscenza di tipo reattivo viene sottratta ai programmi applicativi e codificata sotto forma di regole attive.

In seguito sono presentati triggers che implementano le regole aziendali proposte nella progettazione concettuale. Il file completo si trova presso questo [link](#).

1. Una stanza non deve avere più prenotazioni con periodo di permanenza sovrapponibili.

```

DELIMITER //

CREATE TRIGGER prenotazione_before_insert
BEFORE INSERT ON PrenotazioneStanza FOR EACH ROW

BEGIN

DECLARE numero_prenotazioni_sovrapponibili INT UNSIGNED;

SELECT count(*) INTO numero_prenotazioni_sovrapponibili
FROM PrenotazioneStanza PS
WHERE PS.Stanza = NEW.Stanza AND
      PS.NomeHotel = NEW.NomeHotel AND
      PS.ViaHotel = NEW.ViaHotel AND
      PS.CittaHotel = NEW.CittaHotel AND
      ((PS.DataInizio BETWEEN NEW.DataInizio AND NEW.DataFine) OR
      (PS.DataFine BETWEEN NEW.DataInizio AND NEW.DataFine));

IF ( numero_prenotazioni_sovrapponibili >= 1 ) THEN

    SIGNAL SQLSTATE '45001'
    SET MESSAGE_TEXT='Non è possibile prenotare
                        la stanza per il periodo di
                        permanenza indicato';

END IF;

DELIMITER ;

```

```
END //

DELIMITER ;
```

2. Il numero di posti della prenotazione di una stanza deve essere minore o uguale al numero di posti della stanza prenotata.

```
DELIMITER //

CREATE TRIGGER prenotazione2_before_insert
BEFORE INSERT ON PrenotazioneStanza FOR EACH ROW

BEGIN

DECLARE numero_posti_stanza INT UNSIGNED;

SELECT S.NumPosti INTO numero_posti_stanza
FROM Stanza S
WHERE S.IdStanza = NEW.Stanza AND S.NomeHotel = NEW.NomeHotel AND
      S.ViaHotel = NEW.ViaHotel AND S.CittaHotel = NEW.CittaHotel;

IF (numero_posti_stanza < NEW.NumPosti) THEN

SIGNAL SQLSTATE '45002'
SET MESSAGE_TEXT='I posti della stanza prenotata sono insufficienti !';

END IF;

END //

DELIMITER ;
```

3. Il numero di ospiti di una stanza prenotata deve essere uguale al numero di posti per la relativa prenotazione della stanza.

```
DELIMITER //

CREATE TRIGGER ospite_before_insert
BEFORE INSERT ON Ospite FOR EACH ROW

BEGIN

DECLARE numero_ospiti_prenotazione int;
DECLARE numero_posti_prenotati int;

SELECT count(*) INTO numero_ospiti_prenotazione
FROM Ospite
WHERE PrenotazioneStanza = NEW.PrenotazioneStanza;

SELECT PS.NumPosti INTO numero_posti_prenotati
FROM PrenotazioneStanza PS
WHERE PS.IdPrenotazione = NEW.PrenotazioneStanza;

IF (numero_ospiti_prenotazione + 1) > numero_posti_prenotati THEN
SIGNAL SQLSTATE '45003'
SET message_text='I posti della prenotazione sono già esauriti !';
END IF;

END //

DELIMITER ;
```

4. Una persona ospite di un hotel deve usare un servizio dell'hotel in cui è ospitato e in una data compresa nel periodo di permanenza nell'hotel.

```
DELIMITER //
```

```
CREATE TRIGGER usoservizio_before_insert
BEFORE INSERT ON UsoServizio FOR EACH ROW

BEGIN

DECLARE data_inizio DATE;
DECLARE data_fine DATE;
DECLARE prenotazione CHAR(10);
DECLARE done INT DEFAULT FALSE;
DECLARE result INT DEFAULT FALSE;


DECLARE cur CURSOR FOR
SELECT PS.IdPrenotazione
  FROM Ospite O
  JOIN PrenotazioneStanza PS
    ON O.PrenotazioneStanza = PS.IdPrenotazione
 WHERE PS.NomeHotel = NEW.NomeHotel AND
        PS.ViaHotel = NEW.ViaHotel AND
        PS.CittaHotel = NEW.CittaHotel AND
        O.Persona = NEW.Persona;

DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;
OPEN cur;

readLoop : LOOP

FETCH cur INTO prenotazione;

IF DONE THEN
  LEAVE readLoop;
END IF;

SELECT DataInizio, DataFine INTO data_inizio, data_fine
FROM PrenotazioneStanza PS
WHERE PS.IdPrenotazione = prenotazione;

IF( (NEW.Data >= data_inizio AND NEW.Data <= data_fine) ) THEN
  SET result = TRUE;
END IF;

END LOOP;

CLOSE cur;

IF(result = FALSE ) THEN
  SIGNAL SQLSTATE '45004'
  SET message_text= 'La persona non è stata ospite
                    della struttura di cui ha usufruito il servizio';

END IF;

END //
```

```
DELIMITER ;
```

5. Il costo di prenotazione di una camera si ottiene moltiplicando il numero di ospiti

della prenotazione per il prezzo di pernottamento della camera prenotata e il numero di notti soggiornate.

```
DELIMITER //
```

```
CREATE TRIGGER prezzo_fatturaprenotazione_before_insert
BEFORE INSERT ON FatturaPrenotazione FOR EACH ROW

BEGIN

DECLARE prezzo_stanza DECIMAL(6,2);
DECLARE numero_posti_prenotati INT UNSIGNED;
DECLARE numero_notti_soggiornate INT UNSIGNED;
DECLARE importo_soggiorno DECIMAL(8,2);

SET numero_notti_soggiornate =
(SELECT DATEDIFF(DataFine,DataInizio)
 FROM PrenotazioneStanza
 WHERE IdPrenotazione = NEW.PrenotazioneStanza);

SELECT PS.NumPosti INTO numero_posti_prenotati
FROM PrenotazioneStanza PS
WHERE PS.IdPrenotazione = NEW.PrenotazioneStanza;

SELECT S.Prezzo INTO prezzo_stanza
FROM PrenotazioneStanza PS
JOIN Stanza S
ON PS.Stanza = S.IdStanza AND PS.NomeHotel=S.NomeHotel AND
   PS.ViaHotel=S.ViaHotel AND PS.CittaHotel=PS.CittaHotel
WHERE PS.IdPrenotazione = NEW.PrenotazioneStanza;

SET importo_soggiorno =
(prezzo_stanza * numero_posti_prenotati * numero_notti_soggiornate);

IF( NEW.Totale <> importo_soggiorno ) THEN
SET NEW.Totale = importo_soggiorno;
END IF;

END //
```

```
DELIMITER ;
```

3.12 Cursori

Un cursore è una variabile che permette di iterare su un result set sequenzialmente tupla per tupla. In MariaDB i cursori sono disponibili all'interno degli stored programs (stored procedures e stored function) e hanno le proprietà di essere non-scrollable, read-only e asensitive. Non-scrollable significa che le tuple possono essere recuperate solo nell'ordine specificato dall'istruzione SELECT. Le tuple non possono essere saltate, non è possibile saltare a una tupla specifica e non è possibile recuperare le tuple in ordine inverso. Read-only significa che i dati non possono essere aggiornati attraverso il cursore. Asensitive significa che il cursore punta all'attuale dato sottostante. Questo tipo di cursore è più veloce dell'alternativa insensitive, poiché non vengono copiati dati in una tabella temporanea. Tuttavia, le modifiche ai dati sottostanti si ripercuotono sui dati del cursore. Si riportano dei semplici esempi di utilizzo.

1. Dato il codice di un prodotto, si vuole sapere la variazione del prezzo rispetto al suo acquisto precedente.

```
DELIMITER //
```

```
CREATE FUNCTION variazione_prodotto(var_idProdotto VARCHAR(15))
RETURNS DECIMAL(7,2)
DETERMINISTIC
```

```
BEGIN
```

```
    DECLARE done INT DEFAULT FALSE;
```

```
    DECLARE idFattura CHAR(14);
    DECLARE idProdotto VARCHAR(15) DEFAULT var_idProdotto;
    DECLARE dataFattura DATE;
    DECLARE quantitaOrdine INT UNSIGNED;
    DECLARE prezzoOrdine DECIMAL(7,2);
    DECLARE variazionePrezzo DECIMAL(7,2) DEFAULT 0;
    DECLARE numeroTupla INT UNSIGNED DEFAULT 0;
```

```
    DECLARE cur CURSOR FOR
    SELECT P.IdProdotto, FO.IdFattura, FO.Data, O.Quantita, O.Prezzo

    FROM Prodotto P
    JOIN Ordine O
    ON P.IdProdotto = O.Prodotto
    JOIN FatturaOrdine FO
    ON O.Fattura = FO.IdFattura

    WHERE P.IdProdotto = var_idProdotto
    ORDER BY FO.Data DESC
    LIMIT 2;
```

```
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;
    OPEN cur;
```

```
    readLoop : LOOP
```

```
        FETCH cur INTO idProdotto, idFattura,
                        dataFattura, quantitaOrdine, prezzoOrdine;
```

```
        IF DONE THEN
            LEAVE readLoop;
        END IF;
```

```
        SET variazionePrezzo =
            (prezzoOrdine/quantitaOrdine - variazionePrezzo);
        SET numeroTupla = numeroTupla + 1;
```

```
    END LOOP;
```

```
    CLOSE cur;
```

```
    IF numeroTupla < 2 THEN
        signal sqlstate '45001' SET message_text='Record non sufficienti';
    END IF;
```

```
    RETURN variazionePrezzo*(-1);
```

```
END //
```

```
DELIMITER ;
```

2. Cursore per contare il numero di righe di una tabella (davvero semplice ...)

```
DELIMITER //
```

```
CREATE FUNCTION conta_righe()  
RETURNS INT  
DETERMINISTIC  
  
BEGIN  
  
DECLARE done INT DEFAULT FALSE;  
DECLARE numeroTuple INT UNSIGNED DEFAULT 0;  
DECLARE idProdotto VARCHAR(15);  
  
DECLARE cur CURSOR FOR  
  
    SELECT P.IdProdotto  
  
        FROM Prodotto P  
        JOIN Ordine O  
        ON P.IdProdotto = O.Prodotto  
        JOIN FatturaOrdine FO  
        ON O.Fattura = FO.IdFattura  
  
        WHERE P.IdProdotto = '0077970854924'  
        ORDER BY FO.Data DESC;  
  
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;  
    OPEN cur;  
  
    readLoop : LOOP  
  
        FETCH cur INTO idProdotto;  
  
        IF DONE THEN  
            LEAVE readLoop;  
        END IF;  
  
        SET numeroTuple = numeroTuple + 1;  
  
    END LOOP;  
  
    CLOSE cur;  
  
RETURN numeroTuple;  
  
END //
```

```
DELIMITER ;
```

3.13 Funzioni analitiche

A differenza delle funzioni di aggregazione, che calcolano un valore da un gruppo di tuple, le funzioni analitiche sono in grado di restituire più tuple per ogni gruppo. Sicuramente utili per calcolare medie, totali parziali, percentuali o i primi N risultati all'interno di un gruppo. Includo la possibilità di raggruppare tuple in finestre che rendono possibile partizionare i dati. Le finestre sono definite con l'utilizzo della clausola `OVER` in combinazione con la subclausola `PARTITION BY`.

1. Stampare per ogni mese del 2021 la spesa mensile per l'acquisto dei prodotti per il servizio di Spa dell'Hotel Sheraton di Roma e la spesa totale corrente da inizio anno fino al mese corrente incluso.

```
SELECT month(FO.Data) Mese, sum(FO.Totale) Totale,
sum(sum(FO.Totale))
  over (ORDER BY month(FO.Data)
        rows unbounded preceding) "Spesa totale da inizio 2021"
FROM FatturaOrdine FO
JOIN Ordine O
ON FO.IdFattura = O.Fattura
WHERE FO.NomeServizio = "Spa" AND
      FO.NomeHotel = "Hotel Sheraton" AND
      FO.ViaHotel = "Via Marsala 20" AND
      FO.CittaHotel = "Roma" AND
      YEAR(FO.Data) = "2021"
GROUP BY month(FO.Data)
ORDER BY month(FO.Data);
```

2. Stampare per ogni mese del 2021 gli incassi per le prenotazioni delle stanze dell'Hotel Casino di Sanremo e la media dell'incasso calcolata per ogni mese rispetto al mese precedente, al mese corrente e al mese successivo.

```
SELECT month(FP.Data) Mese, sum(FP.Totale) Totale,
round(avg(sum(FP.Totale))
  over (ORDER BY month(FP.Data)
        rows between 1 preceding and 1 following),2) rolling_avg
FROM FatturaPrenotazione FP
JOIN PrenotazioneStanza PS
ON FP.PrenotazioneStanza = PS.IdPrenotazione
WHERE PS.NomeHotel = "Hotel Casino" AND
      PS.ViaHotel = "Corso degli Inglesi 18" AND
      PS.CittaHotel = "Sanremo" AND
      YEAR(FP.Data) = "2021"
GROUP BY month(FP.Data)
ORDER BY month(FP.Data);
```

3. Stampare per l'anno 2021 la Data dell'acquisto mensile, il Nome, la Marca, il Prezzo di una singola unità del prodotto con codice 1197712992488 rispetto agli acquisti effettuati dal servizio del Casinò dell'Hotel Casino di Sanremo, e stampare per ogni mese la variazione in percentuale del prezzo di una singola unità di prodotto rispetto al mese precedente.

```
SELECT FO.Data, P.Nome, P.Marca,
(O.Prezzo/O.Quantita) "Prezzo Prodotto",
round(((O.Prezzo/O.Quantita) - lag((O.Prezzo/O.Quantita),1)
  over (ORDER BY FO.Data)) /
  lag((O.Prezzo/O.Quantita),1)
  over (ORDER BY FO.Data) * 100, 2) "Variazione in percentuale"
FROM FatturaOrdine FO
JOIN Ordine O
ON O.Fattura = FO.IdFattura
JOIN Prodotto P
ON P.IdProdotto = O.Prodotto
WHERE (FO.NomeServizio = "Casino" AND
      FO.NomeHotel = "Hotel Casino" AND
      FO.ViaHotel = "Corso degli Inglesi 18" AND
      FO.CittaHotel = "Sanremo" AND
```

```
P.IdProdotto = "1197712992488" AND
YEAR(FO.Data) = "2021")
ORDER BY FO.Data;
```

3.14 Gestione utenti e privilegi

MariaDB ha la possibilità di gestire i ruoli. Un ruolo è un contenitore di privilegi. Questa novità introdotta nello standard SQL-3 permette di realizzare il principio del minimo privilegio; tramite il ruolo è possibile variare dinamicamente l'insieme di privilegi disponendo in ogni momento l'insieme minimo di privilegi necessari per una certa attività.

HotelStatistics è organizzato gestendo tre ruoli che verranno assegnati agli utenti desiderati. La Tabella 3.2 descrive il ruolo insieme al livello di permessi fornito.

Ruolo	Permessi
SuperUtente	Lettura e scrittura inclusi DROP e ALTER
Amministratore	Lettura e scrittura
Osservatore	Sola lettura

Tabella 3.2: Ruoli e permessi associati in *Hotel Statistics*

Il ruolo *Osservatore* realizza il principio del minimo privilegio per le applicazioni che dovranno solamente effettuare operazioni di lettura sulla base dati. Sicuramente il ruolo *Osservatore* risulterà utile a tutte quelle applicazioni che realizzano grafici statistici che non hanno la necessità di effettuare operazioni di scrittura. Un esempio di grafico potrebbe essere l'istogramma in Figura 3.4 utilizzando la vista 2. Il ruolo Amministratore è invece destinato a quegli utenti che dovranno effettuare le classiche operazioni DML come inserimenti, modifiche e cancellazioni. L'idea è quella di creare utenze amministrative per la gestione e manutenzione della basi dati. Infine, il ruolo di SuperUtente rappresenta quegli utenti pari al creatore; utilizzando ALTER e DROP possono effettuare modifiche radicali alla base dati influenzandone il funzionamento. In seguito i comandi DCL (Data Control Language) per realizzare i ruoli.

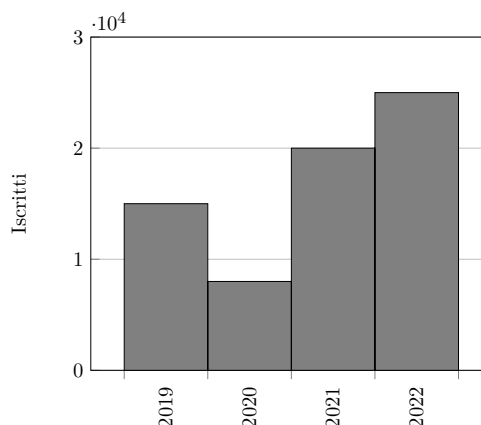


Figura 3.4: Semplice istogramma

```
CREATE ROLE 'SuperUtente', 'Amministratore' , 'Osservatore';
GRANT ALL ON HotelStatistics.* TO 'SuperUtente';
GRANT INSERT, UPDATE, DELETE ON HotelStatistics.* TO 'Amministratore';
GRANT SELECT ON HotelStatistics.* TO 'Osservatore'
```

Una volta creati, questi ruoli possono essere assegnati agli utenti designati. Si fornisce un esempio di creazione utenti con le rispettive assegnazioni di ruoli.

```
CREATE USER 'default_superuser'@'localhost' IDENTIFIED BY 'iamroot';
CREATE USER 'default_admin'@'localhost' IDENTIFIED BY 'iamadmin';
CREATE USER 'default_observer'@'localhost' IDENTIFIED BY 'iamobserver';

GRANT 'SuperUtente' TO 'default_superuser'@'localhost';
GRANT 'Amministratore' TO 'default_admin'@'localhost';
GRANT 'Osservatore' TO 'default_observer'@'localhost';
```

3.15 Algebra relazionale

In questa sezione sono tradotte quattro query SQL nelle loro corrispondenti in algebra relazionale.

1. Stampare nome, cognome e numero di telefono di tutte le persone che sono o sono stati ospiti in una stanza d'hotel.

$$X \leftarrow (\text{Ospite} \bowtie_{\text{Ospite.Persona}=\text{Persona.CF}} \text{Persona})$$

$$\Pi_{(\text{Persona.Nome}, \text{Persona.Cognome}, \text{Persona.Telefono})}(X)$$

2. Stampare per ogni fattura di prenotazione il codice, il costo di prenotazione e la data del pagamento.

$$\rho_{(\text{DataPagamento} \leftarrow \text{Data})} \Pi_{(\text{IdFattura}, \text{Totale}, \text{Data})}(\text{FatturaPrenotazione})$$

3. Stampare le informazioni di tutti gli hotel e dei servizi offerti da ciascun hotel

$$X \leftarrow \left(\text{Hotel} \bowtie_{\substack{\text{Hotel.Nome} = \text{Servizio.NomeHotel} \\ \text{Hotel.Via} = \text{Servizio.ViaHotel} \\ \text{Hotel.Città} = \text{Servizio.CittàHotel}}} \text{Servizio} \right)$$

$$\Pi_{(\text{Hotel.Nome}, \text{Hotel.Via}, \text{Hotel.Città}, \text{Servizio.Nome})}(X)$$

4. Stampare il nome, cognome e telefono di tutte le persone che hanno fatto uso di almeno un servizio, e il nome del servizio da loro utilizzato con la data d'utilizzo.

$$X \leftarrow (\text{UsoServizio} \bowtie_{\text{UsoServizio.Persona}=\text{Persona.CF}} \text{Persona})$$

$$\Pi_{\text{Persona.Nome}, \text{Persona.Cognome}, \text{Persona.Telefono}}(X)$$

$$\Pi_{\text{UsoServizio.NomeServizio}, \text{UsoServizio.Data}}$$

3.16 Calcolo relazionale

In questa sezione sono tradotte quattro query SQL nelle loro corrispondenti in calcolo relazionale su tuple con dichiarazione di range.

1. Stampare nome, cognome e numero di telefono di tutte le persone che sono o sono stati ospiti in una stanza d'hotel.

$$\{ p.Nome, p.Cognome, p.Telefono \mid p(Persona), o(Ospite) \mid o.Persona = p.CF \}$$

2. Stampare per ogni fattura di prenotazione il codice, il costo di prenotazione e la data del pagamento.

$$\{ f.IdFattura, f.Totale, DataPagamento : f.Data \mid f(FatturaPrenotazione) \}$$

3. Stampare le informazioni di tutti gli hotel e dei servizi offerti da ciascun hotel

$$\left\{ h.Nome, h.Via, h.Città, s.Nome \mid h(Hotel), s(Servizio) \mid \begin{array}{l} h.Nome=s.NomeHotel \wedge \\ h.Via=s.ViaHotel \wedge \\ h.Città=s.CittàHotel \end{array} \right\}$$

4. Stampare il nome, cognome e telefono di tutte le persone che hanno fatto uso di almeno un servizio, e il nome del servizio da loro utilizzato con la data d'utilizzo.

$$\left\{ \begin{array}{c} p.Nome, p.Cognome, p.Telefono, \\ u.NomeServizio, u.Data \end{array} \mid u(UsoServizio), p(Persona) \mid u.Persona = p.CF \right\}$$

Progettazione fisica

In questa sezione verranno discusse proprietà relative al DBMS scelto. Tutte le ottimizzazioni e analisi sono comunque compatibili in un ambiente MariaDB o MySQL (notare l'estrema compatibilità di MariaDB nei confronti di MySQL in questa [pagina](#)). Si riportano le cardinalità delle relazioni (il numero dei record) dove verranno effettuate tutte le analisi nella tabella 4.1.

Relazione	Cardinalità
AttivitaManutenzione	140
Convegno	552
Dipendente	260
FatturaOrdina	156
FatturaPrenotazione	67752
Hotel	3
Ordine	5124
Ospite	541960
Persona	279408
PrenotazioneStanza	67752
Prodotto	177
Servizio	13
Societa	655
Stanza	730
UsoServizio	134400

Tabella 4.1: Relazioni con cardinalità

4.1 Storage Engines

Gli Storage engines sono moduli software offerti dal DBMS per gestire dati e indici a livello fisico. Realizzano lo schema interno (detto anche schema fisico) di un DBMS. Per il progetto *HotelStatistics* è utilizzato *InnoDB*. Questo è un buon storage engine transazionale di uso generale. È la scelta di default a partire da MariaDB 10.2. In questa sede, non si può affrontare l'intero tuning di MariaDB. Si è deciso di non apportare modifiche all'implementazione offerta di default, ma si vuole comunque discutere di possibili ottimizzazioni prendendo spunto da questa [pagina](#).

Tutte le configurazioni supplementari sono da apportare nel file *mariadb.conf* o nel file *my.cnf* (in realtà MariaDB legge configurazioni da diversi file **.conf*). Si nota comunque la possibilità di impostare configurazioni aggiuntive anche al momento dell'avvio dell'istanza direttamente come parametri oppure dalla console in esecuzione. Per performance ottimali, bisogna agire sullo storage engine. Per InnoDB, le variabili significative da modificare sono le seguenti:

- `innodb_buffer_pool_size` Specifica la dimensione in bytes del buffer pool; questa area di memoria principale è utilizzata per memorizzare dati e indici. L'idea è quella di massimizzare le sue dimensioni in modo da ridurre gli accessi alla memoria secondaria. È pratica comune e consigliato dalla documentazione impostare questa variabile con un valore pari al 80% della memoria disponibile, ma ovviamente dipende dalle risorse del sistema e dalla dimensione del database. Il valore di default è 134217728 bytes (128 MiB).
- `innodb_log_file_size` Il redo log è una struttura dati residente in memoria secondaria. È utilizzata per il ripristino del sistema e per correggere dati scritti da transazioni incomplete. Maggiore è la dimensione di questa variabile e meno saranno gli accessi sul disco dovuti ad una minore attività di flushing (scrittura fisica di dati residenti memoria principale nella memoria secondaria) dei checkpoint, ma sarà anche maggiore il tempo di recupero dovuto ad un arresto anomalo.
- `innodb_flush_method` Il flushing è quel processo che determina la scrittura su memoria secondaria dei dati residenti in memoria principale. Questa variabile descrive quale metodo utilizzare per realizzare il processo. I metodi sono *fdatsync*, *fsync*, *O_DIRECT*, *O_DSYNC*, *O_DIRECT_NO_FSYNC*, *ALL_O_DIRECT*, *littlesync*, *nosync*. La scelta dipende dal filesystem sottostante, dalla versione di MariaDB e una scelta sbagliata incide pesantemente sulle prestazioni. Di solito si imposta il valore *O_DIRECT*.
- `innodb_thread_sleep_delay` Per elaborare le transazioni, InnoDB dipende dai thread del sistema operativo. Se il numero di thread in esecuzione raggiunge una certa soglia, in nuovi thread vengono posti in una coda di attesa; `innodb_thread_sleep_delay` specifica quanti microsecondi i thread devono rimanere in questa coda. Si nota come questa variabile sia deprecata e ignorata a partire da MariaDB 10.5.5
- `innodb_thread_concurrency` Viene utilizzata per limitare il numero di thread. Si nota come questa sia deprecata e ignorata a partire da MariaDB 10.5.5

4.2 Indici e performance

In questa sezione vedremo come l'utilizzo degli indici migliori in alcuni casi i tempi di risposta delle query coinvolte. Tra i vari criteri di performance la misura scelta è il tempo. Questo non può essere misurato in maniera puntuale ma deve essere fatto su esperimenti ripetuti. Verrà così utilizzato [mysqslap](#); è uno strumento ufficiale per analizzare i tempi di risposta. Permette di simulare connessioni concorrenti e di eseguire la stessa query più volte. Al fine di una giusta analisi viene utilizzata `SQL_NO_CACHE`. Questa hint permette di evitare l'utilizzo della cache da parte del DBMS.

Tutti i relativi test sono eseguiti in un ambiente virtualizzato tramite la suite [QEMU/KVM](#). Nella tabella 4.2 e 4.3 è riportato l'ambiente di esecuzione *guest* insieme alle caratteristiche del sistema *host*.

Tipo	OS	DMBS	vCPU	RAM/SWAP	Disco
Guest	Debian 10	MariaDB 10.3.34	SB 2 core	2/4 GB	SSD 20 GB

Tabella 4.2: Dettagli sull'ambiente di esecuzione

Tipo	OS	Hypervisor	CPU	RAM	Disco
Host	Debian 10	QEMU/KVM	i5-2450M 2.50GHz	8 GB	SSD 256 GB

Tabella 4.3: Dettagli sulla macchina host

Il tuning delle query prese in considerazione segue i seguenti passi :

1. Scrivere istruzioni sql *semplici*
2. Verificare i piani di esecuzione
3. Ottimizzare l'accesso alla singola tabella
4. Ottimizzare l'ordine dei passi
5. Riscrivere la query

Infine si riporta il template del comando *mysqlslap*

```
mysqlslap --no-defaults
          --user=root
          --password
          --delimiter=";"
          --create-schema=HotelStatistics
          --no-drop
          --query=query.sql
          --concurrency=X
          --iterations=Y
```

dove :

- query=query.sql specifica il file che contiene la query in esame
- concurrency=X è il numero di client che eseguono la stessa query
- iterations=Y è il numero di volte che si esegue il test

Per ogni query, il test è ripetuto 10 volte (impostando --iterations=10) e i confronti sono realizzati nel loro caso peggiore (il tempo massimo). Analizziamo ora la seguente query :

```
SELECT SQL_NO_CACHE S.IdStanza
FROM Stanza S
WHERE S.NomeHotel = "Hotel Overlook" AND
S.ViaHotel = "Via Cavour 333" AND
S.CittaHotel = "Torino" AND
S.IdStanza NOT IN (SELECT PS.Stanza
                   FROM PrenotazioneStanza PS
                   WHERE PS.NomeHotel = "Hotel Overlook" AND
                   PS.ViaHotel = "Via Cavour 333" AND
                   PS.CittaHotel = "Torino" AND
                   ((PS.DataInizio BETWEEN "2021-10-08" AND "2021-10-13") OR
                   (PS.DataFine BETWEEN "2021-10-08" AND "2021-10-13")));
```

Mostriamo le informazioni relative al piano di esecuzione tramite il comando EXPLAIN.

```
***** 1. row *****
      id: 1
    select_type: PRIMARY
      table: S
      type: ref
possible_keys: NomeHotel
      key: NomeHotel
    key_len: 526
      ref: const,const,const
      rows: 242
    Extra: Using where; Using index
***** 2. row *****
      id: 2
    select_type: MATERIALIZED
      table: PS
      type: ALL
possible_keys: Stanza
      key: NULL
    key_len: NULL
      ref: NULL
      rows: 67325
    Extra: Using where
```

I tempi di risposta tramite mysqlslap:

```
Benchmark
  Average number of seconds to run all queries: 0.038 seconds
  Minimum number of seconds to run all queries: 0.036 seconds
  Maximum number of seconds to run all queries: 0.050 seconds
  Number of clients running queries: 1
  Average number of queries per client: 1
```

È possibile migliorare il tempo di esecuzione definendo due indici `idx_datainizio` e `idx_datafine` che permettono di ottimizzare l'accesso alla tabella Prenotazione-Stanza passando da un metodo di accesso `type:ALL` (full table scan) a un tipo di accesso `index_merge`. I comandi SQL sono i seguenti

```
CREATE INDEX idx_datainizio ON PrenotazioneStanza(DataInizio);
CREATE INDEX idx_datafine ON PrenotazioneStanza(DataFine);
```

Il relativo comando EXPLAIN

```
***** 1. row *****
      id: 1
    select_type: PRIMARY
      table: S
        type: ref
possible_keys: NomeHotel
         key: NomeHotel
      key_len: 526
         ref: const,const,const
        rows: 242
      Extra: Using where; Using index
***** 2. row *****
      id: 2
    select_type: MATERIALIZED
      table: PS
        type: index_merge
possible_keys: Stanza,idx_datainizio,idx_datafine
         key: idx_datainizio,idx_datafine
      key_len: 4,4
         ref: NULL
        rows: 2290
      Extra: Using sort_union(idx_datainizio,idx_datafine); Using where
```

Utilizziamo ora mysqlslap per analizzare i tempi di risposta :

```
Benchmark
  Average number of seconds to run all queries: 0.007 seconds
  Minimum number of seconds to run all queries: 0.007 seconds
  Maximum number of seconds to run all queries: 0.019 seconds
  Number of clients running queries: 1
  Average number of queries per client: 1
```

Confrontando i risultati nel loro tempo massimo, la versione ottimizzata impiega solamente il 38% del tempo della versione senza ottimizzazione. C'è quindi un miglioramento del **62%**.

Vediamo come l'utilizzo degli indici influisca i tempi di risposta in un ambiente concorrente dove diversi client eseguono la stessa query contemporaneamente (grazie al parametro `--concurrency=X`). Nelle tabelle 4.4 e 4.5 vengono riportati i tempi di

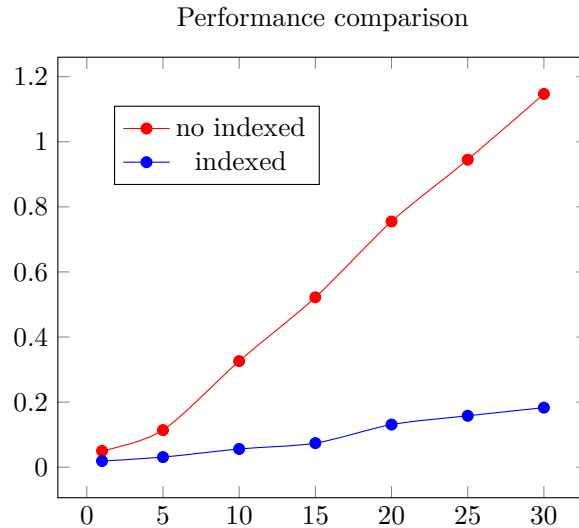
esecuzione concorrenti; i valori indicano rispettivamente il numero di client, il tempo medio, minimo e massimo per soddisfare tutti i client ripetendo l'esecuzione 10 volte.

Clients	AVG	MIN	MAX
1	0.038 s	0.036 s	0.050 s
5	0.103 s	0.100 s	0.114 s
10	0.211 s	0.199 s	0.329 s
15	0.325 s	0.295 s	0.522 s
20	0.537 s	0.462 s	0.755 s
25	0.680 s	0.627 s	0.945 s
30	0.704 s	0.624 s	1.47 s

Tabella 4.4: senza ottimizzazione

Clients	AVG	MIN	MAX
1	0.007 s	0.007 s	0.019 s
5	0.021 s	0.018 s	0.031 s
10	0.039 s	0.036 s	0.056 s
15	0.057 s	0.054 s	0.074 s
20	0.077 s	0.072 s	0.131 s
25	0.096 s	0.091 s	0.158 s
30	0.121 s	0.114 s	0.183 s

Tabella 4.5: con ottimizzazione



dove il grafico è costruito utilizzando le tabelle 4.4 e 4.5 nei loro casi peggiori (colonna MAX). Da questi risultati, si nota come il tempo di esecuzione non aumenti linearmente in funzione del numero dei client (grazie al gestore della concorrenza); la versione senza ottimizzazione con 30 clients concorrenti ha impiegato un tempo massimo di 1.47 s per soddisfare tutte le richieste. Quindi, nel caso peggiore, una richiesta è stata soddisfatta in $1.47/30 = 0.049$ s. Molto simile al tempo di una singola esecuzione senza ottimizzazione nel suo caso peggiore. Da questo si evince che per notare ritardi nelle risposte da parte del DBMS, bisogna aumentare il numero di client concorrenti. L'esperimento si ritiene comunque soddisfacente; questa implementazione di *HotelStatistics* insieme alla configurazione di MariaDB, riesce a gestire 30 client senza nessun problema.

Le successive query sono analizzate in modo meno verboso. Analizziamo ora:

```

SELECT SQL_NO_CACHE C.Nome as Convegno, C.Data as Data, C.NomeHotel as Hotel,
                S.Nome as NomeSala, count(*) as NumeroOspiti
FROM Convegno C, Stanza S, PrenotazioneStanza PS, Ospite O
WHERE C.Stanza = S.IdStanza AND C.NomeHotel = S.NomeHotel AND
      C.ViaHotel = S.ViaHotel AND C.CittaHotel = S.CittaHotel AND
      PS.Stanza = S.IdStanza AND PS.NomeHotel = S.NomeHotel AND
      PS.ViaHotel = S.ViaHotel AND PS.CittaHotel = S.CittaHotel AND
      PS.Stanza = S.IdStanza AND PS.NomeHotel = S.NomeHotel AND
      PS.ViaHotel = S.ViaHotel AND PS.CittaHotel = S.CittaHotel AND
      PS.IdPrenotazione = O.PrenotazioneStanza AND C.Data = PS.DataInizio
GROUP BY C.Nome, C.Data;

```

Il relativo comando EXPLAIN

```

***** 1. row *****
      id: 1
    select_type: SIMPLE
      table: C
        type: index
possible_keys: PRIMARY, Stanza
         key: Stanza
        key_len: 534
         ref: NULL
        rows: 552
      Extra: Using where; Using index; Using temporary; Using filesort
***** 2. row *****
      id: 1
    select_type: SIMPLE
      table: S
        type: eq_ref
possible_keys: PRIMARY, NomeHotel
         key: PRIMARY
        key_len: 530
         ref: HotelStatistics.C.Stanza, HotelStatistics.C.NomeHotel,
              HotelStatistics.C.ViaHotel, HotelStatistics.C.CittaHotel
        rows: 1
      Extra:
***** 3. row *****
      id: 1
    select_type: SIMPLE
      table: PS
        type: ref
possible_keys: PRIMARY, Stanza
         key: Stanza
        key_len: 534
         ref: HotelStatistics.C.Stanza, HotelStatistics.C.NomeHotel,
              HotelStatistics.C.ViaHotel, HotelStatistics.C.CittaHotel
        rows: 48
      Extra: Using where
***** 4. row *****
      id: 1
    select_type: SIMPLE
      table: O
        type: ref
possible_keys: PRIMARY
         key: PRIMARY
        key_len: 40
         ref: HotelStatistics.PS.IdPrenotazione
        rows: 3
      Extra: Using index

```

Tale query contiene una clausola GROUP BY che richiede un sort (che viene realizzato in un file temporaneo). Una possibile ottimizzazione consiste nel semplificare l'operazione di sort creando un indice multicolonna sui campi C.Nome e C.Data che vengono usati per l'aggregazione.

```
CREATE INDEX idx_groupby ON Convegno (Nome,Data);
```

Il relativo comando EXPLAIN :

```
***** 1. row *****
      id: 1
    select_type: SIMPLE
      table: C
        type: index
possible_keys: PRIMARY,Stanza
         key: idx_groupby
      key_len: 405
         ref: NULL
        rows: 552
      Extra: Using where
***** 2. row *****
      id: 1
    select_type: SIMPLE
      table: S
        type: eq_ref
possible_keys: PRIMARY,NomeHotel
         key: PRIMARY
      key_len: 530
         ref: HotelStatistics.C.Stanza,HotelStatistics.C.NomeHotel,
              HotelStatistics.C.ViaHotel,HotelStatistics.C.CittaHotel
        rows: 1
      Extra:
***** 3. row *****
      id: 1
    select_type: SIMPLE
      table: PS
        type: ref
possible_keys: PRIMARY,Stanza
         key: Stanza
      key_len: 534
         ref: HotelStatistics.C.Stanza,HotelStatistics.C.NomeHotel,
              HotelStatistics.C.ViaHotel,HotelStatistics.C.CittaHotel
        rows: 48
      Extra: Using where
***** 4. row *****
      id: 1
    select_type: SIMPLE
      table: O
        type: ref
possible_keys: PRIMARY
         key: PRIMARY
      key_len: 40
         ref: HotelStatistics.PS.IdPrenotazione
        rows: 3
      Extra: Using index
```

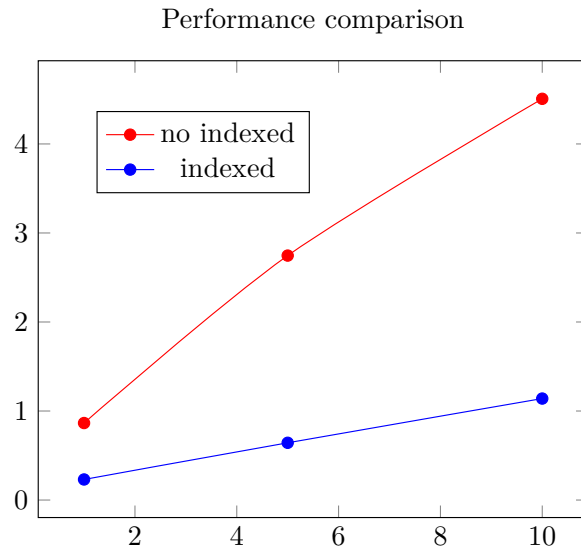
Utilizziamo ora mysqlslap per analizzare i tempi di risposta:

Clients	AVG	MIN	MAX
1	0.820 s	0.805 s	0.865 s
5	2.287 s	2.147 s	2.746 s
10	4.337 s	4.273 s	4.507 s

Tabella 4.6: senza ottimizzazione

Clients	AVG	MIN	MAX
1	0.210 s	0.200 s	0.231 s
5	0.563 s	0.552 s	0.643 s
10	1.110 s	1.100 s	1.140 s

Tabella 4.7: con ottimizzazione



dove il grafico è costruito utilizzando le tabelle 4.6 e 4.7 nei loro casi peggiori (colonna MAX). Prendendo come riferimento la prima riga di entrambe le tabelle (il caso con un solo client) nel loro caso peggiore, la versione ottimizzata impiega solamente il 26.7% del tempo della versione senza ottimizzazione. C'è quindi un miglioramento del **73.3%**.

Analizziamo ora questa semplice query:

```
SELECT SQL_NO_CACHE DISTINCT p.Nome, p.Cognome, p.Telefono
FROM Ospite o, Persona p
WHERE o.Persona = p.CF;
```

Il relativo comando EXPLAIN :

```
***** 1. row *****
      id: 1
    select_type: SIMPLE
      table: p
      type: ALL
possible_keys: PRIMARY
      key: NULL
    key_len: NULL
      ref: NULL
     rows: 277969
    Extra: Using temporary
***** 2. row *****
      id: 1
    select_type: SIMPLE
      table: o
      type: ref
possible_keys: Persona
      key: Persona
    key_len: 64
      ref: HotelStatistics.p.CF
     rows: 1
    Extra: Using index; Distinct
```

L'utilizzo di Using temporary e Distinct incidono pesantemente sulle prestazioni. Scriviamo la stessa query in maniera alternativa evitando la Distinct:

```
SELECT SQL_NO_CACHE p.Nome, p.Cognome, p.Telefono
FROM Persona p
WHERE p.CF IN (SELECT Ospite.Persona FROM Ospite);
```

Il relativo comando EXPLAIN :

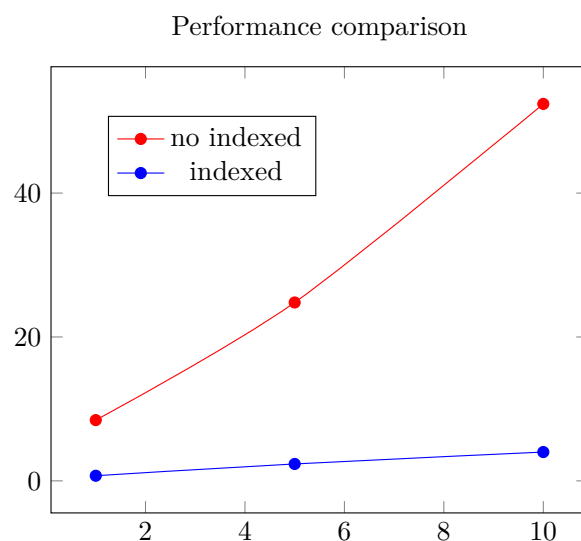
```
***** 1. row *****
      id: 1
    select_type: PRIMARY
      table: p
      type: ALL
possible_keys: PRIMARY
      key: NULL
    key_len: NULL
      ref: NULL
     rows: 277969
    Extra:
***** 2. row *****
      id: 1
    select_type: PRIMARY
      table: Ospite
      type: ref
possible_keys: Persona
      key: Persona
    key_len: 64
      ref: HotelStatistics.p.CF
     rows: 1
    Extra: Using index; FirstMatch(p)
```

Confrontiamo i tempi di risposta con mysqlslap:

Clients	AVG	MIN	MAX	Clients	AVG	MIN	MAX
1	8.383 s	8.333 s	8.447 s	1	0.704 s	0.697 s	0.714 s
5	24.198 s	23.718 s	24.794 s	5	2.269 s	2.222 s	2.353 s
10	50.529 s	49.013 s	52.393 s	10	3.910 s	3.823 s	4.006 s

Tabella 4.8: senza ottimizzazione

Tabella 4.9: con ottimizzazione



dove il grafico è costruito utilizzando le tabelle 4.8 e 4.9 nei loro casi peggiori (colonna MAX). Prendendo come riferimento la prima riga di entrambe le tabelle (il caso con un solo client) nel loro caso peggiore, la versione ottimizzata impiega solamente il 8.5% del tempo della versione senza ottimizzazione. C'è quindi un miglioramento del **91.5%**.

Con questa ultima analisi vogliamo mostrare una tecnica interessante che permette di evitare il join (solo in casi particolari). Analizziamo la seguente query:

```
SELECT SQL_NO_CACHE P.*
FROM Ospite O JOIN Persona P ON O.Persona = P.CF
GROUP BY P.CF
HAVING count(*) > 2;
```

Vogliamo capire le persone che sono state ospiti più di 2 volte nell'intera catena alberghiera. Riportiamo il comando EXPLAIN:

```
***** 1. row *****
      id: 1
    select_type: SIMPLE
      table: P
        type: index
possible_keys: PRIMARY
```

```

        key: PRIMARY
    key_len: 64
        ref: NULL
    rows: 277969
    Extra:
***** 2. row *****
        id: 1
    select_type: SIMPLE
        table: O
        type: ref
possible_keys: Persona
        key: Persona
    key_len: 64
        ref: HotelStatistics.P.CF
    rows: 1
    Extra: Using index

```

Bisogna realizzare il join. Il DMBS accede ad entrambe le tabelle utilizzando gli indici in presenti (in particolare, per la relazione Ospite utilizza direttamente i valori presenti nell'indice). Notiamo come la selettività per la relazione Persona sia molto debole; tutte le tuple sono coinvolte nel join e la relazione Persona viene acceduta tramite indice. Ricordiamo infine anche la presenza del GROUP BY: Riscriviamo la query nel seguente modo:

```

SELECT SQL_NO_CACHE P.*
FROM Persona P
WHERE P.CF IN
    (SELECT O.Persona
     FROM Ospite O
     GROUP BY O.Persona
     HAVING count(*) > 2);

```

Siccome le persone che sono ospiti almeno 3 volte sono probabilmente molte meno del totale, l'idea è quella di utilizzare la [Semi-join Materialization Strategy](#); si memorizza il result set della subquery in memoria (in principale, se disponibile). Si può utilizzare quando la query nidificata non è correlata con la più esterna. La materialization permette di eseguire la subquery solamente una volta. Riportiamo il comando EXPLAIN:

```

***** 1. row *****
        id: 1
    select_type: PRIMARY
        table: P
        type: ALL
possible_keys: PRIMARY
        key: NULL
    key_len: NULL
        ref: NULL
    rows: 277969
    Extra:
***** 2. row *****
        id: 1
    select_type: PRIMARY
        table: <subquery2>
        type: eq_ref
possible_keys: distinct_key
        key: distinct_key
    key_len: 64

```

```

        ref: HotelStatistics.P.CF
        rows: 1
        Extra:
***** 3. row *****
        id: 2
        select_type: MATERIALIZED
        table: 0
        type: index
possible_keys: NULL
        key: Persona
        key_len: 64
        ref: NULL
        rows: 47712
        Extra: Using index

```

In questa query ci sono 2 `SELECT` (infatti abbiamo, `id=1` e `id=2`). La prima riga ci dice che la relazione `Persona` è acceduta in full table scan. La terza riga riguarda l'esecuzione della `SELECT` interna, quella con `id=2`. Il tipo `MATERIALIZED` significa che la `SELECT` interna verrà eseguita e il suo risultato memorizzato in una tabella temporanea (in memoria principale, se possibile) con una chiave che previene la presenza di tuple duplicate. Inoltre la presenza di `type:index` e `Extra: Using index` ci dice che la `SELECT` interna è realizzata utilizzando esclusivamente i valori presenti dell'indice sull'attributo `Persona` della relazione `Ospite`. La seconda riga ci dice che la tabella materializzata, la quale verrà utilizzata dalla `SELECT` esterna è acceduta utilizzando l'indice `distinct_key` creato appositamente come spiegato precedentemente. Confrontiamo i tempi con una semplice esecuzione, otteniamo:

Senza Ottimizzazione	Ottimizzata
1.119	0.860 s

Tabella 4.10: Tempo di una sola esecuzione

La versione con la strategia semi-join materialization impiega il 76.9 % del tempo di quella senza ottimizzazione. Si ha così un guadagno del **23.1** %.

4.3 Deploy e scripting

Per poter implementare *HotelStatistics* bisogna poter caricare i relativi file **.sql*. È possibile caricare tutti i file manualmente con il classico comando `SOURCE` offerto da MariaDB. Si vuole evitare questo e automatizzare la cosa. Lo script *install.sh* scritto in [bash](#) (per quanto vale, noi riteniamo utile un DBMS solamente in ambienti [unix-like](#)) permette di caricare l'intero progetto. In seguito si presenta una versione semplificata di questo script. La versione completa si trova presso questa [pagina](#).

```
#!/bin/bash

echo -ne "Import script to load all SQL files \n\n"
read -p 'Username : ' uservar
read -sp 'Password : ' passvar
echo -ne "\n\n"

mysql --user=$uservar --password=$passvar < schema.sql

if [ $? != 0 ]
then
    echo -ne "\n Schema import FAILED\n\n"
    exit 1
fi

mysql --user=$uservar --password=$passvar < views.sql

if [ $? != 0 ]
then
    echo -ne "\n Views import FAILED\n\n"
    exit 1
fi

mysql --user=$uservar --password=$passvar < procedures.sql

if [ $? != 0 ]
then
    echo -ne "\n Stored procedures import FAILED\n\n"
    exit 1
fi

mysql --user=$uservar --password=$passvar < cursors.sql

if [ $? != 0 ]
then
    echo -ne "\n Cursors import FAILED\n\n"
    exit 1
fi

mysql --user=$uservar --password=$passvar < triggers.sql

if [ $? != 0 ]
then
    echo -ne "\n Triggers import FAILED\n\n"
    exit 1
fi

mysql --user=$uservar --password=$passvar < data.sql

if [ $? != 0 ]
then
    echo -ne "\n Data import FAILED\n\n"
    exit 1
fi

echo -ne "Import COMPLETED\n\n"
```

Un diverso modello dei dati

5.1 MongoDB

MongoDB è un database NoSQL orientato ai documenti; invece di utilizzare tabelle e righe come nei database relazionali, MongoDB fa uso di collezioni e documenti. Quest'ultimi sono costituiti da coppie chiave-valore che sono l'unità base di memorizzazione. Le collezioni contengono insiemi di documenti e mettono a disposizione metodi per l'elaborazione di questi.

In questa sezione parte di *HotelStatistics* è proposta in MongoDB. Definiamo una vista come il join tra Persona e Ospite e successivamente esportiamo il result set in un file *json*:

```
CREATE VIEW persona_xmongo (CF, Nome, Cognome, Mail,
                           Telefono, DataNascita, PrenotazioneStanza) AS
SELECT CF, Nome, Cognome, Mail,
       Telefono, DataNascita, PrenotazioneStanza
FROM Persona P
JOIN Ospite O
ON P.CF = O.Persona;

SELECT JSON_OBJECT("cf", CF, "nome", Nome, "cognome",
                  Cognome, "mail", Mail, "tel",
                  Telefono, "data_nascita", DataNascita,
                  "id_prenotazione", PrenotazioneStanza)
INTO OUTFILE 'path/persona.json' FROM persona_xmongo;
```

Alternativamente possiamo anche non definire una vista:

```
SELECT JSON_OBJECT("_id", IdPrenotazione, "data_inizio", DataInizio,
                  "data_fine", DataFine, "num_posti", NumPosti,
                  "stanza", Stanza, "nome_hotel", NomeHotel,
                  "via_hotel", ViaHotel, "citta_hotel", CittaHotel,
                  "societa", Societa, "persona", Persona, "fattura",
                  IdFattura, "totale", Totale, "data_fattura", Data,
                  "tipo_pagamento", TipoPagamento)
INTO OUTFILE 'path/prenotazioni.json'
FROM PrenotazioneStanza PS JOIN FatturaPrenotazione FP
ON PS.IdPrenotazione = FP.PrenotazioneStanza;
```

```
SELECT JSON_OBJECT("nome", H.Nome, "via", Via, "citta", Citta, "stelle",
                  Stelle, "tel", Telefono, "stanza", IdStanza, "prezzo",
                  Prezzo, "num_posti", NumPosti, "nome_sala", S.Nome)
INTO OUTFILE 'path/hotel.json'
```

```
FROM Hotel H JOIN Stanza S ON H.Nome = S.NomeHotel AND
                           H.Via = S.ViaHotel AND
                           H.Citta = S.CittaHotel;
```

Possiamo importare il file **.json* ottenuti eseguendo :

```
mongoimport --db HotelStatistics
            --collection persona
            --file persona.json

mongoimport --db HotelStatistics
            --collection prenotazioni
            --file prenotazioni.json

mongoimport --db HotelStatistics
            --collection hotel
            --file hotel.json
```

In MongoDB sono ora presenti tre collections *persona*, *prenotazioni* e *hotel* dove :

- *persona* contiene i documenti corrispondenti alle tuple del result set associato al join tra *Persona* e *Ospite*
- *prenotazioni* contiene i documenti corrispondenti alle tuple del result set associato al join tra *PrenotazioneStanza* e *FatturaPrenotazione*
- *hotel* contiene i documenti corrispondenti alle tuple del result set associato al join tra *Hotel* e *Stanza*

Mostriamo esempi di update notando come in realtà tali update servono per convertire alcuni campi stringa dei file json in formato data. Tale risultato potrebbe essere ottenuto con l'opzione *--columnsHaveTypes* dell'utilità *mongoimport* nell'import di file csv

```
db.prenotazioni.updateMany( { },
[ {
  $set: {data_inizio:{$dateFromString: {dateString: "$data_inizio"}}}
}])

db.prenotazioni.updateMany( { },
[ {
  $set: {data_fine:{$dateFromString:{ dateString: "$data_fine"}}}
}])

db.prenotazioni.updateMany( { },
[ {
  $set: {data_fattura:{$dateFromString:{dateString:"$data_fattura"}}}
}])
```

5.2 Query con MongoDB

1. Stampare nome, cognome e numero di telefono di tutte le persone che sono o sono stati ospiti in una stanza d'hotel.

```
db.persona.find({}, {_id:0,nome:1,cognome:1,tel:1})
```

4. Data la stanza N.186 dell'Hotel Overlook trovare (in ordine di DataInizio) se esistono le possibili prenotazioni con periodo di permanenza sovrapponibile al periodo con DataInizio 2021-10-08 e DataFine 2021-10-13.

```
db.prenotazioni.find( {
  $and: [
    {"stanza": 186},
    {"nome_hotel": "Hotel Overlook"},
    {"via_hotel": "Via Cavour 333"},
    {"citta_hotel": "Torino"},
    { $or: [
      {"data_inizio": {$gte: new Date("2021-10-08"),
                        $lte: new Date("2021-10-13")}},
      {"data_fine": {$gte: new Date("2021-10-08"),
                     $lte: new Date("2021-10-13")}}]}
  ] } ).sort({"data_inizio": 1})
```

5. Dato l'Hotel Overlook e il periodo di permanenza DataInizio = 2022-01-08 e DataFine = 2022-01-13 e trovare tutte le stanze non prenotate per quel periodo nell'hotel dato. Il `.toArray().map(function(ele){return ele.stanza})` ha lo scopo di associare alla variabile `stanze_occupate` l'array con tutti i numeri delle stanze occupate nell'hotel e nel periodo indicato.

```
var stanze_occupate = db.prenotazioni.find( {
  $and: [
    {"nome_hotel": "Hotel Overlook"},
    {"via_hotel": "Via Cavour 333"},
    {"citta_hotel": "Torino"},
    { $or: [
      {"data_inizio": {$gte: new Date("2021-10-08"),
                        $lte: new Date("2021-10-13")}},
      {"data_fine": {$gte: new Date("2021-10-08"),
                     $lte: new Date("2021-10-13")}}]}
  ] }, {_id:0,stanza:1}).toArray().map(function(ele){return ele.stanza})

db.hotel.find({ $and: [
  {"nome": "Hotel Overlook"},
  {"via": "Via Cavour 333"},
  {"citta": "Torino"},
  {"stanza": {$nin: stanze_occupate}}], {_id:0,stanza:1})
```

6. Dato l'anno 2021, stampare per ogni mese la somma degli importi pagati dai clienti in quel mese e il numero di fatture emesse.

```
db.prenotazioni.aggregate(
{ $match: { data_fattura: { $gte: ISODate("2021-01-01T00:00:00.0Z"),
  $lt: ISODate("2021-12-31T00:00:00.0Z") } } },
{ $group: { _id: { mese: { $month: "$data_fattura" } },
  numero_fatture: { $sum: 1 }, totale: { $sum: "$totale" } } },
{$sort: {_id:1}})
```

14. Stampare codice fiscale, nome, cognome, numero di telefono di tutte le persone che sono stati ospiti della catena alberghiera più di due volte.


```

db.persona.aggregate([
  { $group: {
    _id : "$cf",
    nome : { $first: "$nome"},
    cognome : { $first: "$cognome"},
    mail : { $first: "$mail"},
    tel : { $first: "$tel"},
    numero_soggiorni : { $sum: 1 } },
  { $match: { numero_soggiorni : { $gt: 2 } } }
])

```

In questa query la group key è il document-field cf. I documenti ottenuti dall'aggregazione presentano anche i campi nome, cognome, etc ... con i relativi valori. Tale risultato è possibile scrivendo nome:{\$first: "\$nome"},...

15. Dato l'anno 2021 e l'Hotel Casino, Corso degli Inglesi 18, Sanremo stampare per ogni mese il numero di prenotazioni ricevute e la somma degli importi pagati per le prenotazioni in quel mese. Nota: In aggiunta ordiniamo i documenti in ordine crescente di mese.

```

db.prenotazioni.aggregate(
{$match:{$and:[{data_fattura:{gte: ISODate("2021-01-01T00:00:00.0Z"),
                    $lt: ISODate("2021-12-31T00:00:00.0Z")}},
{"nome_hotel": "Hotel Casino"},
{"via_hotel": "Corso degli Inglesi 18"},
{"citta_hotel": "Sanremo"}]}},
{ $group: { _id: { mese: { $month: "$data_fattura" },
                    numero_fatture: { $sum: 1 }, totale: { $sum: "$totale" } } },
{$sort: { "_id.mese": 1 } })

```

In alternativa definendo opportune variabili si può riscrivere l'aggregation in maniera più compatta.

```

var yearAndHotel = { $and:[{
  data_fattura: { $gte: ISODate("2021-01-01T00:00:00.0Z"),
                  $lt: ISODate("2021-12-31T00:00:00.0Z") }},
{"nome_hotel": "Hotel Casino"},
{"via_hotel": "Corso degli Inglesi 18"},
{"citta_hotel": "Sanremo"}]}

var reservationsAndProfits = { _id: { mese: { $month: "$data_fattura" },
                                       numero_fatture: { $sum: 1 },
                                       totale: { $sum: "$totale" } } }

db.prenotazioni.aggregate(
{ $match: yearAndHotel },
{ $group: reservationsAndProfits },
{$sort: { "_id.mese": 1 } })

```

È possibile simulare l'operazione di join in MongoDB. Tuttavia tale operazione risulta avere tempi di esecuzioni inefficienti, specialmente se confrontati con quelli di un RDBMS. La seguente query simula il join tra le collection prenotazioni e persona, come se esse fossero delle tabelle in un RDBMS. È all'incirca equivalente alla seguente query, che ha un query response time di 3,90 sec:

```

SELECT *
FROM Ospite o
JOIN Persona p
  ON o.Persona = p.CF
JOIN PrenotazioneStanza ps
  ON ps.IdPrenotazione = o.PrenotazioneStanza
JOIN FatturaPrenotazione fp
  ON fp.PrenotazioneStanza = ps.IdPrenotazione;

```

la stessa query in MongoDB

```

db.prenotazioni.aggregate([
{$lookup: {from: "persona",
           localField: "_id", foreignField: "id_prenotazione",
           as: "ospite"}},
{$unwind: "$ospite"} ])

```

Una stima molto approssimata del query response time può essere ottenuta nel modo seguente. (La variabile risultato è di poco sopra i 10 sec)

```

var inizio = new Date()

db.prenotazioni.aggregate([
{$lookup: {from: "persona",
           localField: "_id", foreignField: "id_prenotazione",
           as: "ospite"}},
{$unwind: "$ospite"} ])

var fine = new Date()

var risultato = fine - inizio
risultato

```

In alternativa da `db.<collection>.<query>.explain('executionStats')` è possibile considerare l'`executionTimeMillis` value.

Il sistema software

In questa si introduce una possibile architettura software in grado di realizzare l'intero progetto. Si evidenzia come questa sezione esuli gli scopi della consegna da parte della professoressa Loredana Vigliano. L'idea è quella di mostrare come solamente l'utilizzo di software open source possa esaudire tutte le esigenze.

Un possibile sistema software per realizzare il progetto può avvalersi delle seguenti tecnologie:

OS	DBMS	Backend	Frontend
Debian GNU/Linux	MariaDB	Apache/PHP	HTML/CSS/JS

Tabella 5.1: Componenti software necessarie

Questo evidenzia come *HotelStatistics* possa essere incluso in un classico sistema LAMP. Il frontend rappresenta il client che eseguirà attraverso opportune maschere il caricamento la visualizzazione dei dati richiesti. Il backend è il lato server del software che prenderà in carico le richieste (con eventualmente i dati) ed eseguirà la parte DQL o DML richiesta (anche attraverso le stored procedure) sulla base dati *HotelStatistics*. Si riporta infine un semplice deployment diagram che descrive ad alto livello il sistema software:

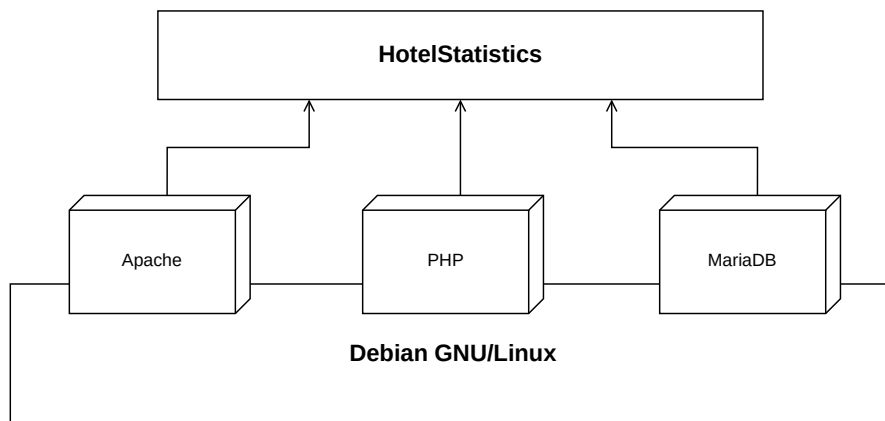


Figura 5.1: Il deployment diagram dell'intero sistema software

Bibliografia

- [1] Paolo Atzeni et al. *Basi di dati*. McGraw Hill, 2018.
- [2] Elena Baralis. *Basi di dati. Temi d'esame svolti*. Esculapio, 2001.
- [3] Alan Beaulieu. *Learning SQL*. O'Reilly, 2020.
- [4] Ramez Elmasri e Shamkant B. Navathe. *Sistemi di basi di dati*. Pearson, 2018.
- [5] MariaDB Foundation. *MariaDB Server Documentation*. 2022.
- [6] Micheal Kofler. *MySQL 5 Guida completa*. Apogeo, 2006.
- [7] Oracle. *MySQL Server Documentation*. 2022.
- [8] George Reese. *MySQL Pocket Reference*. O'Reilly, 2007.