Basi di Dati

Progetto A.A. 2021/2022

Sistema di Customer Relationship Managment

0266523

Valentina Martini

Indice

1. Descrizione del Minimondo

Un sistema di Customer Relationship Management (o gestione delle relazioni con i clienti) è un sistema informativo che verte sulla fidelizzazione del cliente. Si vuole realizzare un sistema CRM per un'azienda marketing-oriented che intende realizzare relazioni durevoli di breve e lungo periodo con i propri clienti, massimizzando quindi il valore degli stessi.
La base di dati del sistema informativo dell'azienda di CRM deve poter memorizzare le informazioni su tutti i clienti di interesse dell'azienda, caratterizzati da nome, cognome, codice fiscale, data di nascita ed un insieme di contatti, sia in forma di indirizzi che di recapiti telefonici, email, fax. Alcuni dei clienti sono società che ricevono servizi dalla società di CRM. Di questi interessa anche mantenere il numero di partita IVA. Di tutti i clienti interessa sapere qual è la data di registrazione nel sistema di CRM.
L'azienda di CRM in questione è di dimensione elevata ed ha a disposizione vari funzionari che interagiscono con i clienti. A ciascun utente aziendale del sistema viene assegnato un sottoinsieme di clienti da gestire.
Su base periodica, gli operatori dell'azienda di CRM contattano i clienti mediante uno dei recapiti forniti. In questa fase operativa, l'utente deve inserire una nota testuale in cui viene riportato un breve resoconto dell'interazione con il cliente, annotando anche possibili risposte affermative alle proposte commerciali. Una risposta positiva di accettazione di una proposta commerciale può essere associata ad un appuntamento in sede. L'azienda ha più sedi, ciascuna caratterizzata da un indirizzo. In ciascuna sede sono presenti una o più sale riunione, in cui è possibile ricevere i clienti. Non è possibile assegnare una stessa sede, nello stesso giorno ed alla stessa ora, a più di un cliente. Agli appuntamenti partecipano i clienti e gli operatori dell'azienda.
L'azienda ha anche un gruppo di manager che definisce quali sono le proposte commerciali che l'azienda offre. Ogni proposta è identificata da un codice alfanumerico definito internamente dall'azienda. I manager hanno la possibilità di creare nuove proposte e di segnalare che alcune proposte già presenti nel sistema sono terminate, ovverosia che non possono più essere fornite ai clienti.
Infine, l'azienda ha un settore commerciale i cui membri reclutano nuovi clienti e li inseriscono all'interno del sistema.
In generale, il sistema informativo deve fornire le seguenti possibilità.
* Visualizzare il singolo cliente, eventualmente con i dati dell'azienda e del referente aziendale, con tutti i dettagli e le caratteristiche, l'elenco delle note cliente e l'elenco dei servizi di consulenza acquistati.
* Possibilità di visualizzare l'elenco clienti a cui un utente è assegnato.
* Gestione delle note cliente: ogni volta che un cliente viene contattato deve essere possibile registrare/modificare/cancellare una o più note relative alla conversazione avvenuta e dell'utente che l'ha registrata.
* Gestione delle opportunità: per ogni cliente deve essere possibile inserire una nuova opportunità, cioè una proposta commerciale.
* Gestione degli appuntamenti: deve essere possibile inserire un appuntamento con una nota descrittiva, una data/ora e un cliente a cui è riferito.
* Visualizzazione dell'agenda degli appuntamenti per un utente.
* Possibilità di inserire nuovi servizi di consulenza (riservata ai manager).
* Possibilità di inserire nuovi clienti (riservata al settore commerciale).
* Possibilità di inserire nuovi utenti dell'applicativo web (riservata ai manager).

2. Analisi dei Requisiti

Identificazione dei termini ambigui e correzioni possibili

Linea	Termine	Nuovo termine	Motivo correzione
7-8	insieme di contatti, sia in forma di indirizzi che di recapiti telefonici, email, fax.	insieme di contatti: email, fax.	La frase era poco chiara.
9	questi	società	Rendere più chiaro che si sta parlando soltanto dei clienti che sono anche società.
12-15-34-37-42	utente	operatore	Il termine utente è troppo generico.
20	sede	sala	In questo caso con "sede" si intende la sala in cui avverrà l'appuntamento fissato.

Specifica disambiguata

Un sistema di Customer Relationship Management (o gestione delle relazioni con i clienti) è un sistema informativo che verte sulla fidelizzazione del cliente. Si vuole realizzare un sistema CRM per un'azienda marketing-oriented che intende realizzare relazioni durevoli di breve e lungo periodo con i propri clienti, massimizzando quindi il valore degli stessi.
La base di dati del sistema informativo dell'azienda di CRM deve poter memorizzare le informazioni su tutti i clienti di interesse dell'azienda, caratterizzati da nome, cognome, codice fiscale, data di nascita ed un insieme di contatti, sia in forma di indirizzi che di recapiti telefonici, email, fax. Alcuni dei clienti sono società che ricevono servizi dalla società di CRM. Delle società interessa anche mantenere il numero di partita IVA. Di tutti i clienti interessa sapere qual è la data di registrazione nel sistema di CRM.
L'azienda di CRM in questione è di dimensione elevata ed ha a disposizione vari funzionari che interagiscono con i clienti. A ciascun operatore viene assegnato un sottoinsieme di clienti da gestire. Su base periodica, gli operatori dell'azienda di CRM contattano i clienti mediante uno dei recapiti forniti. In questa fase operativa, l'operatore deve inserire una nota testuale in cui viene riportato un breve resoconto dell'interazione con il cliente, annotando anche possibili risposte affermative alle proposte commerciali. Una risposta positiva di accettazione di una proposta commerciale può essere

associata ad un appuntamento in sede. L'azienda ha più sedi, ciascuna caratterizzata da un indirizzo. In ciascuna sede sono presenti una o più sale riunione, in cui è possibile ricevere i clienti. Non è possibile assegnare una stessa sala, nello stesso giorno ed alla stessa ora, a più di un cliente. Agli appuntamenti partecipano i clienti e gli operatori dell'azienda.
L'azienda ha anche un gruppo di manager che definisce quali sono le proposte commerciali che l'azienda offre. Ogni proposta è identificata da un codice alfanumerico definito internamente dall'azienda. I manager hanno la possibilità di creare nuove proposte e di segnalare che alcune proposte già presenti nel sistema sono terminate, ovverosia che non possono più essere fornite ai clienti.
Infine, l'azienda ha un settore commerciale i cui membri reclutano nuovi clienti e li inseriscono all'interno del sistema.
In generale, il sistema informativo deve fornire le seguenti possibilità.
* Visualizzare il singolo cliente, eventualmente con i dati dell'azienda e del referente aziendale, con tutti i dettagli e le caratteristiche, l'elenco delle note cliente e l'elenco dei servizi di consulenza acquistati.
* Possibilità di visualizzare l'elenco clienti a cui un operatore è assegnato.
* Gestione delle note cliente: ogni volta che un cliente viene contattato deve essere possibile registrare/modificare/cancellare una o più note relative alla conversazione avvenuta e dell'operatore che l'ha registrata.
* Gestione delle opportunità: per ogni cliente deve essere possibile inserire una nuova opportunità, cioè una proposta commerciale.
* Gestione degli appuntamenti: deve essere possibile inserire un appuntamento con una nota descrittiva, una data/ora e un cliente a cui è riferito.
* Visualizzazione dell'agenda degli appuntamenti per un operatore.
* Possibilità di inserire nuovi servizi di consulenza (riservata ai manager).
* Possibilità di inserire nuovi clienti (riservata al settore commerciale).
* Possibilità di inserire nuovi utenti dell'applicativo web (riservata ai manager).

Glossario dei Termini

Termine	Descrizione	Sinonimi	Collegamenti
Cliente	Cliente dell'azienda, potrebbero essere delle società		Operatori, appuntamento, proposta commerciale

Funzionario	Dipendente dell'azienda	Operatore, Manger, Membro del settore commerciale, utente	
Operatore	Dipendente che si occupa di interagire con il cliente	Funzionario, utente	Appuntamento, cliente
Proposta commerciale	Proposte commerciali che l'azienda offre	Servizi di consulenza, opportunità	Manager, cliente
Sede	Edificio dell'azienda in cui si possono svolgere gli appuntamenti con i clienti		Sala, appuntamento
Sala riunione	Stanze di una sede dell'azienda in cui è possible ricevere i clienti		Sede, appuntamento
Appuntamento	Incontro tra Operatore e Cliente		Cliente, operatore, sala, sede
Manager	Dipendente che si occupa delle proposte commerciali	Funzionario, utente	Proposte commerciali
Membro del settore commerciale	Dipendente che si occupa di trovare nuovi clienti	Funzionario, utente	

Raggruppamento dei requisiti in insiemi omogenei

Frasi relative ai clienti
I clienti sono caratterizzati da nome, cognome, codice fiscale, data di nascita, ed un insieme di contatti: email, fax. Alcuni dei clienti sono società che ricevono servizi dalla società di CRM. Delle società interessa anche mantenere il numero di partita IVA. Di tutti i clienti interessa sapere qual è la data di registrazione nel sistema di CRM. Il cliente viene associato ad un operatore, il quale li contatta su base periodica, mediante uno dei recapiti forniti. Agli appuntamenti partecipano i clienti e gli operatori dell'azienda. Riguardo ai clienti, il sistema deve fornire la possibilità di: Visualizzare il singolo cliente, eventualmente con i dati dell'azienda e del referente aziendale, con tutti i dettagli e le caratteristiche, l'elenco delle note cliente e l'elenco dei servizi di consulenza acquistati.

Frasi relative agli operatori
A ciascun operatore viene assegnato un sottoinsieme di clienti da gestire. Su base periodica, gli operatori dell'azienda di CRM contattano i clienti mediante uno dei recapiti forniti. In questa fase operativa, l'operatore deve inserire una nota testuale in cui viene riportato un breve resoconto dell'interazione con il cliente, annotando anche possibili risposte affermative alle proposte commerciali. Agli appuntamenti partecipano i clienti e gli operatori dell'azienda. Riguardo agli operatori, il sistema deve fornire la possibilità di: Visualizzare l'elenco clienti a cui un operatore è assegnato. Gestire le note cliente: ogni volta che un cliente viene contattato deve essere possibile registrare/modificare/cancellare una o più note relative alla conversazione avvenuta e dell'operatore che l'ha registrata. Visualizzare l'agenda degli appuntamenti per un operatore.

Frasi relative alle proposte commerciali
Una risposta positiva di accettazione di una proposta commerciale può essere associata ad un appuntamento in sede. Le proposte commerciali vengono create dai Manager dell'azienda. Alcune proposte presenti nel sistema posso essere terminate, ovverosia non possono più essere fornite ai clienti. Ogni proposta è identificata da un codice alfanumerico definito internamente dall'azienda. Riguardo alle proposte commerciali, il sistema deve fornire la possibilità di: Gestire le opportunità: per ogni cliente deve essere possibile inserire una nuova opportunità, cioè una proposta commerciale. Inserire nuovi servizi di consulenza.

Frasi relative alle sedi
L'azienda ha più sedi, ciascuna caratterizzata da un indirizzo. In ciascuna sede sono presenti una o più sale riunione, in cui è possibile ricevere i clienti.

Frasi relative alle sale riunione
In ciascuna sede sono presenti una o più sale riunione, in cui è possibile ricevere i clienti. Non è possibile assegnare una stessa sala, nello stesso giorno ed alla stessa ora, a più di un cliente.

Frasi relative agli appuntamenti
Una risposta positiva di accettazione di una proposta commerciale può essere associata ad un appuntamento in sede. Gli appuntamenti avvengono nelle sale riunione delle varie sedi. Non è possibile assegnare una stessa sala, nello stesso giorno ed alla stessa ora, a più di un cliente. Agli appuntamenti partecipano i clienti e gli operatori dell'azienda. Riguardo agli appuntamenti, il sistema deve fornire la possibilità di: Gestire gli appuntamenti: deve essere possibile inserire un appuntamento con una nota descrittiva, una data/ora e un cliente a cui è riferito.

Frasi relative ai manager
L'azienda ha anche un gruppo di manager che definisce quali sono le proposte commerciali che l'azienda offre. I manager hanno la possibilità di creare nuove proposte e di segnalare che alcune proposte già presenti nel sistema sono terminate, ovverosia che non possono più essere fornite ai clienti. Riguardo ai manager, il sistema deve fornire ad essi la possibilità di: Inserire nuovi servizi di consulenza. Inserire nuovi utenti dell'applicativo web.

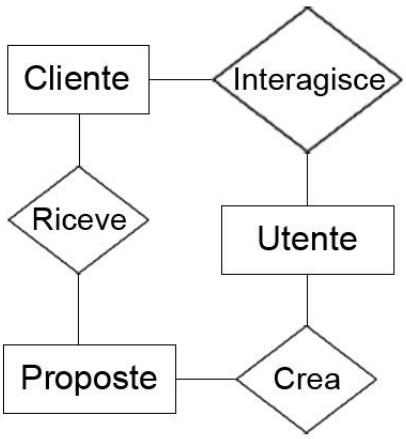
Frasi relative ai membri del settore commerciale
Infine, l'azienda ha un settore commerciale i cui membri reclutano nuovi clienti e li inseriscono all'interno del sistema.

3. Progettazione concettuale

Costruzione dello schema E-R

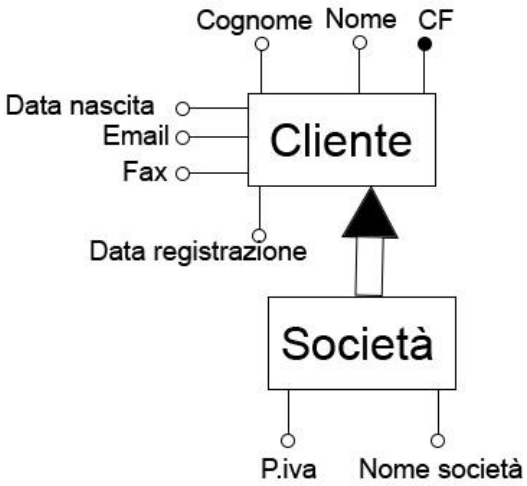
Si è scelta di adottare una strategia mista.

Schema Scheletro:

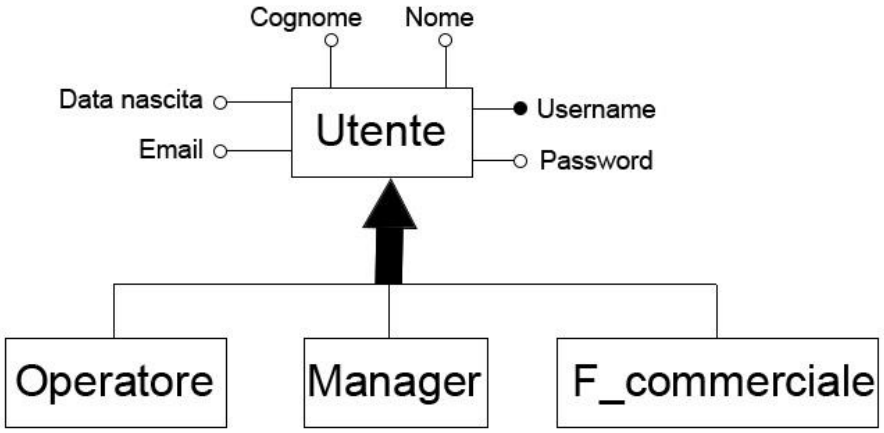


- Approfondisco il concetto Cliente:

  Cliente caratterizzato da: nome, cognome, codice fiscale, data di nascita, ed un insieme di contatti: email, fax. Inoltre, alcuni clienti possono essere delle società di cui interessa: il nome e la partita iva. Se un cliente è anche società, allora i dati relativi al cliente faranno riferimento al referente aziendale.

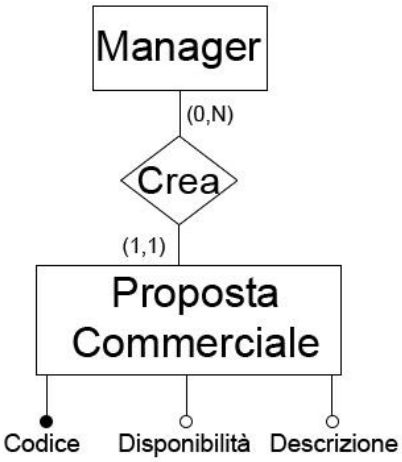


- Approfodisco il concetto Utente. Si hanno 3 tipi di utenti: Operatore, Manager e Funzionario del settore commerciale.



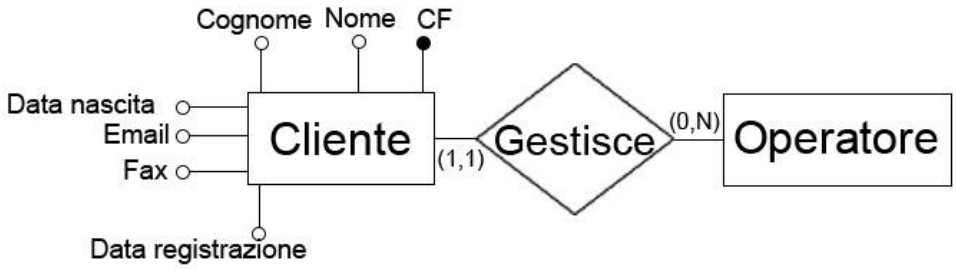
- Approfondisco il concetto Proposte commerciali: Ogni proposta è identificata da un codice alfanumerico, e può essere disponibile, o non disponibile (ovverosia non possono più essere fornite ai clienti).

- Approfondisco il concetto Crea: Le proposte commerciali vengono create soltanto dai Manager dell'azienda

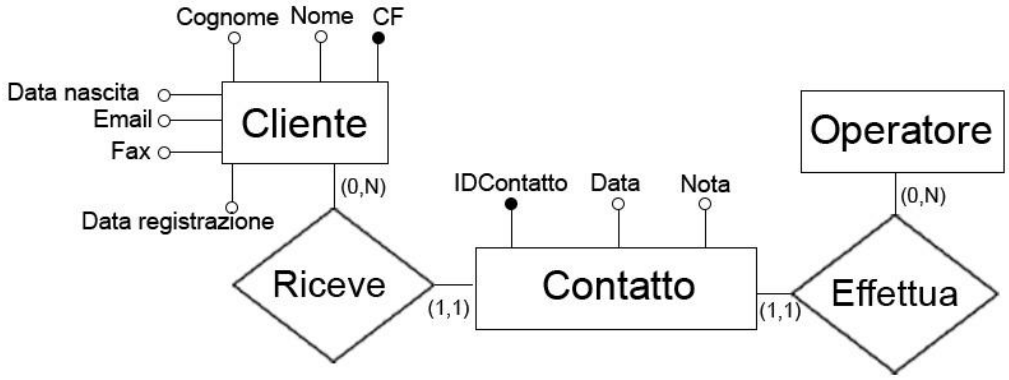


- Approfondisco il concetto Interagisce: soltanto gli Operatori interagiscono con i Clienti. Dalle specifiche si evince che:

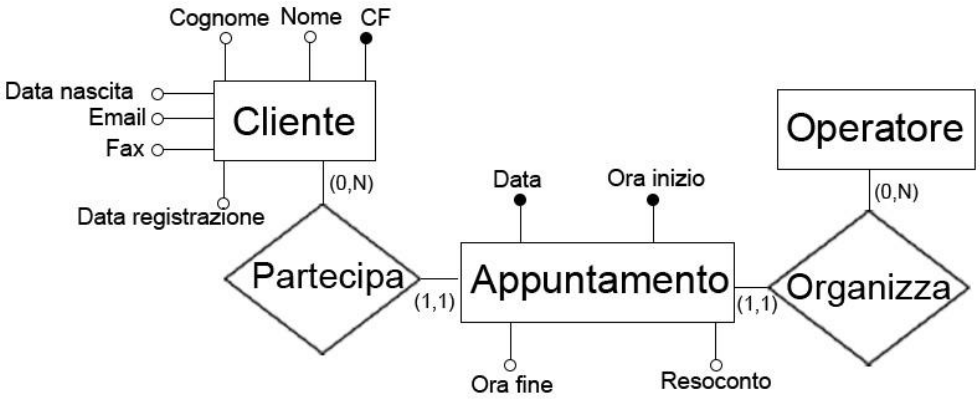
- A ciascun operatore viene assegnato un sottoinsieme di clienti da gestire.



- Su base periodica, gli operatori dell'azienda di CRM contattano i clienti mediante uno dei recapiti forniti. In questa fase operativa, l'operatore deve inserire una nota testuale in cui viene riportato un breve resoconto dell'interazione con il cliente.



- Gli Operatori organizzano Appuntamenti con i Clienti. Degli appuntamenti interessa mantenere: un resoconto, la data e l'ora in cui è avvenuto.

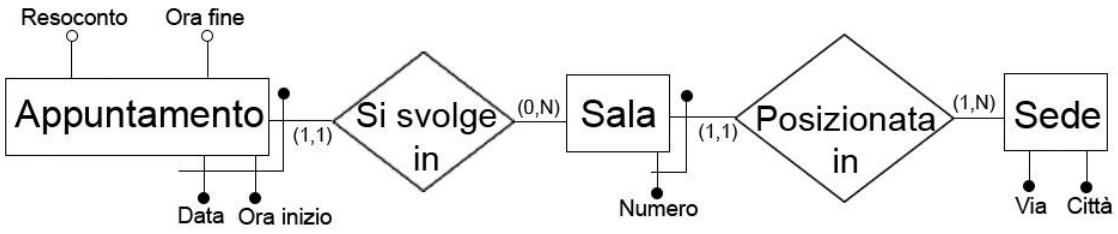


- Approfondisco il concetto di Appuntamento: Gli appuntamenti avvengono nelle sale riunione delle varie sedi.

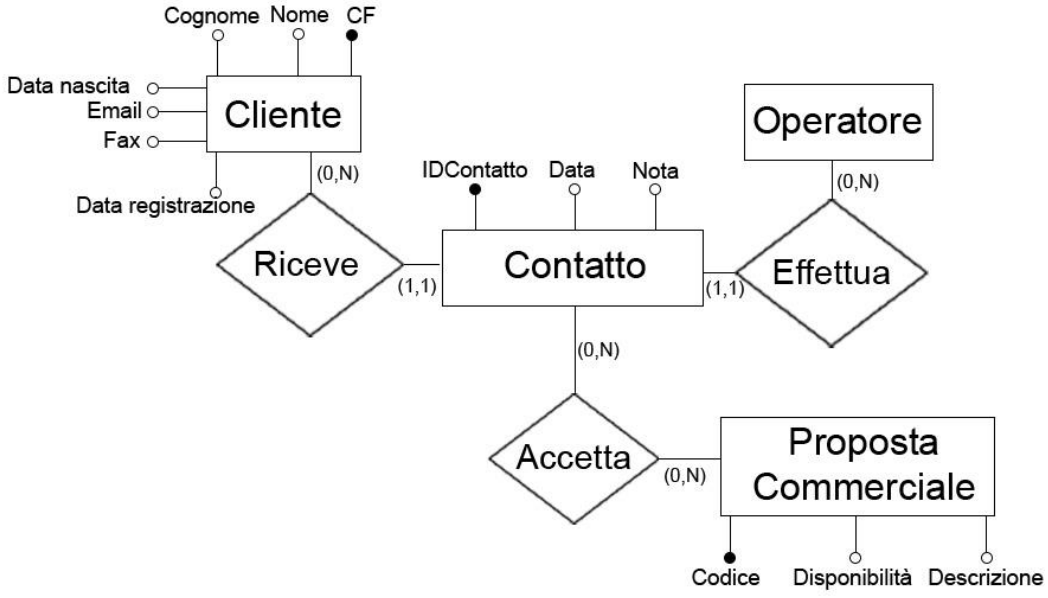
- Approfondisco i concetti di sala e sede: L'azienda ha più sedi, ciascuna caratterizzata da un indirizzo. In ciascuna sede sono presenti una o più sale riunione, in cui è possibile ricevere i clienti.

  Visto che, per esempio, la sala numero 1 è possibile che sia presente in più sedi, e che comunque il concetto di sala non ha senso senza il concetto di sede. Si è scelto che Sala sia un'entità debole di Sede.

  Visto che non è possibile assegnare una stessa Sala, nello stesso giorno ed alla stessa ora, a più di un cliente. Si è scelto di mettere Appuntamento come entità debole di Sala



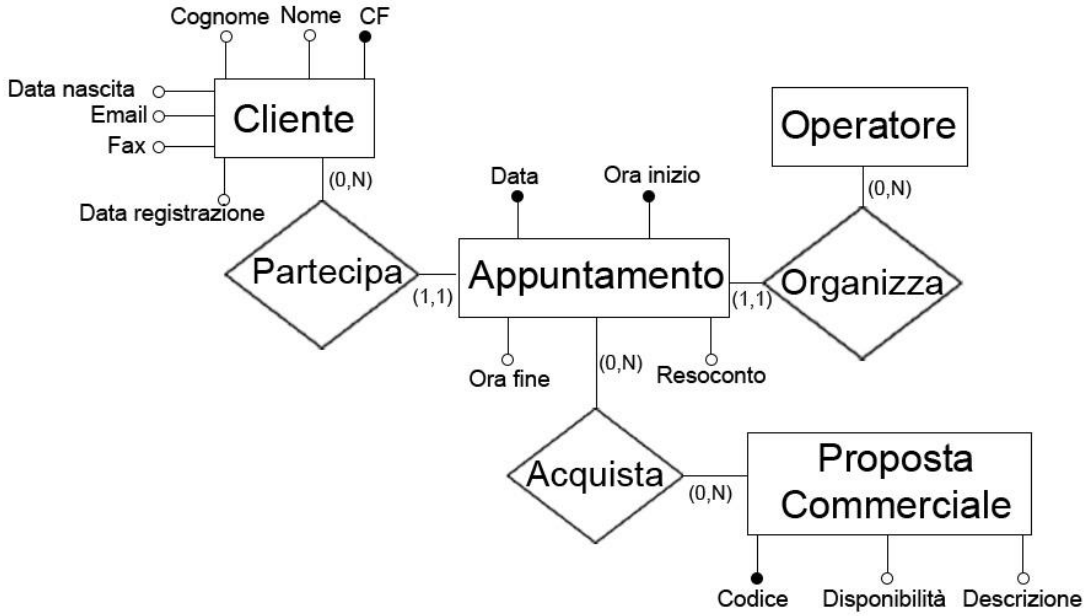
- Approfondisco il concetto di Cliente-Riceve-Proposte:
  - Durante un Contatto con un Operatore, il Cliente può decidere di accettare una o più proposte, o anche nessuna.



Si è scelto di associare la Proposta Commerciale al Contatto e non direttamente al Cliente, in modo tale da mantenere un maggior numero di informazioni. Come, per esempio, in quale

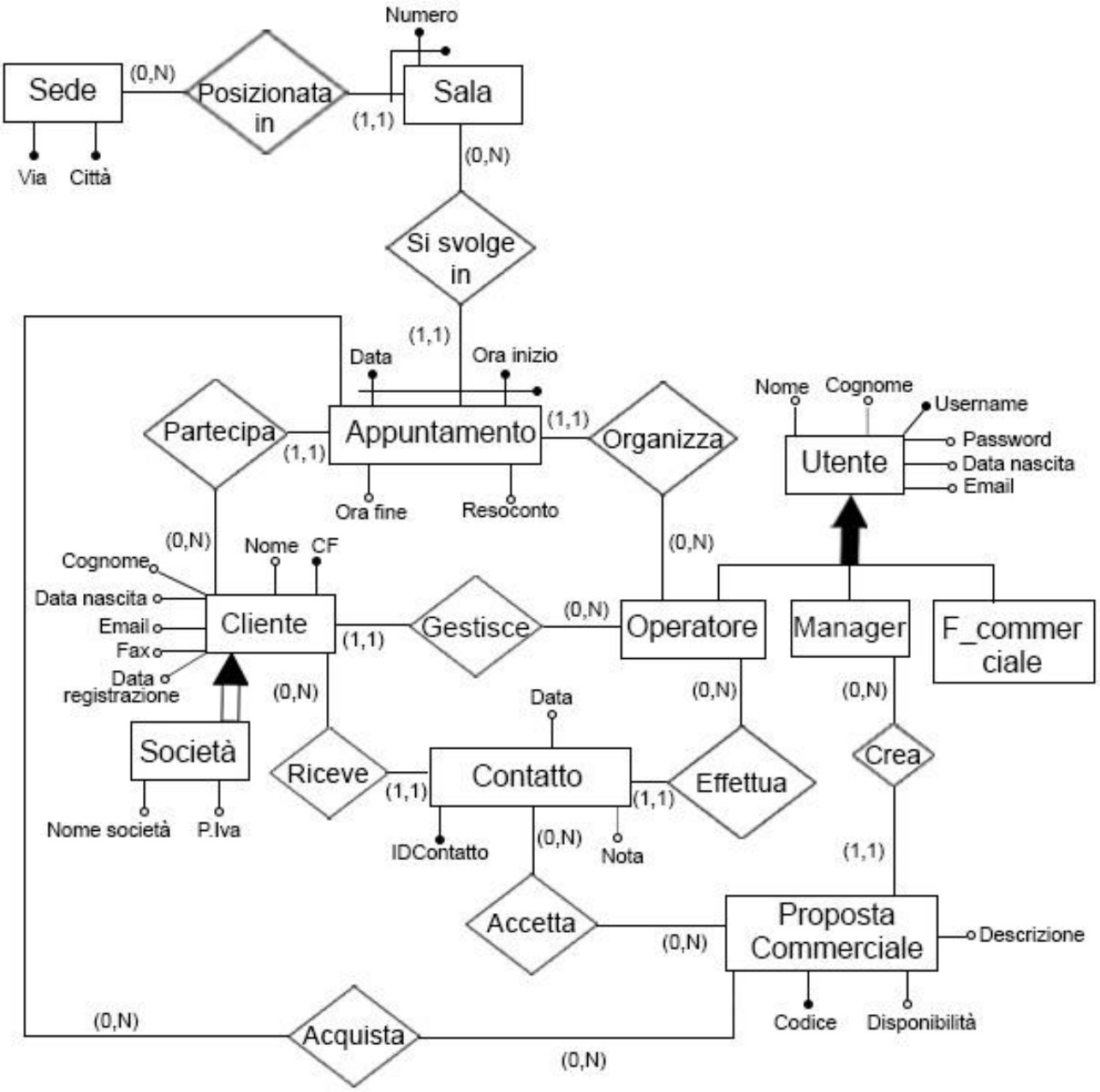
Contatto si è accettata una determinata Proposta, oppure chi è l'Operatore che ha portato all'accettazione della Proposta (sarebbe possibile risalire comunque all'Operatore visto che un cliente è gestito solo da un operatore, ma se il Cliente venisse assegnato ad un nuovo Operatore, si perderebbe tale informazione).

- Durante un Appuntamento con un Operatore, il Cliente può decidere di acquistare una o più proposte, o anche nessuna.



Si è scelto di associare la Proposta Commerciale al Contatto e non direttamente al Cliente, in modo tale da mantenere un maggior numero di informazioni. Come, per esempio, in quale appuntamento si è acquistata una determinata proposta oppure chi è l'Operatore che ha portato all'acquisto della Proposta.

Integrazione finale



Regole aziendali

1. Non è possibile assegnare una stessa sala, nello stesso giorno ed alla stessa ora, a più di un cliente.
2. Una proposta commerciale terminata non deve essere proposta ai clienti.
3. Un cliente non può acquistare proposte commerciali terminate.
4. Un operatore non deve poter contattare un cliente, se il cliente non è tra il sottoinsieme di clienti che l'operatore gestisce.

5. Un operatore non deve poter organizzare un appuntamento con un cliente, se il cliente non è tra il sottoinsieme di clienti che l'operatore gestisce.
6. Un operatore non deve poter fissare più appuntamenti nella stessa data e ora con clienti diversi.

Dizionario dei dati

Entità	Descrizione	Attributi	Identificatori
Cliente	Cliente dell'azienda	Nome, cognome, CF, data nascita, email, fax, data registrazione	CF
Società	Società che sono clienti dell'azienda	Piva, nome società	
Utente	Dipendenti dell'azienda	Username, password, nome, cognome, data nascita, email	Username
Operatore	Dipendenti dell'azienda che interagiscono con i clienti		
Appuntamento	Incontro tra operatore e cliente	Data, ora inizio, ora fine, resoconto	Data, ora inizio
Contatto	Contatto periodico che avviene tra operatore e cliente mediante uno dei recapiti forniti	IDcontatto, data, nota	IDcontatto
Manager	Dipendente che si occupa delle proposte commerciali		
Proposta	Proposta commerciale che l'azienda offer ai clienti	Codice, disponibilità, descrizione	Codice
Sala	Stanza di una sede dell'azienda in cui è possible ricevere i clienti	numero	Numero
Sede	Edificio dell'azienda in cui si possono svolgere gli appuntamenti con i clienti	Via, città	Via, città
F_commerciale	Dipendenti del settore commerciale che si occupano di reclutare nuovi clienti		

4. Progettazione logica

Volume dei dati

Concetto nello schema	Tipo[1]	Volume atteso
Cliente	E	4.800.000
Utente	E	10.000
Operatore	E	6.000
Manager	E	1.000
F_commerciale	E	3.000
Proposte	E	40
Contatto	E	4.800.000 (Almeno pari al numero di clienti)
Appuntamento	E	3.600.000 (circa l'75% dei clienti accorda un appuntamento)
Sala	E	1500 (1 sede = 240 operatori. 1 appuntamento = 1 ora. 1 giornata lavorativa = 8 ore. Al giorno, ogni operatore necessita di usare una sala per 2 appuntamenti. 1 sala = 8 appuntamenti; 4 operatori. 1 sede = 240 / 4 = 60 sale 25 * 60 = 1500
Sede	E	25
Organizza	R	3.600.000 (Uguale al numero di appuntamenti)
Partecipa	R	3.600.000 (Uguale al numero di appuntamenti)
Effettua	R	4.800.000

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

		(Uguale al numero di conatti)
Riceve	R	4.800.000 (Uguale al numero di conatti)
Accetta	R	9.600.000 (In media ogni cliente è interessato a 2 proposte l'anno)
Acquista	R	3.600.000 (In media ogni cliente acquista 1 proposta l'anno)
Crea	R	40
Gestisce	R	4.800.000
Posizionata in	R	1500
Si svolge in	R	3.600.000

Tavola delle operazioni

Cod.	Descrizione	Frequenza attesa
OP1	Visualizzare: i dettagli del singolo cliente, eventualmente con i dati dell'azienda e del referente aziendale	36.000/giorno Supponendo un operatore interagisce con 6 clienti al giorno e che prima di interagire con il singolo cliente voglia rivedere i suoi dati.
OP2	Visualizzare l'elenco dei clienti a cui un operatore è assegnato	36.000/giorno Supponendo che per visualizzare il singolo cliente, bisogna prima visualizzare l'elenco dei clienti.
OP3	Inserire un contatto con la relativa nota della conversazione avvenuta tra operatore e cliente	24.000/giorno Supponendo che ogni operatore contatta 4 clienti al giorno.
OP4	Modificare/Cancellare la nota relativa alla conversazione avvenuta tra operatore e cliente	2000/mese
OP5	Aggiungere l'interesse di un cliente ad una determinata proposta commerciale	48.000/giorno Un cliente viene contattato mediamente 1 volta l'anno, ed ogni cliente è interessato mediamente a 2 proposte.

		Ogni operatore contatta al giorno 4 clienti.
OP6	Aggiungere l'acquisto di una proposta commerciale da parte di un cliente	12.000/giorno Un cliente partecipa ad un appuntamento mediamente 1 volta l'anno, ed ogni cliente acquista mediamente ad 1 proposta. Ogni operatore incontra 2 clienti al giorno.
OP7	Inserire un nuovo appuntamento	18.000/giorno Ogni operatore contatta 4 clienti al giorno. Supponendo che il 75% dei clienti accorda anche un appuntamento in sede.
OP8	Visualizzare l'elenco degli appuntamenti di un operatore	25.800 /giorno Per visualizzare I dettagli del singolo appuntamento, bisogna prima visualizzare l'elenco.
OP9	Inserire una nuova proposta commerciale	10/anno
OP10	Modificare la disponibilità di una proposta commerciale	2/anno
OP11	Inserire un nuovo cliente e assegnarlo ad un operatore	1000/mese
OP12	Inserire un nuovo utente	500/anno
OP13	Modificare la data/ora di un appuntamento	1.800/giorno Supponendo che circa il 10% degli appuntamenti richieda una modifica.
OP14	Visualizzare le proposte a cui un cliente è interessato	12.000/giorno Ogni operatore incontra 2 clienti al giorno, supponendo che prima di fare l'appuntamento l'operatore voglia controllare a quali proposte il cliente è interessato.
OP15	Visualizzare l'elenco delle proposte che sono state acquistate e il numero di volte che sono state acquistate	12/anno Supponendo che un maneger prima di inserire una nuova proposta (10/anno), e prima cambiare la disponibilita (2/anno), voglia controllare quali proposte risultano più richieste e quali meno.
OP16	Visualizzare i dettagli del singolo appuntamento	25.800 /giorno Ogni operatore incontra 2 clienti al giorno, supponendo che prima di fare l'appuntamento voglia riguardare le informazioni (12.000) e

		dopo aver fatto l'appuntamento l'operatore debba andare a modificare il resoconto e inserire eventuali proposte acquistate (12.000). Inoltre per modificare data/ora, bisogna prima visualizzare il singolo appuntamento (1.800)
OP17	Modificare il resoconto relativo ad un appuntamento avvenuto tra operatore e cliente	12.000/giorno Ogni operatore incontra 2 clienti al giorno, supponendo che dopo aver fatto l'appuntamento l'operatore debba andare a modificare il resoconto.
OP18	Visualizzare i dettagli del singolo contatto	24.064 /giorno Supponendo che gli operatori devono visualizza il contatto per poter modificare/eliminare la nota (2000 /mese = 64 /giorno) o per poter inserire le proposte accettate (24.000/giorno)
OP19	Visualizzare l'elenco delle proposte disponibili	24.000 /giorno Supponendo che un'operatore visualizzi l'elenco delle proposte disponibili prima di fare un contatto (4 contatti al giorno), per capire quale proposte proporgli.
OP20	Visualizzare i dettagli di una singola proposta commerciale	2 /anno Supponendo che per modificare la disponibilità di una proposta, bisogna prima visualizzarla
OP21	Login	10.000 /giorno circa il numero di utenti
OP22	Visualizzare l'elenco delle proposte commerciali	2/anno Supponendo che per visualizzare I dettagli di una proposta, bisogna prima visualizzare l'elenco delle proposte
OP23	Visualizzare l'elenco dei contatti di un operatore	24.064/giorno Supponendo che per visualizzare il singolo contatto, bisogna aver visualizza prima l'elenco dei contatti.
OP24	Visualizzare l'elenco di tutti i clienti	1000/mese Per inserire un nuovo cliente, un funzionario commerciale deve prima visualizzare l'elenco
OP25	Visualizzare l'elenco di tutti gli utenti	500/anno Per inserire un nuovo utente, un manager deve

		prima visualizzare l'elenco
OP26	Visualizzare le sale e le relative sedi	18.000/giorno Supponendo che un operatore voglia vedere le sale a disposizione prima di inserire un nuovo appuntamento
OP27	Visualizzare l'elenco dei contatti e delle note relative alle conversazioni con un cliente	24.000/giorno Supponendo che prima di effettuare un contatto con un cliente, l'operatore voglia rivedere i contatti avvenuti con quel cliente
OP28	Visualizzare l'elenco delle proposte commerciali acquistate da un cliente	36.000 /giorno Supponendo che prima di effettuare un contatto e un appuntamento con un cliente, l'operatore voglia farsi un'idea delle proposte che ha già acquistato

Costo delle operazioni

OP1			
Costo operazione = 1		**Costo totale = 36.000 /giorno**	
Concetto	**Costrutto**	**Accessi**	**Tipo**
Cliente	E	1	L

OP2			
Costo operazione = 1601		**Costo totale = 57.636.000 /giorno**	
Concetto	**Costrutto**	**Accessi**	**Tipo**
Operatore	E	1	L
Gestisce	R	N	L
Cliente	E	N	L

N = numero di clienti associati ad un operatore = si suppone che ogni operatore sia associato mediamente a 800 clienti

OP3			
Costo operazione = 8		**Costo totale = 192.000 /giorno**	
Concetto	**Costrutto**	**Accessi**	**Tipo**
Operatore	E	1	L
Effettua	R	1	S
Contatto	E	1	S
Riceve	R	1	S
Cliente	E	1	L

OP4			
Costo operazione = 2		**Costo totale = 4.000 /mese**	
Concetto	**Costrutto**	**Accessi**	**Tipo**
Contatto	E	1	S

OP5			
Costo operazione = 4		**Costo totale = 192.000 /giorno**	
Concetto	**Costrutto**	**Accessi**	**Tipo**
Contatto	E	1	L
Accetta	R	1	S
Proposta	E	1	L

OP6			
Costo operazione = 4		**Costo totale = 48.000 /giorno**	
Concetto	**Costrutto**	**Accessi**	**Tipo**
Appuntamento	E	1	L
Acquista	R	1	S
Proposta	E	1	L

OP7			
Costo operazione = 11		**Costo totale = 198.000 /giorno**	
Concetto	**Costrutto**	**Accessi**	**Tipo**
Operatore	E	1	L
Organizza	R	1	S
Appuntamento	E	1	S
Partecipa	R	1	S
Cliente	E	1	L
Si svolge in	R	1	S
Sala	E	1	L

OP8			
Costo Operazione = 1201		**Costo totale = 30.985.800 /giorno**	
Concetto	**Costrutto**	**Accessi**	**Tipo**
Operatore	E	1	L
Organizza	R	N	L
Appuntamento	E	N	L

N = numero di appuntamenti organizzati da un operatore. Avendo supposto che ogni operatore è associato mediamente a 800 clienti, di cui solo il 75% accetta un appuntamento, e che ogni cliente partecipa mediamente ad 1 appuntamento l'anno:

OP9			
Costo operazione = 5		**Costo totale = 50 /anno**	
Concetto	**Costrutto**	**Accessi**	**Tipo**
Manager	E	1	L
Crea	R	1	S
Proposta	E	1	S

OP10			
Costo operazione = 2		**Costo totale = 4 /anno**	
Concetto	**Costrutto**	**Accessi**	**Tipo**
Proposta	E	1	S

OP11			
Costo operazione = 5		**Costo totale = 5.000 /mese**	
Concetto	**Costrutto**	**Accessi**	**Tipo**
Cliente	E	1	S
Gestisce	R	1	S
Operatore	E	1	L

OP12			
Costo operazione = 2		**Costo totale = 1.000 /anno**	
Concetto	**Costrutto**	**Accessi**	**Tipo**
Utente	E	1	S

OP13			
Costo operazione = 2		**Costo totale = 3.600 /giorno**	
Concetto	**Costrutto**	**Accessi**	**Tipo**
Appuntamento	E	1	S

OP14			
Costo operazione = 7		**Costo totale = 84.000 /giorno**	
Concetto	**Costrutto**	**Accessi**	**Tipo**
Cliente	E	1	L
Riceve	R	N1	L
Contatto	E	N1	L
Accetta	R	N2	L
Proposte	E	N2	L

N1 = numero dei contatti che un cliente ha avuto con l'operatore = si suppone che un cliente venga contattato mediamente 1 volta l'anno

N2 = numero medio delle proposte a cui un cliente mostra interesse per ogni contatto = si suppone che un cliente sia mediamente interessato a 2 proposte per contatto

OP15			
Costo operazione = 3.600.000		**Costo totale = 43.200.000 /giorno**	
Concetto	**Costrutto**	**Accessi**	**Tipo**
Proposte	E	40	L
Acquista	R	N	L

N = numero medio di volte che una proposta viene acquistata = supponendo che ogni cliente acquista in media 1 proposta = 90.000

OP16			
Costo operazione = 1		**Costo totale = 25.800 /giorno**	
Concetto	**Costrutto**	**Accessi**	**Tipo**
Appuntamento	E	1	L

OP17			
Costo operazione = 2		**Costo totale = 24.000 /giorno**	
Concetto	**Costrutto**	**Accessi**	**Tipo**
Appuntamento	E	1	S

OP18			
Costo operazione = 1		**Costo totale = 24.064 /giorno**	
Concetto	**Costrutto**	**Accessi**	**Tipo**
Contatto	E	1	L

OP19			
Costo operazione = 32		**Costo totale = 768.000 /giorno**	
Concetto	**Costrutto**	**Accessi**	**Tipo**
Proposte	E	N	L

N = numero proposte disponibile, suppendo che il 80% delle proposte siano disponibili = 32

OP20			
Costo operazione = 1		**Costo totale = 2 /anno**	
Concetto	**Costrutto**	**Accessi**	**Tipo**
Proposte	E	1	L

OP21			
Costo operazione = 1		**Costo totale = 10.000 /giorno**	
Concetto	**Costrutto**	**Accessi**	**Tipo**
Utenti	E	1	L

OP22			
Costo operazione = 40		**Costo totale = 80 /anno**	
Concetto	**Costrutto**	**Accessi**	**Tipo**
Proposte	E	40	L

OP23			
Costo operazione = 1601		**Costo totale = 38.526.464 /giorno**	
Concetto	**Costrutto**	**Accessi**	**Tipo**
Operatore	E	1	L
Effettua	R	N	L
Contatto	E	N	L

N = numero di contatti effettuati da un operatore. Avendo supposto che ogni operatore è associato mediamente a 800 clienti, e che ogni cliente riceve mediamente ad 1 contatto l'anno:

OP24			
Costo operazione = 4.800.000		**Costo totale = 4.800.000.000 /mese**	
Concetto	**Costrutto**	**Accessi**	**Tipo**
Clienti	E	4.800.000	L

OP25			
Costo operazione = 10.000		**Costo totale = 5.000.000 /anno**	
Concetto	**Costrutto**	**Accessi**	**Tipo**
Utenti	E	10.000	L

OP26			
Costo operazione = 3.000		**Costo totale = 54.000.000 /giorno**	
Concetto	**Costrutto**	**Accessi**	**Tipo**
Sala	E	60	L
Posizionata in	R	60	L
Sede	E	25	L

OP27			
Costo operazione = 3		**Costo totale = 72.000 /giorno**	
Concetto	**Costrutto**	**Accessi**	**Tipo**
Cliente	E	1	L
Riceve	R	N	L
Contatto	E	N	L

N = numero dei contatti che un cliente ha avuto con l'operatore = si suppone che un cliente venga contattato mediamente 1 volta l'anno

OP28			
Costo operazione = 5		**Costo totale = 180.000 /giorno**	
Concetto	**Costrutto**	**Accessi**	**Tipo**
Cliente	E	1	L
Partecipa	R	N1	L
Appuntamento	E	N1	L
Acquista	R	N2	L
Proposta	E	N2	L

N1 = numero degli appuntamenti a cui un cliente ha partecipato con l'operatore = si suppone che un cliente partecipi mediamente ad 1 appuntamento l'anno

N2 = numero medio delle proposte che un cliente acquista per ogni appuntamento = si suppone che un cliente sia mediamente acquisti 1 proposte per appuntamento

Ristrutturazione dello schema E-R

1) Analisi delle ridondanze:

- Si aggiunge una relazione molti e molti Cliente_Acquista, tra Cliente e Proposta, apportando delle modifiche ai costi delle seguenti operazioni:

OP28			
Costo operazione = 3		**Costo totale = 108.000/giorno**	
Concetto	**Costrutto**	**Accessi**	**Tipo**
Cliente	E	1	L
Cliente_Acquista	R	N	L
Proposta	E	N	L

N = numero di proposte commerciali acquistate dal cliente = si suppone che un cliente acquista mediamente 1 proposta all'anno

OP6			
Costo operazione = 8		**Costo totale = 96.000/giorno**	
Concetto	**Costrutto**	**Accessi**	**Tipo**
Cliente	E	1	L
Appuntamento	E	1	L
Acquista	R	1	S
Proposta	E	2	L
Cliente_Acquista	R	1	S

Di conseguenza l'operazione OP28 passa da un costo totale di 180.000/giorno, ad un costo totale di 108.000/giorno, con una diminuizione di 72.000 accessi al giorno. Mentre l'operazione OP6 passa da un costo totale di 48.000/giorno ad un costo totale di 96.000/giorno, con un aumento di 48.000 accessi al giorno.

Con l'aggiunta della ridondanza, si ottiene così una diminuizione di 24.000 accessi al giorno.

- Si era pensato di fare un discorso simile per quanto riguarda le proposte di interesse di un cliente. Aggiungere, quindi, una relazione molti e molti Cliente_Accetta, tra Cliente e Proposta., apportando delle modifiche ai costi delle seguenti operazioni:

OP5			
Costo operazione = 8		**Costo totale = 384.000/giorno**	
Concetto	**Costrutto**	**Accessi**	**Tipo**
Cliente	E	1	L
Contatto	E	1	L
Accetta	R	1	S
Proposta	E	2	L
Cliente_Accetta	R	1	S

OP14			
Costo operazione = 5		**Costo totale = 60.000 /giorno**	
Concetto	**Costrutto**	**Accessi**	**Tipo**
Cliente	E	1	L
Cliente_Accetta	R	N	L
Proposte	E	N	L

N = numero medio delle proposte a cui un cliente mostra interesse per ogni contatto = si suppone che un cliente sia mediamente interessato a 2 proposte per contatto

Di conseguenza l'operazione OP5 passa da un costo totale di 192.000/giorno, ad un costo totale di 384.000/giorno, con un aumento di 192.000 accessi al giorno. Mentre l'operazione OP14 passa da un costo totale di 84.000/giorno ad un costo totale di 60.000/giorno, con una diminuzione di 24.000 accessi al giorno.

Con l'aggiunta della ridondanza, si ottiene così un aumento di 168.000 accessi al giorno. Quindi si decide di non aggiungere la ridondanza.

2) Eliminazione delle generalizzazioni

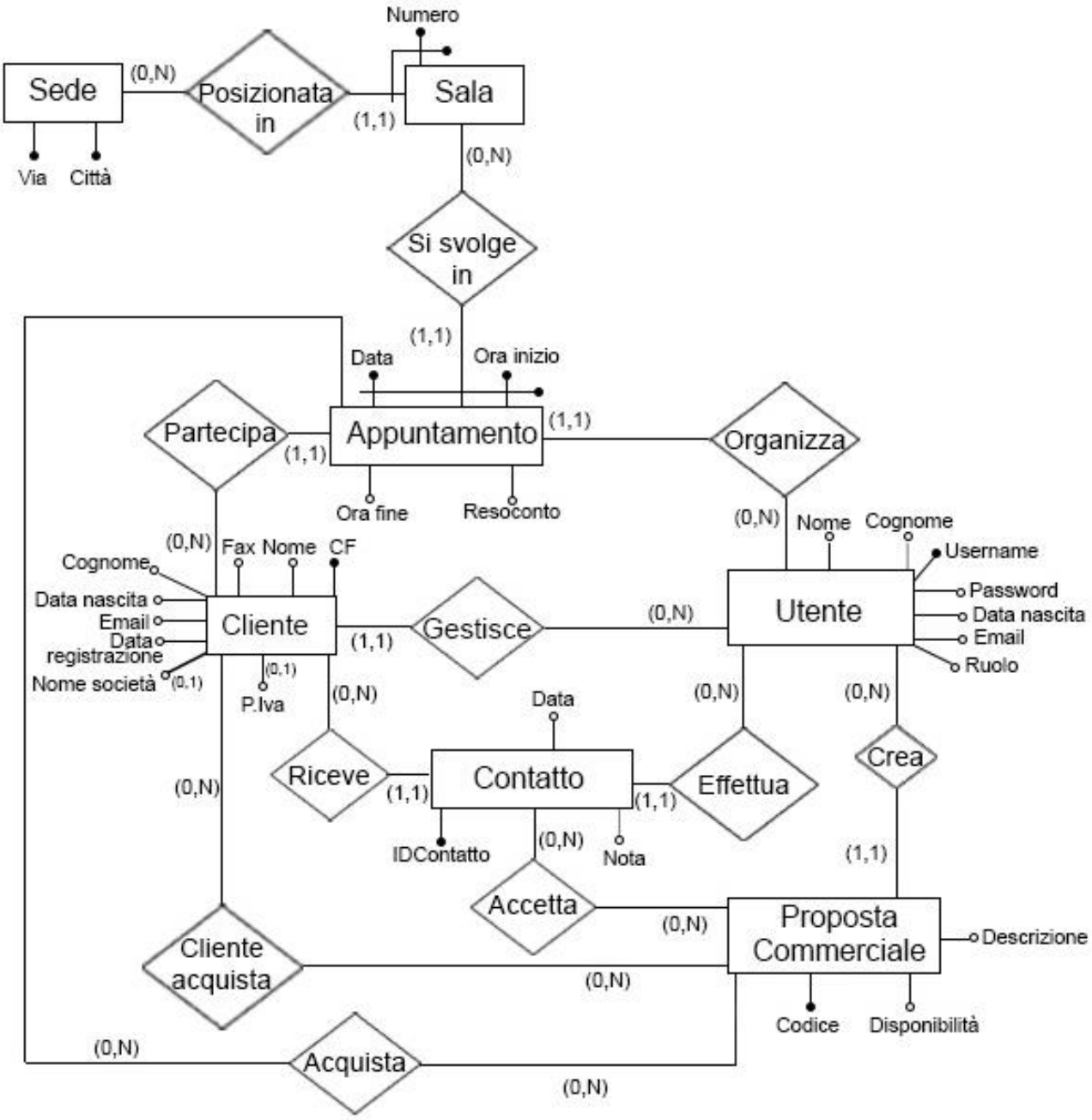
- La generalizzazione riguardo gli utenti viene eliminate accorpando I figli (operatore, manager, F_commerciale) nel padre (utente) aggiungendo un attributo "ruolo" che indica il tipo di funzionario.
- Il sottoinsieme Società dell'entità Cliente, viene eliminato inserendo gli attributi "Piva" e "nome società" (che precedentemente erano dell'entità Società) all'interno dell'entità Cliente, come attributi opzionali.

  Con questa soluzione si verifica un leggero spreco di memoria, dovuto agli eventuali valori nulli negli attributi Piva e Nome società per i clienti che non sono anche società. Ma si è supposto che i clienti che sono anche società siano molto superiori rispetto ai clienti che non sono società. Inoltre non sono previste operazioni che facciano distinzione tra i due clienti.

3) Scelta degli identificatori primari:

- Cliente: CF
- Utente: username
- Contatto: IDcontatto
- Appuntamento: data, ora inizio
- Proposte: codice
- Sala: numero
- Sede: via, città

Modello E-R ristrutturato:



Trasformazione di attributi e identificatori

Sala è un'entità debole quindi avrà un identificatore esterno della Sede.

Visto sono presenti entità deboli in cascata, l'entità Appuntamento avrà due identificatori esterni: uno per la Sede e uno per la Sala.

Traduzione di entità e associazioni

Clienti (CF, nome, cognome, dataNascita, dataRegistrazione, email, fax, piva, nomeSocietà, operatore)

Utenti (username, password, nomeUtente, cognomeUtente, dataNascitaUtente, emailUtente, ruolo)

Appuntamenti (data, oraInizio, sala, viaSede, cittàSede, oraFine, resoconto, cliente, operatore)

Sale (numero, viaSede, cittàSede)

Sedi (via, città)

Contatti (IDcontatto, nota, data, cliente, operatore)

Proposte (Codice, descrizione, disponibilità, manager)

Accetta (contatto, proposta)

Acquista (data, sala, viaSede, cittàSede, proposta)

Cliente_acquista (cliente, proposta)

Clienti(operatore) ⊆ **Utente**(username)

Appuntamenti(sala) ⊆ **Sale**(numero)

Appuntamenti (viaSede, cittàSede) ⊆ **Sedi** (via, città)

Appuntamenti(cliente) ⊆ **Clienti** (CF)

Appuntamenti(operatore) ⊆ **Utenti**(username)

Sale (viaSede, cittàSede) ⊆ **Sedi** (via, città)

Contatti(cliente) ⊆ **Clienti** (CF)

Contatti(operatore) ⊆ **Utenti**(username)

Proposte(manager) ⊆ **Utenti**(username)

Accetta(contatto) ⊆ **Contatti** (IDContatto)

Accetta(proposta) ⊆ **Proposte** (Codice)

Acquista (data, oraInizio, sala, viaSede, cittàSede) ⊆ **Appuntamenti** (data, oraInizio, sala, viaSede, cittàSede)

Acquista(proposta) ⊆ **Proposte** (Codice)

Cliente_acquista(cliente) ⊆ **Clienti** (CF)

Cliente_acquista(proposta) ⊆ **Proposte** (Codice)

Normalizzazione del modello relazionale

Prima Forma Normale (1NF)

> Richiede che tutti gli attributi dello schema abbiano domini "atomici" (ovvero non siano composti o multivalore)

Lo schema è già in 1NF

Seconda Forma Normale (2NF)

- E' in prima forma normale
- Tutti i suoi attributi non-chiave dipendono dall'intera chiave, cioè non possiede attributi che dipendono soltanto da una parte della chiave.

Lo schema è già in 2NF

Terza Forma Normale (3NF)

- E' in Seconda forma normale
- Tutti gli attributi non-chiave dipendono direttamente dalla chiave, cioè non possiede attributi che dipendono da altri attributi che non sono in chiave

Lo schema è già in 3NF

5. Progettazione fisica

Utenti e privilege

- Operatore

  Privilegi relativi alla tabella Clienti, garantiti tramite le store procedures:

  - **visualizza_clienti_operatore** visualizzare i clienti assegnati all'operatore.
  - **visualizza_cliente** visualizzare le informazioni relative ad un cliente.

  Privilegi relativi alla tabella Contatti, garantiti tramite le store procedures:

  - **inserisci_contatto** Inserire un contatto.
  - **modifica_nota_contatto** Modificare la nota di un contatto.
  - **elimina_nota_contatto** Eliminare la nota di un contatto.
  - **visualizza_contatto** Visualizzare le informazioni relative ad un contatto.
  - **visualizza_contatti_operatore** Visualizza l'elenco dei contatti dell'operatore.
  - **visualizza_contatti_cliente** Visualizza l'elenco dei contatti relativi ad un cliente.

  Privilegi relativi alla tabella Appuntamenti, garantiti tramite le store procedures:

  - **inserisci_appuntamento** Inserire un appuntamento.
  - **modifica_resconto_appuntamento** Modificare il resoconto di un appuntamento.
  - **modifica_dataora_appuntamento** Modificare la data/ora di un appuntamento.
  - **visualizza_appuntamento** Visualizzare le informazioni relative ad un appuntamento.
  - **visualizza_appuntamenti_operatore** Visualizzare l'elenco degli appuntamenti dell'operatore.

  Privilegi relativi alla tabella Accetta, garantiti tramite le store procedures:

  - **inserisci_proposta_accettata** Indicare che un cliente è interessato ad una proposta commerciale.
  - **visualizza_proposte_accettate** Visualizzare le proposte commerciali a cui un cliente è interessato.

Privilegi relativi alla tabella Acquista e Cliente_acquista, garantiti tramite le store procedures:

- **inserisci_proposta_acquistata** Indicare che un cliente ha acquistato una proposta commerciale.
- **visualizza_proposte_acquistate** Visualizzare le proposte commerciali acquistate da un cliente.

Privilegi relativi alla tabella Proposte, garantiti tramite le store procedures:

- **visualizza_proposte_disponibili** Visualizzare le proposte commerciali disponibili.

Privilegi relativi alla tabella Sale, garantiti tramite le store procedures:

- **visualizza_sala_sede** Visualizzare l'elenco delle sale a disposizione con le relative sedi.

- Manager

Privilegi relativi alla tabella Proposte, garantiti tramite le store procedures:

- **inserisci_proposta** Aggiungere una proposta commerciale.
- **modifica_disponibilita_proposta** Modificare la disponibilità di una proposta commerciale.
- **visualizza_proposte** Visualizzare le proposte commerciali.
- **visualizza_proposta** Visualizzare le informazioni relative ad una proposta.

Privilegi relativi alla tabella Utenti, garantiti tramite le store procedures:

- **inserisci_utente** Inserire un utente.
- **visualizza_utenti** Visualizzare l'elenco degli utenti.

Privilegi relativi alla tabella Cliente_acquista, garantiti tramite le store procedures:

- **conta_proposte_acquistate** Visualizzare le proposte che sono state acquistate e quante volte sono state acquistate.

- Funzionario del settore commerciale (f_commerciale)

  Privilegi relativi alla tabella Clienti, garantiti tramite le store procedures:

  - **inserisci_cliente** Inserire un cliente e associarlo ad un operatore.
  - **visualizza_clienti** Visualizzare l'elenco dei clienti.

- Login

  Privilegi relativi alla tabella Utenti, garantiti tramite le store procedures:

  - **Login** Effettuare il login

I vari utenti possono solo eseguire le store procedure previste, in modo da cercare di evitare che possano eseguire operazioni non previste.

Strutture di memorizzazione

Tabella clienti		
Colonna	**Tipo di dato**	**Attributi**[2]
CF	CHAR(16)	PK, NN
nome	VARCHAR(25)	NN
cognome	VARCHAR(25)	NN
data_nascita	DATE	NN
data_registrazione	DATE	NN
email	VARCHAR(45)	NN
fax	VARCHAR(45)	NN
piva	CHAR(11)	
nome_società	VARCHAR(45)	
operatore	VARCHAR(20)	NN

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

Tabella utenti		
Colonna	**Tipo di dato**	**Attributi**
username	VARCHAR(20)	PK, NN
password	VARCHAR(32)	NN
nome_utente	VARCHAR(25)	NN
cognome_utente	VARCHAR(25)	NN
data_nascita_utente	DATE	NN
email_utente	VARCHAR(45)	NN, UQ
ruolo	ENUM('operatore', 'manager', 'f_commerciale')	NN

Tabella contatti		
Colonna	**Tipo di dato**	**Attributi**
IDcontatto	INT	PK, NN, AI
nota	VARCHAR(300)	NN
cliente	CHAR(16)	NN
operatore	VARCHAR(20)	NN
data_contatto	DATE	NN

Tabella proposte		
Colonna	**Tipo di dato**	**Attributi**
codice	VARCHAR(10)	PK, NN
disponibilità	TINYINT(1)	NN
descrizione	VARCHAR(100)	NN
operatore	VARCHAR(20)	NN

Tabella proposte_accettate		
Colonna	**Tipo di dato**	**Attributi**
contatto	INT	PK, NN
proposta	VARCHAR(10)	PK, NN

Tabella appuntamenti		
Colonna	**Tipo di dato**	**Attributi**
data_ora_inizio	DATETIME	PK, NN
sala	VARCHAR(3)	PK, NN
via_sala_sede	VARCHAR(45)	PK, NN
città_sala_sede	VARCHAR(20)	PK, NN
resoconto	VARCHAR(300)	
operatore	VARCHAR(20)	NN
cliente	CHAR(16)	NN
data_ora_fine	DATETIME	NN

Tabella proposte_acquistate		
Colonna	**Tipo di dato**	**Attributi**
data_appuntamento	DATETIME	PK, NN
sala_appuntamento	VARCHAR(3)	PK, NN
via_appuntamento	VARCHAR(45)	PK, NN
città_appuntamento	VARCHAR(20)	PK, NN
proposta	VARCHAR(10)	PK, NN

Tabella sale		
Colonna	**Tipo di dato**	**Attributi**
numero	VARCHAR(3)	PK, NN
via_sede	VARCHAR(45)	PK, NN
città_sede	VARCHAR(20)	PK, NN

Tabella sedi		
Colonna	**Tipo di dato**	**Attributi**
via	VARCHAR(45)	PK, NN
città	VARCHAR(20)	PK, NN

Tabella cliente_acquista		
Colonna	**Tipo di dato**	**Attributi**
cliente	CHAR(16)	PK, NN
proposta	VARCHAR(10)	PK, NN

Indici

Tabella clienti	
Indice PRIMARY	**Tipo[3]:**
CF	PR
Indice operatore_idx	**Tipo:**
operatore	IDX

Tabella utenti	
Indice PRIMARY	**Tipo:**
username	PR
Indice email_UNIQUE	**Tipo:**
email_utente	UQ

Tabella contatti	
Indice PRIMARY	**Tipo:**
IDcontatto	PR
Indice cliente_idx	**Tipo:**
cliente	IDX
Indice operatore_idx	**Tipo:**
operatore	IDX

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

Tabella proposte	
Indice PRIMARY	**Tipo:**
codice	PR
Indice manager_idx	**Tipo:**
manager	IDX

Tabella proposte_accettate	
Indice PRIMARY	**Tipo:**
contatto	PR
proposta	PR
Indice fk_proposta_accettata_idx	**Tipo:**
proposta	IDX

Tabella appuntamenti	
Indice PRIMARY	**Tipo:**
data_ora_inizio	PR
sala	PR
via_sala_sede	PR
città_sala_sede	PR
Indice sala_idx	**Tipo:**
sala	IDX
Indice cliente_idx	**Tipo:**
cliente	IDX
Indice operatore_idx	**Tipo:**
operatore	IDX
Indice fk_sede_idx	**Tipo:**
via_sala_sede	IDX
città_sala_sede	IDX

Tabella proposte_acquistate	
Indice PRIMARY	**Tipo:**
data_appuntamento	PR
sala_appuntamento	PR
via_appuntamento	PR
città_appuntamento	PR
proposta	PR
Indice fk_proposta_appuntamento_idx	**Tipo:**
proposta	IDX

Tabella sale	
Indice PRIMARY	**Tipo:**
numero	PR
via_sede	PR
città_sede	PR
Indice sedeVia_idx	**Tipo:**
via_sede	IDX
città_sede	IDX

Tabella sedi	
Indice PRIMARY	**Tipo:**
via	PR
città	PR

Tabella cliente_acquista	
Indice PRIMARY	**Tipo:**
cliente	PR
proposta	PR
Indice fk_proposta_acquistata_idx	**Tipo:**
proposta	IDX

Trigger

- Controlla cliente gestito

Prima di inserire un nuovo appuntamento che coinvolge un operatore e un cliente, si controlla che l'operatore gestisce tale cliente.

```
CREATE DEFINER = CURRENT_USER TRIGGER `mydb`.`appuntamenti_BEFORE_INSERT` BEFORE INSERT ON `appuntamenti` FOR EACH ROW

BEGIN

    /*controlla se il cliente viene gestito dall'operatore*/

    if NEW.operatore != ( SELECT clienti.operatore FROM clienti WHERE clienti.CF = NEW.cliente ) then

        signal sqlstate '45000' set message_text= "ERRORE Inserimento Appuntamento: il cliente non è gestito dall'operatore selezionato";

    end if;

END
```

- Controlla disponibilità sala - insert

Prima di inserire un nuovo appuntamento in una determinata sala con relativa sede, si controlla che quella sala non sia già occupata, in quella data/ora.

```
CREATE DEFINER = CURRENT_USER TRIGGER `mydb`.`appuntamenti_BEFORE_INSERT_1` BEFORE INSERT ON `appuntamenti` FOR EACH ROW

BEGIN

    /*controlla che la sala sia disponibile in un dato orario*/

    if exists (SELECT * FROM appuntamenti WHERE sala = NEW.sala and via_sala_sede = NEW.via_sala_sede and città_sala_sede = NEW.città_sala_sede and ( (data_ora_inizio between
```

```
NEW.data_ora_inizio and date_add(NEW.data_ora_fine, interval -1 second)) or (data_ora_fine
between date_add(NEW.data_ora_inizio, interval +1 second) and NEW.data_ora_fine) ) ) then

        signal sqlstate '45000' set message_text= "ERRORE Inserimento Appuntamento:
l'orario selezionato non è disponibile";

    end if;

END
```

- Controlla disponibilità operatore - insert

Prima di inserire un nuovo appuntamento che coinvolge un operatore, si controlla che l'operatore non abbia già altri appuntamenti in quella data/ora.

```
CREATE DEFINER = CURRENT_USER TRIGGER
`mydb`.`appuntamenti_BEFORE_INSERT_2` BEFORE INSERT ON `appuntamenti` FOR EACH
ROW

BEGIN

    /*controlla se l'operatore e' disponibile in un dato giorno e orario*/

    if exists ( SELECT * FROM appuntamenti WHERE operatore = NEW.operatore and
((data_ora_inizio between NEW.data_ora_inizio and date_add(NEW.data_ora_fine, interval -1
second) ) or (data_ora_fine between date_add(NEW.data_ora_inizio, interval +1 second) and
NEW.data_ora_fine))) then

        signal sqlstate '45000' set message_text= "ERRORE Inserimento Appuntamento:
l'operatore risulta già impegnato nell'orario selezionato";

    end if;

END
```

- Controlla sala

Prima di inserire un nuovo appuntamento in una determinata sala con relativa sede, si controlla che la sala specificata si trovi effettivamente nella sede specificata.

```
CREATE DEFINER = CURRENT_USER TRIGGER
`mydb`.`appuntamenti_BEFORE_INSERT_4` BEFORE INSERT ON `appuntamenti` FOR EACH
ROW

BEGIN

    /*controlla se la sala e' presente nella sede selezionata*/

    if not exists(SELECT * FROM sale WHERE numero = NEW.sala and via_sede =
NEW.via_sala_sede and città_sede = NEW.città_sala_sede) then

        signal sqlstate '45000' set message_text= "ERRORE Inserimento Appuntamento: la
sala selezionata non è presente all'interno della sede selezionata";

    end if;

END
```

- Controlla data – update

Dell'appuntamento è possibile modificare: data/ora di inizio/fine e il resoconto.

Nel caso in cui si stia modificando il resoconto, la chiave non viene modificata, quindi l'update andrà sempre a buon fine.

Nel caso in cui si stia modificando la data/ora: prima di modificare un appuntamento si controlla che la nuova data/ora non sia antecedente a quella odierna e si controlla che l'appuntamento duri al massimo un'ora.

```
CREATE DEFINER = CURRENT_USER TRIGGER `mydb`.`appuntamenti_BEFORE_UPDATE`
BEFORE UPDATE ON `appuntamenti` FOR EACH ROW

BEGIN

    /*controlla se la data dell'appuntamento e' antecedente a quella odierna, e se la data di fine
appuntamento risulta essere al massimo di un'ora dopo la data di inizio*/

    if not (OLD.sala = NEW.sala and OLD.via_sala_sede = NEW.via_sala_sede and
OLD.città_sala_sede = NEW.città_sala_sede and  OLD.data_ora_inizio = NEW.data_ora_inizio and
OLD.data_ora_fine = NEW.data_ora_fine) then
```

```
        if NEW.data_ora_inizio < now() or NEW.data_ora_fine > (NEW.data_ora_inizio +
INTERVAL 1 HOUR) then
            signal sqlstate '45000' set message_text= "ERRORE Aggiornamento
Appuntamento: la data inserita è errata";
        end if;
    end if;
END
```

- Controlla disponibilità sala – update

Dell'appuntamento è possibile modificare: data/ora di inizio/fine e il resoconto.

Nel caso in cui si sta modificando il resoconto, la chiave non viene modificata, quindi l'update andrà sempre a buon fine.

Nel caso in cui si stia modificando la data/ora: prima di modificare un appuntamento si controlla che non si stia modificano un appuntamento già avvenuto, e si controlla che la sala sia disponibile nella nuova data/ora.

```
CREATE DEFINER = CURRENT_USER TRIGGER
`mydb`.`appuntamenti_BEFORE_UPDATE_1` BEFORE UPDATE ON `appuntamenti` FOR EACH
ROW
BEGIN
    /*controlla che la sala sia disponibile in un dato orario*/
    if not (OLD.sala = NEW.sala and OLD.via_sala_sede = NEW.via_sala_sede and
OLD.città_sala_sede = NEW.città_sala_sede and  OLD.data_ora_inizio = NEW.data_ora_inizio and
OLD.data_ora_fine = NEW.data_ora_fine) then
        if ( OLD.data_ora_inizio < now() )then
            signal sqlstate '45000' set message_text= "ERRORE Aggiornamento
Appuntamento: non è possibile modificare l'orario di un appuntamento già avvenuto";
        end if;
```

```
if ( ( NEW.data_ora_inizio > OLD.data_ora_inizio and NEW.data_ora_inizio < OLD.data_ora_fine ) or (NEW.data_ora_fine > OLD.data_ora_inizio and NEW.data_ora_fine < OLD.data_ora_fine ) ) then

if ( ( SELECT count(*) FROM appuntamenti WHERE sala = NEW.sala and via_sala_sede = NEW.via_sala_sede and città_sala_sede = NEW.città_sala_sede and ( (data_ora_inizio between NEW.data_ora_inizio and date_add(NEW.data_ora_fine, interval -1 second)) or (data_ora_fine between date_add(NEW.data_ora_inizio, interval +1 second) and NEW.data_ora_fine) ) ) > 1 ) then

signal sqlstate '45000' set message_text= "ERRORE Aggiornamento Appuntamento: l'orario selezionato non è disponibile";

end if;

elseif exists (SELECT * FROM appuntamenti WHERE sala = NEW.sala and via_sala_sede = NEW.via_sala_sede and città_sala_sede = NEW.città_sala_sede and ( (data_ora_inizio between NEW.data_ora_inizio and date_add(NEW.data_ora_fine, interval -1 second)) or (data_ora_fine between date_add(NEW.data_ora_inizio, interval +1 second) and NEW.data_ora_fine) ) )then

signal sqlstate '45000' set message_text= "ERRORE Aggiornamento Appuntamento: l'orario selezionato non è disponibile";

end if;

end if;

END
```

- Controlla disponibilità operatore – update

Dell’appuntamento è possibile modificare: data/ora di inizio/fine e il resoconto.

Nel caso in cui si stia modificando il resoconto, la chiave non viene modificata, quindi l’update andrà sempre a buon fine.

Nel caso in cui si stia modificando la data/ora: prima di modificare un appuntamento si controlla che non si stia modificano un appuntamento già avvenuto, e si controlla che l'operatore non abbia già altri appuntamenti nella nuova data/ora.

```
CREATE DEFINER = CURRENT_USER TRIGGER
`mydb`.`appuntamenti_BEFORE_UPDATE_2` BEFORE UPDATE ON `appuntamenti` FOR EACH
ROW

BEGIN

	/*controlla se l'operatore e' disponibile in un dato giorno e orario*/

	if not (OLD.sala = NEW.sala and OLD.via_sala_sede = NEW.via_sala_sede and
OLD.città_sala_sede = NEW.città_sala_sede and  OLD.data_ora_inizio = NEW.data_ora_inizio and
OLD.data_ora_fine = NEW.data_ora_fine) then

		if ( OLD.data_ora_inizio < now() )then

			signal sqlstate '45000' set message_text= "ERRORE Aggiornamento
Appuntamento: non è possibile modificare l'orario di un appuntamento già avvenuto";

		end if;

		if ( ( NEW.data_ora_inizio > OLD.data_ora_inizio and NEW.data_ora_inizio <
OLD.data_ora_fine ) or (NEW.data_ora_fine > OLD.data_ora_inizio and NEW.data_ora_fine <
OLD.data_ora_fine ) ) then

			if ( ( SELECT count(*) FROM appuntamenti WHERE operatore =
NEW.operatore and ((data_ora_inizio between NEW.data_ora_inizio and
date_add(NEW.data_ora_fine, interval -1 second)) or (data_ora_fine between
date_add(NEW.data_ora_inizio, interval +1 second) and NEW.data_ora_fine) ) ) > 1 )then

				signal sqlstate '45000' set message_text= "ERRORE Aggiornamento
Appuntamento: l'operatore risulta già impegnato nell'orario selezionato";

			end if;

		elseif exists (SELECT * FROM appuntamenti WHERE operatore = NEW.operatore
and ((data_ora_inizio between NEW.data_ora_inizio and date_add(NEW.data_ora_fine, interval -1
second)) or (data_ora_fine between date_add(NEW.data_ora_inizio, interval +1 second) and
NEW.data_ora_fine))) then
```

```
                signal sqlstate '45000' set message_text= "ERRORE Aggiornamento
Appuntamento: l'operatore risulta già impegnato nell'orario selezionato";
            end if;
        end if;
END
```

- Controlla disponibilità proposta

Prima di inserire l'acquisto di una proposta da parte di un cliente, si controlla che quella proposta sia disponibile.

```
CREATE DEFINER = CURRENT_USER TRIGGER
`mydb`.`cliente_acquista_BEFORE_INSERT` BEFORE INSERT ON `cliente_acquista` FOR
EACH ROW
BEGIN
    /*controlla che la proposta sia disponibile*/
    if (SELECT disponibilità FROM proposte WHERE codice = NEW.proposta) = 0 then
        signal sqlstate '45000' set message_text= "ERRORE Inserimento Proposta Acquistata
Shortcut: la proposta indicata non è disponibile";
    end if;
END
```

- Controlla cliente gestito

Prima di inserire un nuovo contatto avvenuto tra operatore e cliente si controlla che l'operatore gestisce tale cliente.

```
CREATE DEFINER = CURRENT_USER TRIGGER `mydb`.`contatti_BEFORE_INSERT`
BEFORE INSERT ON `contatti` FOR EACH ROW
BEGIN
    /*controlla che il cliente sia gestito dall'operatore*/
```

if NEW.operatore != (SELECT clienti.operatore FROM clienti WHERE clienti.CF = NEW.cliente) then

signal sqlstate '45000' set message_text= "ERRORE Inserimenti Contatto: il cliente non è gestito da tale operatore";

end if;

END

- Controlla disponibilità proposta

Prima di inserire l'accettazione di una proposta da parte di un cliente, si controlla che quella proposta sia disponibile.

CREATE DEFINER = CURRENT_USER TRIGGER `mydb`.`proposte_accettate_BEFORE_INSERT` BEFORE INSERT ON `proposte_accettate` FOR EACH ROW

BEGIN

/*controlla che la proposta sia disponibile*/

if (SELECT disponibilità FROM proposte WHERE codice = NEW.proposta) = 0 then

signal sqlstate '45000' set message_text= "ERRORE Inserimento Proposta Accettata: la proposta indicata non è disponibile";

end if;

END

- Controlla disponibilità proposta

Prima di inserire l'acquisto di una proposta da parte di un cliente, si controlla che quella proposta sia disponibile.

CREATE DEFINER = CURRENT_USER TRIGGER `mydb`.`proposte_acquistate_BEFORE_INSERT` BEFORE INSERT ON `proposte_acquistate` FOR EACH ROW

```
BEGIN
    /*controlla che la proposta sia disponibile*/
    if (SELECT disponibilità FROM proposte WHERE codice = NEW.proposta) = 0 then
        signal sqlstate '45000' set message_text= "ERRORE Inserimento Proposta Acquistata: la proposta indicata non è disponibile";
    end if;
END
```

- Controlla data di nascita

Prima di inserire un nuovo cliente, controlla che la data di nascita indicata sia antecedente alla data odierna.

```
CREATE DEFINER = CURRENT_USER TRIGGER `mydb`.`clienti_BEFORE_INSERT`
BEFORE INSERT ON `clienti` FOR EACH ROW
BEGIN
    if(NEW.data_nascita > now()) then
        signal sqlstate '45000' set message_text= "ERRORE Inserimento Cliente: la data di nascita non è valida";
    end if;
END
```

- Controlla data di nascita

Prima di inserire un nuovo utente, controlla che la data di nascita indicata sia antecedente alla data odierna.

```
CREATE DEFINER = CURRENT_USER TRIGGER `mydb`.`utenti_BEFORE_INSERT` BEFORE INSERT ON `utenti` FOR EACH ROW
BEGIN
```

```
    if(NEW.data_nascita_utente > now()) then
        signal sqlstate '45000' set message_text= "ERRORE Inserimento Utente: la data di nascita non è valida";
    end if;
END
```

Eventi

Non è stato implementato nessun evento.

Viste

Non è stata inserita alcuna vista.

Stored Procedures e transazioni

- Conta_proposte_acquistate
  - Procedura per visualizzare le proposte che sono state acquistate (da almeno un cliente), inoltre conta e mostra il numero di volte che sono state acquistate.
  - Il livello di isolamento scelto è Read Committed per evitare di leggere/contare proposte acquistate che non sono state committate.

```
CREATE PROCEDURE `conta_proposte_acquistate` ()
BEGIN
    declare exit handler for sqlexception
    begin
        rollback;
        resignal;
    end;
    SET transaction isolation level read committed;
    SET transaction read only;
```

```
    start transaction;
        SELECT proposta as 'CODICE', count(*) as 'NUMERO DI ACQUISTI'
        FROM cliente_acquista
        GROUP BY proposta;
    commit;
END
```

- Elimina_nota
  - Procedura che permette di eliminare la nota di un contatto, solo se il contatto esiste.
  - Il livello di isolamento scelto è Read Committed per evitare di leggere contatti che non sono ancora stati committati.

```
CREATE PROCEDURE `elimina_nota` (in id INT)
BEGIN
    declare exit handler for sqlexception
    begin
        rollback;
        resignal;
    end;
    set transaction isolation level read committed;
    start transaction;
        if not exists ( SELECT * FROM contatti WHERE IDcontatto = id) then
            signal sqlstate '45000' set message_text= "ERRORE Elimina Nota: il contatto non esiste";
        end if;
        UPDATE contatti SET nota = " WHERE IDcontatto = id;
```

```
    commit;
END
```

- Inserisci_appuntamento
  - Procedura che permette l'inserimento di un nuovo appuntamento solo se la data/ora indicata è successiva a quella odierna e se l'appuntamento dura massimo un'ora.
    I controlli per verificare che la sala sia disponibile in un dato giorno/orario, che l'operatore sia disponibile in un dato giorno/orario sono effettuati tramite trigger.

```
CREATE PROCEDURE `inserisci_appuntamento` (in data_ora_in DATETIME, in numero_sala INT, in via_sede VARCHAR(45), in città_sede VARCHAR(20), in operatore VARCHAR(20), in cliente CHAR(16),in data_ora_fin DATETIME)
BEGIN
        if data_ora_in < now() or data_ora_fin > (data_ora_in + INTERVAL 1 HOUR) then
                signal sqlstate '45000' set message_text= "ERRORE Inserimento Appuntamento: la data inserita è errata";
        else
                INSERT INTO appuntamenti(data_ora_inizio, sala, via_sala_sede, città_sala_sede, operatore, cliente,data_ora_fine) VALUES (data_ora_in, numero_sala, via_sede, città_sede, operatore, cliente, data_ora_fin);
        end if;
END
```

- Inserisci_cliente
  - Procedura che permette l'inserimento di un nuovo cliente e lo assegna automaticamente all'operatore che gestisce il minor numero di clienti.
  - Il livello di isolamento scelto è Repeatable read poiché si vuole mantenere il lock in lettura fino alla fine della transazione, in modo che nel caso in cui più funzionari commerciali

stiano inserendo nuovi clienti concorrentemente, il minimo venga scelgo in maniera coerente.

```
CREATE PROCEDURE `inserisci_cliente` (in cf CHAR(16), in nome VARCHAR(25), in cognome VARCHAR(25), data_nascita DATE, email VARCHAR(45), in fax VARCHAR(45), in piva CHAR(11), in nome_società VARCHAR(45))
BEGIN
    declare operatore_selezionato VARCHAR(20);
    declare exit handler for sqlexception
    begin
        rollback;
        resignal;
    end;
    SET transaction isolation level repeatable read;
    start transaction;
        SELECT operatore into operatore_selezionato
        FROM clienti
        GROUP BY operatore
        ORDER BY count(*)
        LIMIT 1;
        INSERT INTO clienti(cf, nome, cognome, data_nascita, data_registrazione, email, fax, piva, nome_società, operatore) VALUES (cf, nome, cognome, data_nascita, now(), email, fax, piva, nome_società, operatore_selezionato);
   commit;
END
```

- Inserisci_contatto
    - Procedura che permette di inserire un contatto, solo se la data è antecedente o uguale alla data odierna. Questo poiché gli operatori non possono inserire contatti futuri, ma soltanto contatti già avvenuti.

```
CREATE PROCEDURE `inserisci_contatto` (in var_nota VARCHAR(300), in var_cliente CHAR(16), in var_operatore VARCHAR(20), in var_data_contatto DATE)

BEGIN

    if var_data_contatto <= curdate()  then

        INSERT INTO contatti(nota, cliente, operatore, data_contatto) VALUES (var_nota, var_cliente, var_operatore, var_data_contatto);

    else

        signal sqlstate '45000' set message_text= "ERRORE Inserimento Contatto: la data inserita è errata";

    end if;

END
```

- Inserisci_proposta

```
CREATE PROCEDURE `inserisci_proposta` (in proposta VARCHAR(10), in descrizione VARCHAR(300), in manager VARCHAR(20))

BEGIN

    INSERT INTO proposte(codice, disponibilità, descrizione, manager) VALUES (proposta, 1, descrizione, manager);

END
```

- Inserisci_proposta_accettata
  - Procedura che permette di indicare che durante un determinato contatto è stata accettata una proposta. Indica quindi che il relativo cliente è interessato a quella proposta commerciale.
    Il controllo che la proposta sia disponibile, viene implementato tramite trigger.

```
CREATE PROCEDURE `inserisci_proposta_accettata` (in IDcontatto INT, in IDproposta
VARCHAR(10))

BEGIN

    INSERT INTO proposte_accettate(contatto, proposta) VALUES (IDcontatto, IDproposta);

END
```

- Inserisci_proposta_acquistata
  - Procedura che permette di indicare che durante un determinato appuntamento è stata acquistata una proposta. Indica quindi che il relativo cliente ha acquistata quella proposta commerciale. La proposta viene inserita solo se l'appuntamento è già avvenuto.
    Il controllo che la proposta sia disponibile, viene implementato tramite trigger.
  - Si utilizza una transazione per sfruttare la strategia "all or nothing".

```
CREATE PROCEDURE `inserisci_proposta_acquistata` (in data_app DATETIME, in sala INT, in
via VARCHAR(45), in città VARCHAR(20), in proposta_acquistata VARCHAR(10), in
cliente_acquista CHAR(16) )

BEGIN

    declare exit handler for sqlexception

    begin

        rollback;

        resignal;

    end;

    SET transaction isolation level read uncommitted;

    start transaction;
```

if (now() < data_app) then

signal sqlstate '45000' set message_text= "ERRORE Inserimento Proposta Acquistata: l'appuntamento non è ancora avvenuto";

end if;

if not exists(SELECT * FROM appuntamenti WHERE data_ora_inizio = data_app and sala = sala and via_sala_sede = via and città_sala_sede = città and cliente = cliente_acquista) then

signal sqlstate '45000' set message_text= "ERRORE Inserimento Proposta Acquistata: non esiste questo appuntamento per il cliente indicato";

end if;

INSERT INTO proposte_acquistate(data_appuntamento, sala_appuntamento, via_appuntamento, città_appuntamento, proposta) VALUES (data_app, sala, via, città, proposta_acquistata);

INSERT INTO cliente_acquista(cliente, proposta) VALUES (cliente_acquista, proposta_acquistata);

commit;

END

- Inserisci_utente
  - Procedura che permette di inserire un nuovo utente. Si effettuano insert diversi a seconda del ruolo che viene specificato.

CREATE PROCEDURE `inserisci_utente` (in username VARCHAR(20), in pssw VARCHAR(32), in nome_utente VARCHAR(25), in cognome_utente VARCHAR(25), in data_nascita_utente DATE, in email_utente VARCHAR(45), in ruolo CHAR(1))

BEGIN

if (ruolo = 'O') then

INSERT INTO utenti(username, password, nome_utente, cognome_utente, data_nascita_utente, email_utente, ruolo) values (username, md5(pssw), nome_utente, cognome_utente, data_nascita_utente, email_utente, 'operatore');

elseif (ruolo = 'M') then

INSERT INTO utenti(username, password, nome_utente, cognome_utente, data_nascita_utente, email_utente, ruolo) values (username, md5(pssw), nome_utente, cognome_utente, data_nascita_utente, email_utente, 'manager');

elseif (ruolo = 'F') then

INSERT INTO utenti(username, password, nome_utente, cognome_utente, data_nascita_utente, email_utente, ruolo) values (username, md5(pssw), nome_utente, cognome_utente, data_nascita_utente, email_utente, 'f_commerciale');

else

signal sqlstate '45000' set message_text= "ERRORE Inserimento Utente: la tipologia di utente selezionata non esiste";

end if;

END

- Login

CREATE PROCEDURE `login` (in var_username VARCHAR(20), in var_password VARCHAR(32), out var_ruolo INT)

BEGIN

declare var_utente_ruolo ENUM('operatore','manager','f_commerciale');

SELECT ruolo FROM utenti WHERE username = var_username and password = md5(var_password) INTO var_utente_ruolo;

if var_utente_ruolo = 'operatore' then

set var_ruolo = 1;

elseif var_utente_ruolo = 'manager' then

```
        set var_ruolo = 2;
    elseif var_utente_ruolo = 'f_commerciale' then
        set var_ruolo = 3;
    else
        set var_ruolo = 4;
    end if;
END
```

- Modifica_dataora_appuntamento
  - Procedura che permette di modificare la data/ora di un appuntamento, solo se l'appuntamento indicato esiste.
    I controlli per verificare che la sala sia disponibile in un dato giorno/orario, che l'operatore sia disponibile in un dato giorno/orario e che la data dell'appuntamento sia valida, sono effettuati tramite trigger.
  - Il livello di isolamento scelto è Repeatable Read poiché si vuole mantenere il lock in lettura fino alla fine della transazazione, in modo da evitare che su aggiornamenti concorrenti il primo aggiornamento vada a buon fine, mentre il secondo abbiamo un finto successo.

```
CREATE PROCEDURE `modifica_dataora_appuntamento` (in data_ora DATETIME, in
numero_sala INT, in via_sede VARCHAR(45), in città_sede VARCHAR(20), in
nuova_dataora_inizio DATETIME, in nuova_dataora_fine DATETIME)
BEGIN
    declare exit handler for sqlexception
    begin
        rollback;
        resignal;
    end;
    set transaction isolation level repeatable read;
```

```
start transaction;
    if not exists ( SELECT * FROM appuntamenti WHERE data_ora_inizio = data_ora and sala = numero_sala and via_sala_sede = via_sede and città_sala_sede = città_sede) then
        signal sqlstate '45000' set message_text= "ERRORE Modifica data/ora Appuntamento: l'appuntamento selezionato non esiste";
    end if;
    UPDATE appuntamenti set data_ora_inizio = nuova_dataora_inizio, data_ora_fine = nuova_dataora_fine WHERE data_ora_inizio = data_ora and sala = numero_sala and via_sala_sede = via_sede and città_sala_sede = città_sede;
    commit;
END
```

- modifica_disponibilita_proposta
  - Procedura che permette di modificare la disponibilità di una proposta commerciale, solo se la proposta indicata esiste.
  - Il livello di isolamento scelto è Read Committed per evitare di leggere proposte che non sono state committate.

```
CREATE PROCEDURE `modifica_disponibilita_proposta` (in proposta VARCHAR(10), in disponibilità tinyint(1))
BEGIN
    declare exit handler for sqlexception
    begin
        rollback;
        resignal;
    end;
    set transaction isolation level read committed;
    start transaction;
```

```
        if not exists ( SELECT * FROM proposte WHERE codice = proposta) then
            signal sqlstate '45000' set message_text= "ERRORE Modifica Proposta: la proposta selezionata non esiste!";
        end if;
        UPDATE proposte SET proposte.disponibilità = disponibilità WHERE codice = proposta;
    commit;
END
```

- modifica_nota_contatto
  - Procedura che permette di modificare la nota di un contatto, solo se il contatto esiste.
  - Il livello di isolamento scelto è Read Committed per evitare di leggere contatti che non sono ancora stati committati.

```
CREATE PROCEDURE `modifica_nota_contatto` (in id INT, in nuova_nota VARCHAR(300))
BEGIN
    declare exit handler for sqlexception
    begin
        rollback;
        resignal;
    end;
    set transaction isolation level read committed;
    start transaction;
        if not exists ( SELECT * FROM contatti WHERE IDcontatto = id) then
            signal sqlstate '45000' set message_text= "ERRORE Modifica Nota: il contatto non esiste";
        end if;
```

```
        UPDATE contatti SET nota = nuova_nota WHERE IDcontatto = id;
    commit;
END
```

- modifica_resoconto_appuntamento
  - Procedura che permette di modificare il resoconto di un appuntamento, solo se l'appuntamento esiste ed è già avvenuto.
  - Il livello di isolamento scelto è Read Committed per evitare di leggere appuntamenti che non sono ancora stati committati.

```
CREATE PROCEDURE `modifica_resoconto_appuntamento` (in nuovo_resoconto
VARCHAR(300), in data_ora DATETIME, in numero_sala INT, in via_sede VARCHAR(45), in
città_sede VARCHAR(20))
BEGIN
    declare exit handler for sqlexception
    begin
        rollback;
        resignal;
    end;
    set transaction isolation level read committed;
    start transaction;
        if not exists (SELECT * FROM appuntamenti WHERE data_ora_inizio = data_ora
and sala = numero_sala and via_sala_sede = via_sede and città_sala_sede = città_sede) then
            signal sqlstate '45000' set message_text= "ERRORE Modifica Resoconto:
l'appuntamento non esiste";
        end if;
        if ( now() <  data_ora ) then
```

signal sqlstate '45000' set message_text= "ERRORE Modifica Resoconto: l'appuntamento non è ancora avvenuto";

end if;

UPDATE appuntamenti SET resoconto = nuovo_resoconto WHERE data_ora_inizio = data_ora AND sala = numero_sala AND via_sala_sede = via_sede AND città_sala_sede = città_sede;

commit;

END

- visualizza_appuntamenti_operatore
  - Procedura che permette di visualizzare l'elenco degli appuntamenti di un determinato operatore.
  - Il livello di isolamento scelto è Read Committed per evitare di leggere appuntamenti inseriti/modificati che non sono ancora stati committati.

CREATE PROCEDURE `visualizza_appuntamenti_operatore` (in operatore VARCHAR(20))

BEGIN

declare exit handler for sqlexception

begin

rollback;

resignal;

end;

SET transaction read only;

SET transaction isolation level read committed;

start transaction;

SELECT data_ora_inizio as 'DATA INIZIO' , sala as 'SALA', via_sala_sede as 'VIA', città_sala_sede as 'CITTA\", cliente as 'CLIENTE'

FROM appuntamenti

```
        WHERE appuntamenti.operatore = operatore
        ORDER BY data_ora_inizio;
    commit;
END
```

- visualizza_appuntamento
  - Procedura che permette di visualizzare un determinato appuntamento.
  - Il livello di isolamento scelto è Read Committed per evitare di leggere appuntamenti che non sono ancora stati committati.

```
CREATE PROCEDURE `visualizza_appuntamento` (in data_ora DATETIME, in numero_sala INT, in via_sede VARCHAR(45), in città_sede VARCHAR(20))
BEGIN
    declare exit handler for sqlexception
    begin
        rollback;
        resignal;
    end;
    SET transaction isolation level read committed;
    SET transaction read only;
    start transaction;
        SELECT data_ora_inizio as 'DATA INIZIO' , sala as 'SALA', via_sala_sede as 'VIA', città_sala_sede as 'CITTA\", resoconto as 'RESOCONTO', cliente as 'CLIENTE', data_ora_FINE as 'DATA FINE'
        FROM appuntamenti
        WHERE data_ora_inizio = data_ora AND sala = numero_sala AND via_sala_sede = via_sede AND città_sala_sede = città_sede;
```

```
        commit;
END
```

- Visualizza_cliente
  - Procedura che permette di visualizzare un determinato cliente.
  - Il livello di isolamento scelto è Read Committed per evitare di leggere clienti che non sono ancora stati committati.

```
CREATE PROCEDURE `visualizza_cliente` (in cliente CHAR(16))
BEGIN
        declare exit handler for sqlexception
        begin
                rollback;
                resignal;
        end;
        SET transaction isolation level read committed;
        SET transaction read only;
        start transaction;
                SELECT cf as 'CLIENTE', nome as 'NOME', cognome as 'COGNOME', data_nascita as 'DATA DI NASCITA', data_registrazione as 'DATA DI REGISTRAZIONE', email as 'EMAIL', fax as 'FAX', piva as 'PARTITA IVA', nome_società as 'NOME SOCIETA\"
                FROM clienti
                WHERE CF = cliente;
   commit;
END
```

- Visualizza_clienti
  - Procedura che permette di visualizzare l'elenco di tutti i clienti.
  - Il livello di isolamento scelto è Read Committed per evitare di leggere clienti iscritti che non sono ancora stati committati.

```
CREATE PROCEDURE `visualizza_clienti` ()

BEGIN

    declare exit handler for sqlexception

    begin

        rollback;

        resignal;

    end;

    SET transaction isolation level read committed;

    SET transaction read only;

    start transaction;

        SELECT cf as 'CLIENTE', nome as 'NOME', cognome as 'COGNOME', data_nascita as 'DATA DI NASCITA', data_registrazione as 'DATA DI REGISTRAZIONE', email as 'EMAIL', fax as 'FAX', piva as 'PARTITA IVA', nome_società as 'NOME SOCIETA\" , operatore as 'OPERATORE'

        FROM clienti;

    commit;

END
```

- Visualizza_clienti_operatore
  - Procedura che permette di visualizzare l'elenco dei clienti assegnati ad un operatore.
  - Il livello di isolamento scelto è Read Committed per evitare di leggere clienti assegnati ad un operatore che non sono ancora stati committati.

```
CREATE PROCEDURE `visualizza_clienti_operatore` (in operatore VARCHAR(20))
```

```
BEGIN
    declare exit handler for sqlexception
    begin
        rollback;
        resignal;
    end;
    SET transaction isolation level read committed;
    SET transaction read only;
    start transaction;
        SELECT cf as 'CLIENTE', nome as 'NOME', cognome as 'COGNOME', email as 'EMAIL', nome_società as 'NOME SOCIETA\"
        FROM clienti
        WHERE clienti.operatore = operatore;
    commit;
END
```

- Visualizza_contatti_cliente
  - Procedura che permette di visualizzare l'elenco dei contatti con un determinato cliente.
  - Il livello di isolamento scelto è Read Committed per evitare di leggere contatti inseriti/modificati che non sono ancora stati committati.

```
CREATE PROCEDURE `visualizza_contatti_cliente` (in cf CHAR(16))
BEGIN
    declare exit handler for sqlexception
    begin
        rollback;
```

```
            resignal;
        end;
        SET transaction isolation level read committed;
        SET transaction read only;
        start transaction;
            SELECT IDcontatto as 'CONTATTO',nota as 'NOTA', data_contatto as 'DATA'
            FROM contatti
            WHERE cliente = cf;
        commit;
END
```

- Visualizza_contatti_operatore
  - Procedura che permette di visualizzare l'elenco dei contatti di un determinato operatore.
  - Il livello di isolamento scelto è Read Committed per evitare di leggere contatti inseriti che non sono ancora stati committati.

```
CREATE PROCEDURE `visualizza_contatti_operatore` (in operatore VARCHAR(20))
BEGIN
        declare exit handler for sqlexception
        begin
            rollback;
            resignal;
        end;
        SET transaction isolation level read committed;
        SET transaction read only;
        start transaction;
```

```
        SELECT IDcontatto as 'CONTATTO', cliente as 'CLIENTE', data_contatto as
'DATA'
        FROM contatti
        WHERE contatti.operatore = operatore
        ORDER BY data_contatto DESC;
    commit;
END
```

- Visualizza_contatto
  - Procedura che permette di visualizzare un determinato contatto.
  - Il livello di isolamento scelto è Read Committed per evitare di leggere contatti modificati che non sono ancora stati committati.

```
CREATE PROCEDURE `visualizza_contatto` (in contatto INT)
BEGIN
    declare exit handler for sqlexception
    begin
        rollback;
        resignal;
    end;
    SET transaction isolation level read committed;
    SET transaction read only;
    start transaction;
        SELECT IDcontatto as 'CONTATTO',nota as 'NOTA', cliente as 'CLIENTE',
data_contatto as 'DATA'
        FROM contatti
        WHERE  IDcontatto = contatto;
```

```
        commit;
END
```

- Visualizza_proposta
  - Procedura che permette di visualizzare una determina proposta commerciale.
  - Il livello di isolamento scelto è Read Committed per evitare di leggere proposta modificate che non sono ancora state committate.

```
CREATE PROCEDURE `visualizza_proposta` (in proposta VARCHAR(10))
BEGIN
        declare exit handler for sqlexception
        begin
                rollback;
                resignal;
        end;
        SET transaction isolation level read committed;
        SET transaction read only;
        start transaction;
                SELECT codice as 'CODICE', disponibilità as 'DISPONIBILITA\", descrizione as 'DESCRIZIONE', manager as 'MANAGER'
                FROM proposte
                WHERE codice = proposta;
        commit;
END
```

- Visualizza_proposte
  - Procedura che permette di visualizzare l'elenco delle proposte commerciali.
  - Il livello di isolamento scelto è Read Committed per evitare di leggere proposte inserite/modificate non ancora committate.

```
CREATE PROCEDURE `visualizza_proposte` ()

BEGIN

    declare exit handler for sqlexception

    begin

        rollback;

        resignal;

    end;

    SET transaction isolation level read committed;

    SET transaction read only;

    start transaction;

        SELECT codice as 'CODICE', disponibilità as 'DISPONIBILITA\", descrizione as 'DESCRIZIONE', manager as 'MANAGER'

        FROM proposte;

    commit;

END
```

- Visualizza_proposte_accettate
  - Procedura che permette di visualizzare l'elenco delle proposte commerciali accettate da un determinato cliente. Ovvero le proposte a cui è interessato.
  - Il livello di isolamento scelto è Read Committed per evitare di leggere proposte inserite/modificate non ancora committate.

```
CREATE PROCEDURE `visualizza_proposte_accettate` (in cliente CHAR(16))

BEGIN
```

```
declare exit handler for sqlexception
begin
        rollback;
        resignal;
end;
SET transaction isolation level read committed;
SET transaction read only;
start transaction;
        SELECT codice as 'CODICE', disponibilità as 'DISPONIBILITA\'', descrizione as 'DESCRIZIONE', manager as 'MANAGER'
        FROM proposte as p
                join proposte_accettate as p_a on p.codice = p_a.proposta
                join contatti as c on p_a.contatto = c.IDcontatto
        WHERE c.cliente = cliente;
  commit;
END
```

- Visualizza_proposte_acquistate
  - Procedura che permette di visualizzare l'elenco delle proposte commerciali acquistate da un determinato cliente.
  - Il livello di isolamento scelto è Read Committed per evitare di leggere proposte inserite/modificate non ancora committate.

```
CREATE PROCEDURE `visualizza_proposte_acquistate` (in cliente CHAR(16))
BEGIN
        declare exit handler for sqlexception
        begin
```

```
            rollback;
            resignal;
        end;
        SET transaction isolation level read committed;
        SET transaction read only;
        start transaction;
            SELECT codice as 'CODICE', disponibilità as 'DISPONIBILITA\", descrizione as
'DESCRIZIONE', manager as 'MANAGER'
             FROM proposte JOIN cliente_acquista ON codice = proposta
            WHERE cliente_acquista.cliente = cliente;
        commit;
END
```

- Visualizza_proposte_disponibili
  - Procedura che permette di visualizzare l'elenco delle proposte commerciali attualmente disponibili.
  - Il livello di isolamento scelto è Read Committed per evitare di leggere proposte commerciali inserite/modificate ma non ancora committate.

```
CREATE PROCEDURE `visualizza_proposte_disponibili` ()
BEGIN
        declare exit handler for sqlexception
        begin
            rollback;
            resignal;
        end;
        SET transaction isolation level read committed;
```

```
SET transaction read only;
start transaction;
SELECT codice as 'CODICE', disponibilità as 'DISPONIBILITA\", descrizione as 'DESCRIZIONE', manager as 'MANAGER'
FROM proposte
WHERE disponibilità = 1;
commit;
END
```

- Visualizza_sala_sede
  - Procedura che permette di visualizzare l'elenco delle sale a disposizione con le relative sedi.
  - Il livello di isolamento scelto è Read Committed per evitare di leggere delle sale che non sono state committate.

```
CREATE PROCEDURE `visualizza_sala_sede` ()
BEGIN
declare exit handler for sqlexception
begin
rollback;
resignal;
end;
SET transaction isolation level read committed;
SET transaction read only;
start transaction;
SELECT numero as 'SALA', via_sede as 'VIA', città_sede as 'CITTA\"
FROM sale;
```

```
        commit;
END
```

- Visualizza_utenti
  - Procedura che permette di visualizzare l'elenco degli utenti registrati.
  - Il livello di isolamento scelto è Read Committed per evitare di leggere utenti registrati che non sono ancora stati committati.

```
CREATE PROCEDURE `visualizza_utenti` ()
BEGIN
        declare exit handler for sqlexception
        begin
                rollback;
                resignal;
        end;
        SET transaction isolation level read committed;
        SET transaction read only;
        start transaction;
                SELECT username as 'USERNAME', nome_utente as 'NOME', cognome_utente as 'COGNOME', data_nascita_utente as 'DATA DI NASCITA', email_utente as 'EMAIL', ruolo as 'RUOLO'
                FROM utenti;
    commit;
END
```

Appendice: Implementazione

Codice SQL per instanziare il database

```
SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0;

SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS,
FOREIGN_KEY_CHECKS=0;

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


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

-- Schema mydb

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

DROP SCHEMA IF EXISTS `mydb` ;


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

-- Schema mydb

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

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

USE `mydb` ;


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

-- Table `mydb`.`utenti`

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

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

```
CREATE TABLE IF NOT EXISTS `mydb`.`utenti` (
  `username` VARCHAR(20) NOT NULL,
  `password` VARCHAR(32) NOT NULL,
  `nome_utente` VARCHAR(25) NOT NULL,
  `cognome_utente` VARCHAR(25) NOT NULL,
  `data_nascita_utente` DATE NOT NULL,
  `email_utente` VARCHAR(45) NOT NULL,
  `ruolo` ENUM('operatore', 'manager', 'f_commerciale') NOT NULL,
  PRIMARY KEY (`username`),
  UNIQUE INDEX `email_UNIQUE` (`email_utente` ASC) VISIBLE)
ENGINE = InnoDB;


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

CREATE TABLE IF NOT EXISTS `mydb`.`clienti` (
  `CF` CHAR(16) NOT NULL,
  `nome` VARCHAR(25) NOT NULL,
  `cognome` VARCHAR(25) NOT NULL,
  `data_nascita` DATE NOT NULL,
```

```
  `data_registrazione` DATE NOT NULL,
  `email` VARCHAR(45) NOT NULL,
  `fax` VARCHAR(45) NOT NULL,
  `piva` CHAR(11) NULL,
  `nome_società` VARCHAR(45) NULL,
  `operatore` VARCHAR(20) NOT NULL,
  PRIMARY KEY (`CF`),
  INDEX `operatore_idx` (`operatore` ASC) VISIBLE,
  CONSTRAINT `fk_operatore_cliente`
    FOREIGN KEY (`operatore`)
    REFERENCES `mydb`.`utenti` (`username`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;



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

CREATE TABLE IF NOT EXISTS `mydb`.`contatti` (
  `IDcontatto` INT NOT NULL AUTO_INCREMENT,
  `nota` VARCHAR(300) NOT NULL,
```

```
  `cliente` CHAR(16) NOT NULL,
  `operatore` VARCHAR(20) NOT NULL,
  `data_contatto` DATE NOT NULL,
  PRIMARY KEY (`IDcontatto`),
  INDEX `cliente_idx` (`cliente` ASC) VISIBLE,
  INDEX `operatore_idx` (`operatore` ASC) INVISIBLE,
  CONSTRAINT `fk_cliente_contatto`
    FOREIGN KEY (`cliente`)
    REFERENCES `mydb`.`clienti` (`CF`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_operatore_contatto`
    FOREIGN KEY (`operatore`)
    REFERENCES `mydb`.`utenti` (`username`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;



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

```
CREATE TABLE IF NOT EXISTS `mydb`.`proposte` (
  `codice` VARCHAR(10) NOT NULL,
  `disponibilità` TINYINT(1) NOT NULL,
  `descrizione` VARCHAR(100) NOT NULL,
  `manager` VARCHAR(20) NOT NULL,
  PRIMARY KEY (`codice`),
  INDEX `manager_idx` (`manager` ASC) VISIBLE,
  CONSTRAINT `fk_manager`
    FOREIGN KEY (`manager`)
    REFERENCES `mydb`.`utenti` (`username`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;



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

CREATE TABLE IF NOT EXISTS `mydb`.`proposte_accettate` (
  `contatto` INT NOT NULL,
  `proposta` VARCHAR(10) NOT NULL,
  PRIMARY KEY (`contatto`, `proposta`),
```

```
  INDEX `fk_proposta_accettata_idx` (`proposta` ASC) VISIBLE,
  CONSTRAINT `fk_contatto`
    FOREIGN KEY (`contatto`)
    REFERENCES `mydb`.`contatti` (`IDcontatto`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_proposta_accettata`
    FOREIGN KEY (`proposta`)
    REFERENCES `mydb`.`proposte` (`codice`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;



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

CREATE TABLE IF NOT EXISTS `mydb`.`sedi` (
  `via` VARCHAR(45) NOT NULL,
  `città` VARCHAR(20) NOT NULL,
  PRIMARY KEY (`via`, `città`))
ENGINE = InnoDB;
```

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

CREATE TABLE IF NOT EXISTS `mydb`.`sale` (
  `numero` VARCHAR(3) NOT NULL,
  `via_sede` VARCHAR(45) NOT NULL,
  `città_sede` VARCHAR(20) NOT NULL,
  PRIMARY KEY (`numero`, `via_sede`, `città_sede`),
  INDEX `sedeVia_idx` (`via_sede` ASC, `città_sede` ASC) VISIBLE,
  CONSTRAINT `fk_sede_sala`
    FOREIGN KEY (`via_sede` , `città_sede`)
    REFERENCES `mydb`.`sedi` (`via` , `città`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `mydb`.`appuntamenti`
-- -----------------------------------------------------
```

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


CREATE TABLE IF NOT EXISTS `mydb`.`appuntamenti` (
  `data_ora_inizio` DATETIME NOT NULL,
  `sala` VARCHAR(3) NOT NULL,
  `via_sala_sede` VARCHAR(45) NOT NULL,
  `città_sala_sede` VARCHAR(20) NOT NULL,
  `resoconto` VARCHAR(300) NULL,
  `operatore` VARCHAR(20) NOT NULL,
  `cliente` CHAR(16) NOT NULL,
  `data_ora_fine` DATETIME NOT NULL,
  PRIMARY KEY (`data_ora_inizio`, `sala`, `via_sala_sede`, `città_sala_sede`),
  INDEX `sala_idx` (`sala` ASC) INVISIBLE,
  INDEX `cliente_idx` (`cliente` ASC) VISIBLE,
  INDEX `operatore_idx` (`operatore` ASC) VISIBLE,
  INDEX `fk_sede_idx` (`via_sala_sede` ASC, `città_sala_sede` ASC) INVISIBLE,
  CONSTRAINT `fk_sala`
    FOREIGN KEY (`sala`)
    REFERENCES `mydb`.`sale` (`numero`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_cliente_app`
    FOREIGN KEY (`cliente`)
    REFERENCES `mydb`.`clienti` (`CF`)
```

```
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_operatore_app`
    FOREIGN KEY (`operatore`)
    REFERENCES `mydb`.`utenti` (`username`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_sede`
    FOREIGN KEY (`via_sala_sede` , `città_sala_sede`)
    REFERENCES `mydb`.`sedi` (`via` , `città`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;



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

CREATE TABLE IF NOT EXISTS `mydb`.`proposte_acquistate` (
  `data_appuntamento` DATETIME NOT NULL,
  `sala_appuntamento` VARCHAR(3) NOT NULL,
  `via_appuntamento` VARCHAR(45) NOT NULL,
```

```
  `città_appuntamento` VARCHAR(20) NOT NULL,
  `proposta` VARCHAR(10) NOT NULL,
  PRIMARY KEY (`data_appuntamento`, `sala_appuntamento`, `via_appuntamento`,
`città_appuntamento`, `proposta`),
  INDEX `fk_proposta_appuntamento_idx` (`proposta` ASC) INVISIBLE,
  CONSTRAINT `fk_appuntamento`
    FOREIGN KEY (`data_appuntamento` , `sala_appuntamento` , `via_appuntamento` ,
`città_appuntamento`)
    REFERENCES `mydb`.`appuntamenti` (`data_ora_inizio` , `sala` , `via_sala_sede` ,
`città_sala_sede`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_proposta_appuntamento`
    FOREIGN KEY (`proposta`)
    REFERENCES `mydb`.`proposte` (`codice`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


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

```
CREATE TABLE IF NOT EXISTS `mydb`.`cliente_acquista` (
  `cliente` CHAR(16) NOT NULL,
  `proposta` VARCHAR(10) NOT NULL,
  PRIMARY KEY (`cliente`, `proposta`),
  INDEX `fk_proposta_acquistata_idx` (`proposta` ASC) INVISIBLE,
  CONSTRAINT `fk_cliente_acquista`
    FOREIGN KEY (`cliente`)
    REFERENCES `mydb`.`clienti` (`CF`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_proposta_acquistata`
    FOREIGN KEY (`proposta`)
    REFERENCES `mydb`.`proposte` (`codice`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
USE `mydb` ;


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

```
GRANT EXECUTE ON procedure `mydb`.`visualizza_clienti_operatore` TO 'operatore';

GRANT EXECUTE ON procedure `mydb`.`visualizza_cliente` TO 'operatore';

GRANT EXECUTE ON procedure `mydb`.`inserisci_contatto` TO 'operatore';

GRANT EXECUTE ON procedure `mydb`.`modifica_nota_contatto` TO 'operatore';

GRANT EXECUTE ON procedure `mydb`.`visualizza_contatto` TO 'operatore';

GRANT EXECUTE ON procedure `mydb`.`inserisci_appuntamento` TO 'operatore';

GRANT EXECUTE ON procedure `mydb`.`modifica_resoconto_appuntamento` TO 'operatore';

GRANT EXECUTE ON procedure `mydb`.`modifica_dataora_appuntamento` TO 'operatore';

GRANT EXECUTE ON procedure `mydb`.`visualizza_appuntamento` TO 'operatore';

GRANT EXECUTE ON procedure `mydb`.`visualizza_appuntamenti_operatore` TO 'operatore';

GRANT EXECUTE ON procedure `mydb`.`inserisci_proposta_accettata` TO 'operatore';

GRANT EXECUTE ON procedure `mydb`.`visualizza_proposte_accettate` TO 'operatore';

GRANT EXECUTE ON procedure `mydb`.`inserisci_proposta_acquistata` TO 'operatore';

GRANT EXECUTE ON procedure `mydb`.`visualizza_proposte_acquistate` TO 'operatore';

GRANT EXECUTE ON procedure `mydb`.`visualizza_proposte_disponibili` TO 'operatore';

GRANT EXECUTE ON procedure `mydb`.`visualizza_contatti_operatore` TO 'operatore';

GRANT EXECUTE ON procedure `mydb`.`elimina_nota` TO 'operatore';

GRANT EXECUTE ON procedure `mydb`.`visualizza_contatti_cliente` TO 'operatore';

GRANT EXECUTE ON procedure `mydb`.`visualizza_sala_sede` TO 'operatore';

SET SQL_MODE = '';

DROP USER IF EXISTS manager;

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

```
CREATE USER 'manager' IDENTIFIED BY 'manager';


GRANT EXECUTE ON procedure `mydb`.`inserisci_proposta` TO 'manager';

GRANT EXECUTE ON procedure `mydb`.`modifica_disponibilita_proposta` TO 'manager';

GRANT EXECUTE ON procedure `mydb`.`visualizza_proposte` TO 'manager';

GRANT EXECUTE ON procedure `mydb`.`inserisci_utente` TO 'manager';

GRANT EXECUTE ON procedure `mydb`.`visualizza_proposta` TO 'manager';

GRANT EXECUTE ON procedure `mydb`.`visualizza_utenti` TO 'manager';

GRANT EXECUTE ON procedure `mydb`.`conta_proposte_acquistate` TO 'manager';

SET SQL_MODE = '';

DROP USER IF EXISTS f_commerciale;

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

CREATE USER 'f_commerciale' IDENTIFIED BY 'commerciale';


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

GRANT EXECUTE ON procedure `mydb`.`visualizza_clienti` TO 'f_commerciale';

SET SQL_MODE = '';

DROP USER IF EXISTS login;

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

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

```
GRANT EXECUTE ON procedure `mydb`.`login` TO 'login';


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


-- ------------------------------------------------------
-- Data for table `mydb`.`utenti`
-- ------------------------------------------------------
START TRANSACTION;
USE `mydb`;
INSERT INTO `mydb`.`utenti` (`username`, `password`, `nome_utente`, `cognome_utente`, `data_nascita_utente`, `email_utente`, `ruolo`) VALUES ('ValentinaMartini99', 'ab3ab964804dc9ae20de3b02d379b1bd', 'Valentina', 'Martini', '1999-05-08', 'valentina.martini@gmail.com', 'operatore');
INSERT INTO `mydb`.`utenti` (`username`, `password`, `nome_utente`, `cognome_utente`, `data_nascita_utente`, `email_utente`, `ruolo`) VALUES ('ValerioGanci95', '7d9644c560a8e85d9e53265ef4739927', 'Valerio', 'Ganci', '1995-04-26', 'valerio.ganci@gmail.com', 'manager');
INSERT INTO `mydb`.`utenti` (`username`, `password`, `nome_utente`, `cognome_utente`, `data_nascita_utente`, `email_utente`, `ruolo`) VALUES ('MatteoMartini93', '150be5b860e60a7fc7c7d9b9815e93d1', 'Matteo', 'Martini', '1993-11-12', 'matteo.martini@gmail.com', 'operatore');
INSERT INTO `mydb`.`utenti` (`username`, `password`, `nome_utente`, `cognome_utente`, `data_nascita_utente`, `email_utente`, `ruolo`) VALUES ('AlessiaZamba94', '6332e88a4c7dba6f7743d3a7a0c6ea2c', 'Alessia', 'Zamba', '1994-07-01', 'alessia.zamba@gmail.com', 'f_commerciale');
```

```
INSERT INTO `mydb`.`utenti` (`username`, `password`, `nome_utente`, `cognome_utente`, `data_nascita_utente`, `email_utente`, `ruolo`) VALUES ('ChiaraNatalizi98', '243a3b6f7ddfea2599743ce3370d5229', 'Chiara', 'Natalizi', '1998-08-17', 'chiara.natalizi@gmail.com', 'f_commerciale');

INSERT INTO `mydb`.`utenti` (`username`, `password`, `nome_utente`, `cognome_utente`, `data_nascita_utente`, `email_utente`, `ruolo`) VALUES ('MonicaNatalizi68', 'ff0d813dd5d2f64dd372c6c4b6aed086', 'Monica', 'Natalizi', '1968-09-13', 'monica.natalizi@gmail.com', 'manager');


COMMIT;



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

-- Data for table `mydb`.`clienti`

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

START TRANSACTION;

USE `mydb`;

INSERT INTO `mydb`.`clienti` (`CF`, `nome`, `cognome`, `data_nascita`, `data_registrazione`, `email`, `fax`, `piva`, `nome_società`, `operatore`) VALUES ('MLFCNT99C48H501J', 'Malefica', 'Maleficent', '1999-03-18', '2006-07-16', 'malefica@gmail.com', '12345', NULL, NULL, 'ValentinaMartini99');

INSERT INTO `mydb`.`clienti` (`CF`, `nome`, `cognome`, `data_nascita`, `data_registrazione`, `email`, `fax`, `piva`, `nome_società`, `operatore`) VALUES ('SCRLTW93W39H501T', 'Scarlet', 'Witch', '1993-09-10', '2010-10-29', 'wanda.maximof@gmail.com', '98765', '99836782966', 'Wanda Vision', 'MatteoMartini93');

INSERT INTO `mydb`.`clienti` (`CF`, `nome`, `cognome`, `data_nascita`, `data_registrazione`, `email`, `fax`, `piva`, `nome_società`, `operatore`) VALUES ('HRMGRN99E48H501J', 'Hermione', 'Granger', '1999-10-11', '2005-06-04', 'hermione.granger@gmail.com', 'abcd', NULL, NULL, 'ValentinaMartini99');
```

INSERT INTO `mydb`.`clienti` (`CF`, `nome`, `cognome`, `data_nascita`, `data_registrazione`, `email`, `fax`, `piva`, `nome_società`, `operatore`) VALUES ('PRNJSM90E48H501K', 'Jasmine', 'Princess', '1990-01-20', '2012-04-10', 'jasmine.princess@gmail.com', 'faxfax', '88760448729', 'Disney Family', 'ValentinaMartini99');

INSERT INTO `mydb`.`clienti` (`CF`, `nome`, `cognome`, `data_nascita`, `data_registrazione`, `email`, `fax`, `piva`, `nome_società`, `operatore`) VALUES ('PTTHRR91C40H501P', 'Harry', 'Potter', '1991-03-11', '2015-01-14', 'harry.potter@gmail.com', 'harryfax', NULL, NULL, 'MatteoMartini93');

INSERT INTO `mydb`.`clienti` (`CF`, `nome`, `cognome`, `data_nascita`, `data_registrazione`, `email`, `fax`, `piva`, `nome_società`, `operatore`) VALUES ('WSLRNL99R40H601A', 'Ronald', 'Weasly', '1999-06-25', '2012-08-08', 'ronald.weasly@gmail.com', 'ronaldfax', '11098668537', 'Weasly Community', 'ValentinaMartini99');

INSERT INTO `mydb`.`clienti` (`CF`, `nome`, `cognome`, `data_nascita`, `data_registrazione`, `email`, `fax`, `piva`, `nome_società`, `operatore`) VALUES ('BLCSRS80R30H501J', 'Sirius', 'Black', '1980-02-16', '2000-03-19', 'sirius.black@gmail.com', 'siriusfax', '99633587288', 'I Malandrini', 'ValentinaMartini99');

INSERT INTO `mydb`.`clienti` (`CF`, `nome`, `cognome`, `data_nascita`, `data_registrazione`, `email`, `fax`, `piva`, `nome_società`, `operatore`) VALUES ('PCKNVL90F48H501K', 'Neville', 'Paciok', '1990-10-11', '2003-10-18', 'neville.paciok@gmail.com', 'nevillefax', NULL, NULL, 'MatteoMartini93');

COMMIT;

-- ---

-- Data for table `mydb`.`contatti`

-- ---

START TRANSACTION;

USE `mydb`;

```
INSERT INTO `mydb`.`contatti` (`IDcontatto`, `nota`, `cliente`, `operatore`, `data_contatto`)
VALUES (1, 'nota1', 'HRMGRN99E48H501J', 'ValentinaMartini99', '2022-01-10');

INSERT INTO `mydb`.`contatti` (`IDcontatto`, `nota`, `cliente`, `operatore`, `data_contatto`)
VALUES (2, 'nota2', 'SCRLTW93W39H501T', 'MatteoMartini93', '2022-03-22');

INSERT INTO `mydb`.`contatti` (`IDcontatto`, `nota`, `cliente`, `operatore`, `data_contatto`)
VALUES (3, 'nota3', 'PRNJSM90E48H501K', 'ValentinaMartini99', '2022-04-19');

INSERT INTO `mydb`.`contatti` (`IDcontatto`, `nota`, `cliente`, `operatore`, `data_contatto`)
VALUES (4, 'nota4', 'BLCSRS80R30H501J', 'ValentinaMartini99', '2022-02-05');

INSERT INTO `mydb`.`contatti` (`IDcontatto`, `nota`, `cliente`, `operatore`, `data_contatto`)
VALUES (5, 'nota5', 'PTTHRR91C40H501P', 'MatteoMartini93', '2022-05-28');

INSERT INTO `mydb`.`contatti` (`IDcontatto`, `nota`, `cliente`, `operatore`, `data_contatto`)
VALUES (6, 'nota6', 'WSLRNL99R40H601A', 'ValentinaMartini99', '2022-06-15');

INSERT INTO `mydb`.`contatti` (`IDcontatto`, `nota`, `cliente`, `operatore`, `data_contatto`)
VALUES (7, 'nota7', 'MLFCNT99C48H501J', 'ValentinaMartini99', '2022-06-28');


COMMIT;



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

-- Data for table `mydb`.`proposte`

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

START TRANSACTION;

USE `mydb`;

INSERT INTO `mydb`.`proposte` (`codice`, `disponibilità`, `descrizione`, `manager`) VALUES
('COD1', 1, 'descrizione COD1', 'ValerioGanci95');

INSERT INTO `mydb`.`proposte` (`codice`, `disponibilità`, `descrizione`, `manager`) VALUES
('COD2', 1, 'descrizione COD2', 'MonicaNatalizi68');
```

```
INSERT INTO `mydb`.`proposte` (`codice`, `disponibilità`, `descrizione`, `manager`) VALUES ('COD3', 1, 'descrizione COD3', 'ValerioGanci95');

INSERT INTO `mydb`.`proposte` (`codice`, `disponibilità`, `descrizione`, `manager`) VALUES ('COD4', 0, 'descrizione COD4', 'MonicaNatalizi68');


COMMIT;



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

-- Data for table `mydb`.`proposte_accettate`

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

START TRANSACTION;

USE `mydb`;

INSERT INTO `mydb`.`proposte_accettate` (`contatto`, `proposta`) VALUES (1, 'COD1');

INSERT INTO `mydb`.`proposte_accettate` (`contatto`, `proposta`) VALUES (1, 'COD3');

INSERT INTO `mydb`.`proposte_accettate` (`contatto`, `proposta`) VALUES (2, 'COD1');

INSERT INTO `mydb`.`proposte_accettate` (`contatto`, `proposta`) VALUES (2, 'COD2');

INSERT INTO `mydb`.`proposte_accettate` (`contatto`, `proposta`) VALUES (2, 'COD3');

INSERT INTO `mydb`.`proposte_accettate` (`contatto`, `proposta`) VALUES (3, 'COD2');

INSERT INTO `mydb`.`proposte_accettate` (`contatto`, `proposta`) VALUES (4, 'COD2');

INSERT INTO `mydb`.`proposte_accettate` (`contatto`, `proposta`) VALUES (4, 'COD3');

INSERT INTO `mydb`.`proposte_accettate` (`contatto`, `proposta`) VALUES (5, 'COD1');

INSERT INTO `mydb`.`proposte_accettate` (`contatto`, `proposta`) VALUES (5, 'COD2');

INSERT INTO `mydb`.`proposte_accettate` (`contatto`, `proposta`) VALUES (6, 'COD1');

INSERT INTO `mydb`.`proposte_accettate` (`contatto`, `proposta`) VALUES (6, 'COD3');
```

```
INSERT INTO `mydb`.`proposte_accettate` (`contatto`, `proposta`) VALUES (7, 'COD1');


COMMIT;



-- -----------------------------------------------------
-- Data for table `mydb`.`sedi`
-- -----------------------------------------------------
START TRANSACTION;
USE `mydb`;
INSERT INTO `mydb`.`sedi` (`via`, `città`) VALUES ('via prova1, 1', 'Roma');
INSERT INTO `mydb`.`sedi` (`via`, `città`) VALUES ('via prova2, 2', 'Roma');
INSERT INTO `mydb`.`sedi` (`via`, `città`) VALUES ('via prova3, 3', 'Milano');
INSERT INTO `mydb`.`sedi` (`via`, `città`) VALUES ('via prova4, 4', 'Napoli');

COMMIT;



-- -----------------------------------------------------
-- Data for table `mydb`.`sale`
-- -----------------------------------------------------
START TRANSACTION;
USE `mydb`;
INSERT INTO `mydb`.`sale` (`numero`, `via_sede`, `città_sede`) VALUES ('1', 'via prova1, 1',
'Roma');
```

```
INSERT INTO `mydb`.`sale` (`numero`, `via_sede`, `città_sede`) VALUES ('2', 'via prova1, 1', 'Roma');

INSERT INTO `mydb`.`sale` (`numero`, `via_sede`, `città_sede`) VALUES ('3', 'via prova1, 1', 'Roma');

INSERT INTO `mydb`.`sale` (`numero`, `via_sede`, `città_sede`) VALUES ('1', 'via prova3, 3', 'Milano');

INSERT INTO `mydb`.`sale` (`numero`, `via_sede`, `città_sede`) VALUES ('2', 'via prova3, 3', 'Milano');

INSERT INTO `mydb`.`sale` (`numero`, `via_sede`, `città_sede`) VALUES ('3', 'via prova3, 3', 'Milano');

INSERT INTO `mydb`.`sale` (`numero`, `via_sede`, `città_sede`) VALUES ('4', 'via prova3, 3', 'Milano');

INSERT INTO `mydb`.`sale` (`numero`, `via_sede`, `città_sede`) VALUES ('1', 'via prova4, 4', 'Napoli');

INSERT INTO `mydb`.`sale` (`numero`, `via_sede`, `città_sede`) VALUES ('2', 'via prova4, 4', 'Napoli');


COMMIT;



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

-- Data for table `mydb`.`appuntamenti`

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

START TRANSACTION;

USE `mydb`;
```

INSERT INTO `mydb`.`appuntamenti` (`data_ora_inizio`, `sala`, `via_sala_sede`, `città_sala_sede`, `resoconto`, `operatore`, `cliente`, `data_ora_fine`) VALUES ('2022-10-20 16:00:00', '2', 'via prova1, 1', 'Roma', NULL, 'ValentinaMartini99', 'WSLRNL99R40H601A', '2022-10-20 17:00:00');

INSERT INTO `mydb`.`appuntamenti` (`data_ora_inizio`, `sala`, `via_sala_sede`, `città_sala_sede`, `resoconto`, `operatore`, `cliente`, `data_ora_fine`) VALUES ('2022-01-25 17:15:00', '3', 'via prova1, 1', 'Roma', 'resoconto Hermione', 'ValentinaMartini99', 'HRMGRN99E48H501J', '2022-01-25 18:15:00');

INSERT INTO `mydb`.`appuntamenti` (`data_ora_inizio`, `sala`, `via_sala_sede`, `città_sala_sede`, `resoconto`, `operatore`, `cliente`, `data_ora_fine`) VALUES ('2022-09-12 09:00:00', '2', 'via prova4, 4', 'Napoli', NULL, 'MatteoMartini93', 'SCRLTW93W39H501T', '2022-09-12 10:00:00');

INSERT INTO `mydb`.`appuntamenti` (`data_ora_inizio`, `sala`, `via_sala_sede`, `città_sala_sede`, `resoconto`, `operatore`, `cliente`, `data_ora_fine`) VALUES ('2022-03-05 10:00:00', '1', 'via prova1, 1', 'Roma', 'resoconto Sirius', 'ValentinaMartini99', 'BLCSRS80R30H501J', '2022-03-05 11:00:00');

INSERT INTO `mydb`.`appuntamenti` (`data_ora_inizio`, `sala`, `via_sala_sede`, `città_sala_sede`, `resoconto`, `operatore`, `cliente`, `data_ora_fine`) VALUES ('2022-09-17 11:00:00', '1', 'via prova1, 1', 'Roma', NULL, 'ValentinaMartini99', 'PRNJSM90E48H501K', '2022-09-17 12:00:00');

INSERT INTO `mydb`.`appuntamenti` (`data_ora_inizio`, `sala`, `via_sala_sede`, `città_sala_sede`, `resoconto`, `operatore`, `cliente`, `data_ora_fine`) VALUES ('2022-06-15 14:30:00', '1', 'via prova4, 4', 'Napoli', 'resoconto Harry', 'MatteoMartini93', 'PTTHRR91C40H501P', '2022-06-15 15:30:00');

INSERT INTO `mydb`.`appuntamenti` (`data_ora_inizio`, `sala`, `via_sala_sede`, `città_sala_sede`, `resoconto`, `operatore`, `cliente`, `data_ora_fine`) VALUES ('2022-09-03 14:00:00', '1', 'via prova1, 1', 'Roma', NULL, 'ValentinaMartini99', 'MLFCNT99C48H501J', '2022-09-03 15:00:00');

COMMIT;

-- --

-- Data for table `mydb`.`proposte_acquistate`

-- --

```
START TRANSACTION;

USE `mydb`;

INSERT INTO `mydb`.`proposte_acquistate` (`data_appuntamento`, `sala_appuntamento`, `via_appuntamento`, `città_appuntamento`, `proposta`) VALUES ('2022-01-25 17:15:00', '3', 'via prova1, 1', 'Roma', 'COD1');

INSERT INTO `mydb`.`proposte_acquistate` (`data_appuntamento`, `sala_appuntamento`, `via_appuntamento`, `città_appuntamento`, `proposta`) VALUES ('2022-03-05 10:00:00', '1', 'via prova1, 1', 'Roma', 'COD2');

INSERT INTO `mydb`.`proposte_acquistate` (`data_appuntamento`, `sala_appuntamento`, `via_appuntamento`, `città_appuntamento`, `proposta`) VALUES ('2022-06-15 14:30:00', '1', 'via prova4, 4', 'Napoli', 'COD1');

INSERT INTO `mydb`.`proposte_acquistate` (`data_appuntamento`, `sala_appuntamento`, `via_appuntamento`, `città_appuntamento`, `proposta`) VALUES ('2022-01-25 17:15:00', '3', 'via prova1, 1', 'Roma', 'COD3');


COMMIT;



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

-- Data for table `mydb`.`cliente_acquista`

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

START TRANSACTION;

USE `mydb`;

INSERT INTO `mydb`.`cliente_acquista` (`cliente`, `proposta`) VALUES ('HRMGRN99E48H501J', 'COD1');

INSERT INTO `mydb`.`cliente_acquista` (`cliente`, `proposta`) VALUES ('HRMGRN99E48H501J', 'COD3');
```

INSERT INTO `mydb`.`cliente_acquista` (`cliente`, `proposta`) VALUES ('BLCSRS80R30H501J', 'COD2');

INSERT INTO `mydb`.`cliente_acquista` (`cliente`, `proposta`) VALUES ('PTTHRR91C40H501P', 'COD1');

COMMIT;

Codice del Front-End

- Main.c

```
#include "functions.h"
#include "sqlFunctions.h"
#include "utility.h"

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


MYSQL *conn;
bool cont = true;
bool logout = false;

void executeCommandOperatore(char* cmd, char *loggedUser){
   if(strcmp(cmd,"exit")==0){
      cont=false;
   }
   else if(strcmp(cmd,"logout")==0){
      cont = false;
      logout = true;
   }
   else if(strcmp(cmd,"visualizza elenco clienti") == 0){
      cmdVisualizzaClientiOperatore(loggedUser);
   }
   else if(strcmp(cmd,"visualizza proposte disponibili") == 0){
      cmdVisualizzaProposteDisponibili(conn);
   }
   else if(strcmp(cmd,"visualizza elenco contatti") == 0){
      cmdVisualizzaContattiOperatore(conn, loggedUser);
```

```
    }
    else if(strcmp(cmd,"visualizza elenco appuntamenti") == 0){
        cmdVisualizzaAppuntamentiOperatore(conn,loggedUser);
    }
    else if(strcmp(cmd,"inserisci contatto") == 0){
        cmdInserisciContatto(conn, loggedUser);
    }
    else if(strcmp(cmd,"inserisci appuntamento") == 0){
        cmdInserisciAppuntamento(conn, loggedUser);
    }
    else{
        printf("\nComando non riconosciuto!\n");
    }
}

void executeCommandManager(char* cmd, char *loggedUser){
    if(strcmp(cmd,"exit")==0){
        cont=false;
    }
    else if(strcmp(cmd,"logout")==0){
        cont = false;
        logout = true;
    }
    else if(strcmp(cmd,"visualizza elenco proposte") == 0){
        cmdVisualizzaProposte(conn);
    }
    else if(strcmp(cmd,"inserisci proposta") == 0){
        cmdInserisciProposta(conn, loggedUser);
    }
    else if(strcmp(cmd,"visualizza proposte acquistate") == 0){
        cmdContaProposteAcquistate(conn);
    }
    else if(strcmp(cmd,"visualizza elenco utenti") == 0){
        cmdVisualizzaUtenti(conn);
    }
    else{
        printf("\nComando non riconosciuto!\n");
    }
}

void executeCommandFcommerciale(char* cmd){
    if(strcmp(cmd,"exit")==0){
        cont=false;
    }
    else if(strcmp(cmd,"logout")==0){
        cont = false;
        logout = true;
    }
    else if(strcmp(cmd,"visualizza elenco clienti") == 0){
        cmdVisualizzaClienti(conn);
```

```
    }
    else{
        printf("\nComando non riconosciuto!\n");
    }
}


int main() {

    conn = connectToDb();

    if(conn==NULL){
        exit(EXIT_FAILURE);
    }

    login:
        cont = true;
        logout = false;

        char *username = takeCustomInput("Inserisci l'username:", 22);
        char *password = takeCustomInput("Inserisci la password:", 34);
        int role = cmdLogin(conn, username,password);

        switch(role){
            case 1:
                while(cont){
                    printf("\n\nI comandi disponibili sono: \n - visualizza elenco clienti \n - visualizza
elenco contatti \n - visualizza elenco appuntamenti \n - visualizza proposte disponibili \n - inserisci
contatto \n - inserisci appuntamento \n - logout \n - exit \n");
                    connect_as_operatore();
                    executeCommandOperatore(takeInput("Inserisci un comando:"), username);
                }
                break;

            case 2:
                while(cont){
                    printf("\n\nI comandi disponibili sono: \n - visualizza elenco proposte \n - inserisci
proposta \n - visualizza proposte acquistate \n - visualizza elenco utenti \n - logout \n - exit \n");
                    connect_as_manager();
                    executeCommandManager(takeInput("Inserisci un comando:"), username);
                }
                break;

            case 3:
                while(cont){
                    printf("\n\nI comandi disponibili sono: \n - visualizza elenco clienti \n - logout \n - exit
\n");
                    connect_as_commerciale();
                    executeCommandFcommerciale(takeInput("Inserisci un comando:"));
                }
```

```
            break;
        default:
            printf("\nLogin fallito, le credenziali sono errate\n");
            goto login;


    }

    if (logout == true){
        connect_as_login();
        goto login;
    }

  closeConnection();
  exit(EXIT_SUCCESS);
}
```

- sqlFunctions.c

```
#include "sqlFunctions.h"
#include "utility.h"

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

static char *opt_host_name = "localhost"; /* host (default=localhost) */
static char *opt_user_name = "login"; /* username (default=login name)*/
static char *opt_password = "login"; /* password (default=none) */
static unsigned int opt_port_num = 3306; /* port number (use built-in) */
static char *opt_socket_name = NULL; /* socket name (use built-in) */
static char *opt_db_name ="mydb";/* database name (default=none) */
static unsigned int opt_flags = 0; /* connection flags (none) */

static char *opt_user_name_operatore = "operatore";
static char *opt_password_operatore = "operatore";

static char *opt_user_name_commerciale= "f_commerciale";
static char *opt_password_commerciale = "commerciale";

static char *opt_user_name_manager = "manager";
static char *opt_password_manager = "manager";

static MYSQL *conn;

MYSQL* connectToDb(){
  conn=mysql_init(NULL);

  if(conn==NULL){
```

```
        fprintf(stderr,"mysql_init() fallita\n");
        return NULL;
    }

    if(mysql_real_connect(conn, opt_host_name, opt_user_name, opt_password, opt_db_name,
opt_port_num, opt_socket_name, opt_flags) == NULL) {
        mysql_close(conn);
        fprintf(stderr,"Connessione fallita!\n");
        return NULL;
    }

    printf("Connessione riuscita!\n");
    return conn;
}

void connect_as_operatore(){
    if ( mysql_change_user(conn, opt_user_name_operatore, opt_password_operatore, opt_db_name)
){
        fprintf(stderr, "mysql_change_user fallita");
        exit(EXIT_FAILURE);
    }
}

void connect_as_manager(){
    if ( mysql_change_user(conn, opt_user_name_manager, opt_password_manager, opt_db_name) ){
        fprintf(stderr, "mysql_change_user fallita");
        exit(EXIT_FAILURE);
    }
}

void connect_as_commerciale(){
    if ( mysql_change_user(conn, opt_user_name_commerciale, opt_password_commerciale,
opt_db_name) ){
        fprintf(stderr, "mysql_change_user fallita");
        exit(EXIT_FAILURE);
    }
}

void connect_as_login(){
    if ( mysql_change_user(conn, opt_user_name, opt_password, opt_db_name) ){
        fprintf(stderr, "mysql_change_user fallita");
        exit(EXIT_FAILURE);
    }
}

void closeConnection(){
    mysql_close(conn);
}
```

```
bool init_stmt(MYSQL_STMT **stmt, char* procedure){

   bool update_length = true;

   *stmt = mysql_stmt_init(conn);

   if(*stmt == NULL){
      fprintf(stderr,"Errore nella inizializzazione dello statement\n");
      return false;
   }

   if(mysql_stmt_prepare(*stmt, procedure,strlen(procedure))!=0){
      stmtError(*stmt,"Errore nella prepare statement\n", true);
      return false;
   }

   mysql_stmt_attr_set(*stmt, STMT_ATTR_UPDATE_MAX_LENGTH, &update_length);
   return true;

}
```

- Functions.c

```
#include "functions.h"
#include "sqlFunctions.h"
#include "structures.h"
#include "utility.h"

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

#define size_cf 18
#define size_nome 27
#define size_cognome 27
#define size_email 47
#define size_fax 47
#define size_partitaiva 13
#define size_società 47
#define size_proposta 12
#define size_nota 302
#define size_resoconto 302
#define size_sala 5
#define size_via 47
#define size_città 22
#define size_ruolo 3
#define size_descrizione 102
#define size_username 22
```

```
#define size_password 34


int cmdLogin(MYSQL *conn, char *username, char *password){

   int role = 0;

   MYSQL_STMT *stmt;
   stmt = mysql_stmt_init(conn);

   if(stmt == NULL){
      fprintf(stderr,"Errore nella init dello statement\n");
      return -1;
   }

   char* stmtStr = "CALL login(?,?,?);";
   if(mysql_stmt_prepare(stmt, stmtStr,strlen(stmtStr))!=0){
      stmtError(stmt,"Errore nella prepare statement\n", true);
      return -1;
   }

   MYSQL_BIND ps_params[3];
   memset(ps_params, 0, sizeof (ps_params));

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

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

   ps_params[2].buffer_type = MYSQL_TYPE_LONG;
   ps_params[2].buffer = &role;
   ps_params[2].buffer_length = sizeof(role);

   if( mysql_stmt_bind_param(stmt, ps_params) != 0 ){
      stmtError(stmt, "Errore nel bind dei parametri per Login", true);
      return 4;
   }
   if (mysql_stmt_execute(stmt)!= 0){
      stmtError(stmt, "Non è possibile  effettuare il login", true);
      return 4;
   }

   memset(ps_params, 0, sizeof (ps_params));

   ps_params[0].buffer_type = MYSQL_TYPE_LONG;
   ps_params[0].buffer = &role;
   ps_params[0].buffer_length = sizeof(role);
```

```
    if ( mysql_stmt_bind_result(stmt, ps_params) ) {
        stmtError(stmt, "Non è possibile fare il bind dei parametri di output", true);
        return 4;
    }
    if ( mysql_stmt_fetch(stmt) ){
        stmtError(stmt, "Non è possibile recuperare i parametri di output", true);
        return 4;
    }

    mysql_stmt_close(stmt);
    return role;
}


void cmdVisualizzaClientiOperatore(char *loggedUser){
    bool cont = true;

    char* stmtStr = "CALL visualizza_clienti_operatore(?);";

    visualizza_clienti:
    MYSQL_STMT *stmt;
    if(!init_stmt(&stmt,stmtStr)){
        return;
    }

    MYSQL_BIND ps_params[1];
    memset(ps_params, 0, sizeof (ps_params));
    ps_params[0].buffer_type = MYSQL_TYPE_STRING;
    ps_params[0].buffer = loggedUser;
    ps_params[0].buffer_length = strlen(loggedUser);

    if ( mysql_stmt_bind_param(stmt, ps_params) != 0 ){
        stmtError(stmt, "Impossibile fare il bind dei parametri per Visualizza Clienti Operatore", true);
        return;
    }
    if ( mysql_stmt_execute(stmt) != 0){
        stmtError(stmt, "Errore nella visualizzazione dell'elenco clienti", true);
        return;
    }
    if (mysql_stmt_store_result(stmt)) {
        stmtError(stmt, "Impossibile salvare lo statement", true);
        return;
    }

    struct clienti *cli;
    int max = parse_clienti(stmt, &cli);
    mysql_stmt_data_seek(stmt,0);

    printResult(stmt);
```

```
    mysql_stmt_close(stmt);

    if(max == 0){
        printf("\n\nL'operatore non gestisce ancora alcun cliente\n\n");
        return;
    }

    while(cont){
        printf("\n\nI comandi disponibili: \n -visualizza cliente \n - back \n");
        char *cmd = takeInput("Inserisci il comando da eseguire:");
        if(strcmp(cmd,"back")==0){
            cont=false;
        }
        else if(strcmp(cmd,"visualizza cliente") == 0){
            cmdVisualizzaCliente(cli,max);
            free(cli);
            goto visualizza_clienti;
        }
        else{
            printf("\nComando non riconosciuto!\n");
        }
    }
    free(cli);
}

void cmdVisualizzaCliente(struct clienti *cli, int max){
    bool cont = true;
    char cf[17];

    inserire_numero:
    int number = atoi(takeInput("Inserisci numero cliente:"));

    if(number <= 0 || number > max){
        printf("Il numero inserito non corrisponde a nessun cliente\n");
        goto inserire_numero;
    }

    strcpy(cf,cli[number-1].cf);

    char* stmtStr = "CALL visualizza_cliente(?);";

    MYSQL_STMT *stmt;
    if(!init_stmt(&stmt,stmtStr)){
        return;
    }

    MYSQL_BIND ps_params[1];
    memset(ps_params, 0, sizeof (ps_params));
    ps_params[0].buffer_type = MYSQL_TYPE_STRING;
    ps_params[0].buffer = cf;
```

```
    ps_params[0].buffer_length = strlen(cf);

    if ( mysql_stmt_bind_param(stmt, ps_params) != 0 ){
        stmtError(stmt, "Impossibile fare il bind dei parametri per Visualizza Cliente", true);
        return;
    }
    if ( mysql_stmt_execute(stmt) != 0){
        stmtError(stmt, "Errore nella visualizzazione del cliente", true);
        return;
    }
    if (mysql_stmt_store_result(stmt)) {
        stmtError(stmt, "Impossibile salvare lo statement", true);
        return;
    }

    printResult(stmt);
    mysql_stmt_close(stmt);

    while(cont){
        printf("\n\nI comandi disponibili: \n - visualizza proposte accettate \n - visualizza proposte
acquistate \n - visualizza contatti \n - back \n");
        char *cmd = takeInput("Inserisci il comando da eseguire:");

        if(strcmp(cmd,"back")==0){
            cont=false;
        }
        else if(strcmp(cmd,"visualizza proposte accettate") == 0){
            cmdVisualizzaProposteAccettate(cf);
        }
        else if(strcmp(cmd,"visualizza proposte acquistate") == 0){
            cmdVisualizzaProposteAcquistate(cf);
        }
        else if(strcmp(cmd,"visualizza contatti") == 0){
            cmdVisualizzaContattiCliente(cf);
        }
        else{
            printf("\nComando non riconosciuto!\n");
        }
    }
}

void cmdVisualizzaProposteAccettate(char *cf){

    char* stmtStr = "CALL visualizza_proposte_accettate(?);";
    MYSQL_STMT *stmt;
    if(!init_stmt(&stmt,stmtStr)){
        return;
    }

    MYSQL_BIND ps_params[1];
```

```
    memset(ps_params, 0, sizeof (ps_params));
    ps_params[0].buffer_type = MYSQL_TYPE_STRING;
    ps_params[0].buffer = cf;
    ps_params[0].buffer_length = strlen(cf);

    if ( mysql_stmt_bind_param(stmt, ps_params) != 0 ){
        stmtError(stmt, "Impossibile fare il bind dei parametri per Visualizza Proposte Accettate", true);
        return;
    }
    if ( mysql_stmt_execute(stmt) != 0){
        stmtError(stmt, "Errore nella visualizzazione delle proposte accettate", true);
        return;
    }
    if (mysql_stmt_store_result(stmt)) {
        stmtError(stmt, "Impossibile salvare lo statement", true);
        return;
    }
    if (mysql_stmt_num_rows(stmt)==0){
        printf("\n\nIl cliente non ha ancora accettato delle proposte");
    }

    printResult(stmt);
    mysql_stmt_close(stmt);

}

void cmdVisualizzaProposteAcquistate(char *cf){

    char* stmtStr = "CALL visualizza_proposte_acquistate(?);";

    MYSQL_STMT *stmt;
    if(!init_stmt(&stmt,stmtStr)){
        return;
    }

    MYSQL_BIND ps_params[1];
    memset(ps_params, 0, sizeof (ps_params));
    ps_params[0].buffer_type = MYSQL_TYPE_STRING;
    ps_params[0].buffer = cf;
    ps_params[0].buffer_length = strlen(cf);

    if ( mysql_stmt_bind_param(stmt, ps_params) != 0 ){
        stmtError(stmt, "Impossibile fare il bind dei parametri per Visualizza Proposte Acquistate",
true);
        return;
    }
    if ( mysql_stmt_execute(stmt) != 0){
        stmtError(stmt, "Errore nella visualizzazione delle proposte acquistate", true);
        return;
    }
```

```
    if (mysql_stmt_store_result(stmt)) {
        stmtError(stmt, "Impossibile salvare lo statement", true);
        return;
    }
    if (mysql_stmt_num_rows(stmt)==0){
        printf("\n\nIl cliente non ha ancora acquistato delle proposte");
    }

    printResult(stmt);
    mysql_stmt_close(stmt);
}

void cmdVisualizzaContattiCliente(char *cf){
    char* stmtStr = "CALL visualizza_contatti_cliente(?);";
    MYSQL_STMT *stmt;
    if(!init_stmt(&stmt,stmtStr)){
        return;
    }

    MYSQL_BIND ps_params[1];
    memset(ps_params, 0, sizeof (ps_params));
    ps_params[0].buffer_type = MYSQL_TYPE_STRING;
    ps_params[0].buffer = cf;
    ps_params[0].buffer_length = strlen(cf);

    if ( mysql_stmt_bind_param(stmt, ps_params) != 0 ){
        stmtError(stmt, "Impossibile fare il bind dei parametri per Visualizza Contatti Cliente", true);
        return;
    }
    if ( mysql_stmt_execute(stmt) != 0){
        stmtError(stmt, "Errore nella visualizzazione dei contatti con tale cliente", true);
        return;
    }
    if (mysql_stmt_store_result(stmt)) {
        stmtError(stmt, "Impossibile salvare lo statement", true);
        return;
    }
    if (mysql_stmt_num_rows(stmt)==0){
        printf("\n\nNon ci sono ancora contatti con questo cliente");
    }

    printResult(stmt);
    mysql_stmt_close(stmt);
}



void cmdVisualizzaContattiOperatore(MYSQL *conn, char *loggedUser){
    bool cont = true;
```

```
char* stmtStr = "CALL visualizza_contatti_operatore(?);";

visualizza_contatti:

MYSQL_STMT *stmt;
if(!init_stmt(&stmt,stmtStr)){
    return;
}

MYSQL_BIND ps_params[1];
memset(ps_params, 0, sizeof (ps_params));
ps_params[0].buffer_type = MYSQL_TYPE_STRING;
ps_params[0].buffer = loggedUser;
ps_params[0].buffer_length = strlen(loggedUser);

if( mysql_stmt_bind_param(stmt, ps_params) != 0 ){
    stmtError(stmt, "Errore nel binding dei parametri per Visualizza Elenco Contatti",true);
    return;
}
if (mysql_stmt_execute(stmt)!= 0){
    stmtError(stmt, "Non è possibile visualizzare l'elenco dei contatto", true);
    return;
}
if (mysql_stmt_store_result(stmt)) {
    stmtError(stmt, "Impossibile salvare lo statement", true);
    return;
}

struct contatti *contatto;
int max = parse_contatti(stmt, &contatto);
mysql_stmt_data_seek(stmt,0);

printResult(stmt);
mysql_stmt_close(stmt);

if(max == 0){
    printf("\n\nNon ci sono ancora contatti per questo operatore");
    return;
}

while(cont){
    printf("\n\nI comandi disponibili sono: \n - visualizza contatto \n - back \n");
    char *cmd = takeInput("Inserisci il comando da eseguire");

    if(strcmp(cmd,"back")==0){
        cont=false;
    }
    else if(strcmp(cmd,"visualizza contatto") == 0){
        cmdVisualizzaContatto(conn, contatto, max);
        free(contatto);
```

```
            goto visualizza_contatti;
        }
        else{
            printf("\nComando non riconosciuto!\n");
        }
    }
    free(contatto);
}

void cmdVisualizzaContatto(MYSQL *conn,struct contatti *contatto, int max){
    bool cont = true;
    int IDcontatto;

    inserire_numero:
    int number = atoi(takeInput("Inserisci numero contatto:"));

    if(number <= 0 || number > max){
        printf("Il numero inserito non corrisponde a nessun contatto\n");
        goto inserire_numero;
    }

    IDcontatto = contatto[number-1].IDcontatto;

    char* stmtStr = "CALL visualizza_contatto(?);";
    MYSQL_STMT *stmt;
    if(!init_stmt(&stmt,stmtStr)){
        return;
    }

    MYSQL_BIND ps_params[1];
    memset(ps_params, 0, sizeof (ps_params));

    ps_params[0].buffer_type = MYSQL_TYPE_LONG;
    ps_params[0].buffer = &IDcontatto;
    ps_params[0].buffer_length = sizeof(IDcontatto);

    if( mysql_stmt_bind_param(stmt, ps_params) != 0 ){
        stmtError(stmt, "Errore nel binding dei parametri per Visualizza Contatto",true);
        return;
    }
    if (mysql_stmt_execute(stmt)!= 0){
        stmtError(stmt, "Non è possibile visualizza il contatto", true);
        return;
    }
    if (mysql_stmt_store_result(stmt)) {
        stmtError(stmt, "Impossibile salvare lo statement", true);
        return;
    }

    printResult(stmt);
```

```
    mysql_stmt_close(stmt);

    while(cont){
        printf("\n\nI comandi disponibili sono: \n - modifica nota \n - elimina nota \n - inserisci proposta
accettata \n - back \n");
        char *cmd = takeInput("Inserisci il comando da eseguire");

        if(strcmp(cmd,"back")==0){
            cont=false;
        }
        else if(strcmp(cmd,"modifica nota") == 0){
            cmdModificaNotaContatto(conn, IDcontatto);
        }
        else if(strcmp(cmd,"elimina nota") == 0){
            cmdEliminaNotaContatto(conn, IDcontatto);
        }
        else if(strcmp(cmd,"inserisci proposta accettata") == 0){
            cmdInserisciPropostaAccettata(conn,IDcontatto);
        }
        else{
            printf("\nComando non riconosciuto!\n");
        }
    }
}

void cmdModificaNotaContatto(MYSQL *conn, int IDcontatto){
    char* nota = takeCustomInput("Inserisci la nuova nota da inserire (massimo 300
caratteri):",size_nota);

    MYSQL_STMT *stmt;
    stmt = mysql_stmt_init(conn);
    if(stmt == NULL){
        fprintf(stderr,"Errore nella init dello statement\n");
        return;
    }
    char* stmtStr = "CALL modifica_nota_contatto(?,?);";
    if(mysql_stmt_prepare(stmt, stmtStr,strlen(stmtStr))!=0){
        stmtError(stmt,"Errore nella prepare statement\n", true);
        return;
    }

    MYSQL_BIND ps_params[2];
    memset(ps_params, 0, sizeof (ps_params));

    ps_params[0].buffer_type = MYSQL_TYPE_LONG;
    ps_params[0].buffer = &IDcontatto;
    ps_params[0].buffer_length = sizeof(IDcontatto);

    ps_params[1].buffer_type = MYSQL_TYPE_VAR_STRING;
    ps_params[1].buffer = nota;
```

```
    ps_params[1].buffer_length = strlen(nota);

    if( mysql_stmt_bind_param(stmt, ps_params) != 0 ){
        stmtError(stmt, "Errore nel binding dei parametri per Modifica Nota",true);
        return;
    }
    if (mysql_stmt_execute(stmt)!= 0){
        stmtPrintError(stmt, "Errore nella modifica della nota");
    }
    else{
        printf("\n\nLa nota è stata modifica con successo!\n");
    }
    mysql_stmt_close(stmt);
}

void cmdEliminaNotaContatto(MYSQL *conn, int IDcontatto){

    MYSQL_STMT *stmt;
    stmt = mysql_stmt_init(conn);
    if(stmt == NULL){
        fprintf(stderr,"Errore nella init dello statement\n");
        return;
    }
    char* stmtStr = "CALL elimina_nota(?);";
    if(mysql_stmt_prepare(stmt, stmtStr,strlen(stmtStr))!=0){
        stmtError(stmt,"Errore nella prepare statement\n", true);
        return;
    }

    MYSQL_BIND ps_params[1];
    memset(ps_params, 0, sizeof (ps_params));

    ps_params[0].buffer_type = MYSQL_TYPE_LONG;
    ps_params[0].buffer = &IDcontatto;
    ps_params[0].buffer_length = sizeof(IDcontatto);

    if( mysql_stmt_bind_param(stmt, ps_params) != 0 ){
        stmtError( stmt, "Errore nel binding dei parametri per Elimina Nota",true);
        return;
    }
    if (mysql_stmt_execute(stmt)!= 0){
        stmtPrintError(stmt, "Errore nell'eliminazione della nota");
    }
    else{
        printf("\n\nLa nota è stata eliminata con successo!\n");
    }
    mysql_stmt_close(stmt);
}

void cmdInserisciPropostaAccettata(MYSQL *conn, int IDcontatto){
```

```
    char *codice = takeCustomInput("Inserisci il codice della proposta accettata:", size_proposta);

    MYSQL_STMT *stmt;
    stmt = mysql_stmt_init(conn);
    if(stmt == NULL){
        fprintf(stderr,"Errore nella init dello statement\n");
        return;
    }
    char* stmtStr = "CALL inserisci_proposta_accettata(?,?);";
    if(mysql_stmt_prepare(stmt, stmtStr,strlen(stmtStr))!=0){
        stmtError(stmt,"Errore nella prepare statement\n", true);
        return;
    }

    MYSQL_BIND ps_params[2];
    memset(ps_params, 0, sizeof (ps_params));
    ps_params[0].buffer_type = MYSQL_TYPE_LONG;
    ps_params[0].buffer = &IDcontatto;
    ps_params[0].buffer_length = sizeof(IDcontatto);

    ps_params[1].buffer_type = MYSQL_TYPE_STRING;
    ps_params[1].buffer = codice;
    ps_params[1].buffer_length = strlen(codice);

    if ( mysql_stmt_bind_param(stmt, ps_params) != 0 ){
        stmtError(stmt, "Impossibile fare il bind dei parametri per Inserisci Proposta Accettata", true);
        return;
    }
    if (mysql_stmt_execute(stmt) != 0){
        stmtPrintError(stmt, "Non è stato possibile inserire l'accettazione della proposta");
        checkError(stmt,"La proposta inserita non esiste\n", "Questa proposta è stata già accettata\n");
    }
    else{
        printf("\n\nAccettazione della proposta inserita con successo!\n");
    }

    mysql_stmt_close(stmt);
}



void cmdVisualizzaAppuntamentiOperatore(MYSQL *conn, char *loggedUser){
    bool cont = true;

    char* stmtStr = "CALL visualizza_appuntamenti_operatore(?);";
    visualizza_appuntamenti:
    MYSQL_STMT *stmt;
     if(!init_stmt(&stmt,stmtStr)){
        return;
```

```
}

MYSQL_BIND ps_params[1];
memset(ps_params, 0, sizeof (ps_params));
ps_params[0].buffer_type = MYSQL_TYPE_STRING;
ps_params[0].buffer = loggedUser;
ps_params[0].buffer_length = strlen(loggedUser);

if( mysql_stmt_bind_param(stmt, ps_params) != 0 ){
   stmtError(stmt, "Errore nel binding dei parametri per Visualizza Elenco Appuntamenti",true);
   return;
}
if (mysql_stmt_execute(stmt)!= 0){
   stmtError(stmt, "Non è possibile visualizzare l'elenco degli appuntamenti", true);
   return;
}
if (mysql_stmt_store_result(stmt)) {
   stmtError(stmt, "Impossibile salvare lo statement", true);
   return;
}

struct appuntamenti *app;
int max = parse_appuntamenti(stmt, &app);
mysql_stmt_data_seek(stmt,0);

printResult(stmt);
mysql_stmt_close(stmt);

if(max == 0){
   printf("\n\nNon ci sono ancora appuntamenti per questo operatore");
   return;
}

while(cont){
   printf("\n\nI comandi disponibili sono: \n - visualizza appuntamento \n - back \n");
   char *cmd = takeInput("Inserisci il comando da eseguire");

   if(strcmp(cmd,"back")==0){
      cont=false;
   }
   else if(strcmp(cmd,"visualizza appuntamento") == 0){
      cmdVisualizzaAppuntamento(conn, app, max);
      free(app);
      goto visualizza_appuntamenti;
   }
   else{
      printf("\nComando non riconosciuto!\n");
   }
}
free(app);
```

```
}

void cmdVisualizzaAppuntamento(MYSQL *conn, struct appuntamenti *app, int max){
   bool cont = true;
   MYSQL_TIME data_inizio;
   char sala[4];
   char via[46];
   char città[21];
   char cliente[17];

   inserire_numero:

   int number = atoi(takeInput("Inserisci numero dell'appuntamento:"));

   if(number <= 0 || number > max){
      printf("Il numero inserito non corrisponde a nessun appuntamento\n");
      goto inserire_numero;
   }

   memcpy(&data_inizio,&app[number-1].inizio, sizeof(app[number-1].inizio));
   strcpy(sala,app[number-1].sala);
   strcpy(via,app[number-1].via);
   strcpy(città,app[number-1].città);
   strcpy(cliente,app[number-1].cliente);

   char* stmtStr = "CALL visualizza_appuntamento(?,?,?,?);";
   MYSQL_STMT *stmt;
    if(!init_stmt(&stmt,stmtStr)){
      return;
   }

   MYSQL_BIND ps_params[4];
   memset(ps_params, 0, sizeof (ps_params));
   ps_params[0].buffer_type = MYSQL_TYPE_DATETIME;
   ps_params[0].buffer = (char*)&data_inizio;
   ps_params[0].buffer_length = sizeof(data_inizio);

   ps_params[1].buffer_type = MYSQL_TYPE_VAR_STRING;
   ps_params[1].buffer = sala;
   ps_params[1].buffer_length = strlen(sala);

   ps_params[2].buffer_type = MYSQL_TYPE_VAR_STRING;
   ps_params[2].buffer = via;
   ps_params[2].buffer_length = strlen(via);

   ps_params[3].buffer_type = MYSQL_TYPE_VAR_STRING;
   ps_params[3].buffer = città;
   ps_params[3].buffer_length = strlen(città);

   if( mysql_stmt_bind_param(stmt, ps_params) != 0 ){
```

```
        stmtError(stmt, "Errore nel binding dei parametri per Visualizza Appuntamento",true);
        return;
    }
    if (mysql_stmt_execute(stmt)!= 0){
        stmtError(stmt, "Non è possibile visualizzare l'appuntamento", true);
        return;
    }
    if (mysql_stmt_store_result(stmt)) {
        stmtError(stmt, "Impossibile salvare lo statement", true);
        return;
    }

    printResult(stmt);
    mysql_stmt_close(stmt);

    while(cont){
        printf("\n\nI comandi disponibili sono: \n - modifica resoconto \n - modifica data/ora \n -
inserisci proposta acquistata \n - back \n");
        char *cmd = takeInput("Inserisci il comando da eseguire");

        if(strcmp(cmd,"back")==0){
            cont=false;
        }
        else if(strcmp(cmd,"modifica resoconto") == 0){
            cmdModificaResoconto(conn,data_inizio, sala, via, città);
        }
        else if(strcmp(cmd,"modifica data/ora") == 0){
            bool modifica = cmdModificaDataAppuntamento(conn,data_inizio, sala, via, città);
            if (modifica){
                cont = false;
            }
        }
        else if(strcmp(cmd,"inserisci proposta acquistata") == 0){
            cmdInserisciPropostaAcquistata(conn, data_inizio, sala, via, città, cliente);
        }
        else{
            printf("\nComando non riconosciuto!\n");
        }
    }
}

void cmdModificaResoconto(MYSQL *conn, MYSQL_TIME data_inizio, char *sala, char *via,
char *città){
    char* resoconto = takeCustomInput("Inserisci il nuovo resoconto (massimo 300 caratteri):",
size_resoconto);

    MYSQL_STMT *stmt;
    stmt = mysql_stmt_init(conn);
    if(stmt == NULL){
        fprintf(stderr,"Errore nella init dello statement\n");
```

```
        return;
    }
    char* stmtStr = "CALL modifica_resoconto_appuntamento(?,?,?,?,?);";
    if(mysql_stmt_prepare(stmt, stmtStr,strlen(stmtStr))!=0){
        stmtError(stmt,"Errore nella prepare statement\n", true);
        return;
    }

    MYSQL_BIND ps_params[5];
    memset(ps_params, 0, sizeof (ps_params));
    ps_params[0].buffer_type = MYSQL_TYPE_VAR_STRING;
    ps_params[0].buffer = resoconto;
    ps_params[0].buffer_length = strlen(resoconto);

    ps_params[1].buffer_type = MYSQL_TYPE_DATETIME;
    ps_params[1].buffer = (char*)&data_inizio;
    ps_params[1].buffer_length = sizeof(data_inizio);

    ps_params[2].buffer_type = MYSQL_TYPE_VAR_STRING;
    ps_params[2].buffer = sala;
    ps_params[2].buffer_length = strlen(sala);

    ps_params[3].buffer_type = MYSQL_TYPE_VAR_STRING;
    ps_params[3].buffer = via;
    ps_params[3].buffer_length = strlen(via);

    ps_params[4].buffer_type = MYSQL_TYPE_VAR_STRING;
    ps_params[4].buffer = città;
    ps_params[4].buffer_length = strlen(città);

    if( mysql_stmt_bind_param(stmt, ps_params) != 0 ){
        stmtError(stmt, "Errore nel binding dei parametri per Modifica Resoconto",true);
        return;
    }
    if (mysql_stmt_execute(stmt) != 0){
        stmtPrintError(stmt, "Non è stato possibile modificare il resoconto");
    }
    else{
        printf("\n\nResoconto modificato con successo!\n");
    }

    mysql_stmt_close(stmt);
}

bool cmdModificaDataAppuntamento(MYSQL *conn, MYSQL_TIME data_inizio, char *sala, char
*via, char *città){
    char *day;
    char *month;
    char *year;
    MYSQL_TIME nuova_data_inizio;
```

```
MYSQL_TIME nuova_data_fine;

memset(&nuova_data_inizio, 0, sizeof(nuova_data_inizio));
printf("Inserisci la nuova data:\n");
day = takeInput("Inserisci il giorno");
nuova_data_inizio.day = atoi(day);
month = takeInput("Inserisci il mese");
nuova_data_inizio.month = atoi(month);
year = takeInput("Inserisci l'anno"),
nuova_data_inizio.year = atoi(year);

printf("Nuovo orario di inizio dell'appuntamento:\n");
nuova_data_inizio.hour = atoi(takeInput("Inserisci l'ora (senza i minuti):"));
nuova_data_inizio.minute = atoi(takeInput("Inserisci i minuti:"));

memset(&nuova_data_fine, 0, sizeof(nuova_data_fine));
nuova_data_fine.day = atoi(day);
nuova_data_fine.month = atoi(month);
nuova_data_fine.year = atoi(year);

printf("Nuovo orario di fine dell'appuntamento (l'appuntamento può durare massimo un'ora):\n");
nuova_data_fine.hour = atoi(takeInput("Inserisci l'ora (senza i minuti):"));
nuova_data_fine.minute = atoi(takeInput("Inserisci i minuti:"));

MYSQL_STMT *stmt;
stmt = mysql_stmt_init(conn);
if(stmt == NULL){
   fprintf(stderr,"Errore nella init dello statement\n");
   return false;
}
char* stmtStr = "CALL modifica_dataora_appuntamento(?,?,?,?,?, ?);";
if(mysql_stmt_prepare(stmt, stmtStr,strlen(stmtStr))!=0){
   stmtError(stmt,"Errore nella prepare statement\n", true);
   return false;
}

MYSQL_BIND ps_params[6];
memset(ps_params, 0, sizeof (ps_params));

ps_params[0].buffer_type = MYSQL_TYPE_DATETIME;
ps_params[0].buffer = (char*)&data_inizio;
ps_params[0].buffer_length = sizeof(data_inizio);

ps_params[1].buffer_type = MYSQL_TYPE_VAR_STRING;
ps_params[1].buffer = sala;
ps_params[1].buffer_length = strlen(sala);

ps_params[2].buffer_type = MYSQL_TYPE_VAR_STRING;
ps_params[2].buffer = via;
ps_params[2].buffer_length = strlen(via);
```

```
    ps_params[3].buffer_type = MYSQL_TYPE_VAR_STRING;
    ps_params[3].buffer = città;
    ps_params[3].buffer_length = strlen(città);

    ps_params[4].buffer_type = MYSQL_TYPE_DATETIME;
    ps_params[4].buffer = (char*)&nuova_data_inizio;
    ps_params[4].buffer_length = sizeof(nuova_data_inizio);

    ps_params[5].buffer_type = MYSQL_TYPE_DATETIME;
    ps_params[5].buffer = (char*)&nuova_data_fine;
    ps_params[5].buffer_length = sizeof(nuova_data_fine);

    if( mysql_stmt_bind_param(stmt, ps_params) != 0 ){
        stmtError(stmt, "Errore nel binding dei parametri per Modifica Data/Ora",true);
        return false;
    }
    if (mysql_stmt_execute(stmt) != 0){
        stmtError(stmt, "Non è stato possibile modificare la data/ora", true);
        return false;
    }
    else{
        printf("\n\nData/Ora modificata con successo!\n");
    }

    mysql_stmt_close(stmt);
    return true;
}

void cmdInserisciPropostaAcquistata(MYSQL *conn, MYSQL_TIME data_inizio, char *sala, char
*via, char *città, char *cliente){

    char* codice_proposta = takeCustomInput("Inserire il codice della proposta che è stata
acquistata:", size_proposta);

    MYSQL_STMT *stmt;
    stmt = mysql_stmt_init(conn);
    if(stmt == NULL){
        fprintf(stderr,"Errore nella init dello statement\n");
        return;
    }
    char* stmtStr = "CALL inserisci_proposta_acquistata(?,?,?,?,?,?);";
    if(mysql_stmt_prepare(stmt, stmtStr,strlen(stmtStr))!=0){
        stmtError(stmt,"Errore nella prepare statement\n", true);
        return;
    }

    MYSQL_BIND ps_params[6];
    memset(ps_params, 0, sizeof (ps_params));
    ps_params[0].buffer_type = MYSQL_TYPE_DATETIME;
```

```
    ps_params[0].buffer = (char*)&data_inizio;
    ps_params[0].buffer_length = sizeof(data_inizio);

    ps_params[1].buffer_type = MYSQL_TYPE_VAR_STRING;
    ps_params[1].buffer = sala;
    ps_params[1].buffer_length = strlen(sala);

    ps_params[2].buffer_type = MYSQL_TYPE_VAR_STRING;
    ps_params[2].buffer = via;
    ps_params[2].buffer_length = strlen(via);

    ps_params[3].buffer_type = MYSQL_TYPE_VAR_STRING;
    ps_params[3].buffer = città;
    ps_params[3].buffer_length = strlen(città);

    ps_params[4].buffer_type = MYSQL_TYPE_VAR_STRING;
    ps_params[4].buffer = codice_proposta;
    ps_params[4].buffer_length = strlen(codice_proposta);

    ps_params[5].buffer_type = MYSQL_TYPE_STRING;
    ps_params[5].buffer = cliente;
    ps_params[5].buffer_length = strlen(cliente);

    if( mysql_stmt_bind_param(stmt, ps_params) != 0 ){
        stmtError(stmt, "Errore nel binding dei parametri per Inserisci Proposta Acquistata",true);
        return;
    }
    if (mysql_stmt_execute(stmt) != 0){
        stmtPrintError(stmt, "Non è stato possibile inserire l'acquisto della proposta");
        checkError(stmt,"La proposta inserita non esiste\n", "Questa proposta è stata già acquistata");
    }
    else{
        printf("\n\nAcquisto della proposta salvato con successo!\n");
    }

    mysql_stmt_close(stmt);
}



void cmdVisualizzaProposteDisponibili(){

    char* stmtStr = "CALL visualizza_proposte_disponibili();";

    MYSQL_STMT *stmt;
     if(!init_stmt(&stmt,stmtStr)){
        return;
    }

    if (mysql_stmt_execute(stmt)!= 0){
```

```
        stmtError(stmt, "Non è possibile visualizzare l'elenco delle proposte disponibili", true);
        return;
    }
    if (mysql_stmt_store_result(stmt)) {
        stmtError(stmt, "Impossibile salvare lo statement", true);
        return;
    }

    printResult(stmt);
    mysql_stmt_close(stmt);
}

void cmdInserisciContatto(MYSQL *conn, char *loggedUser){
    char *nota = takeCustomInput("Inserisci la nota (massimo 300 caratteri):", size_nota);
    char *cf = takeCustomInput("Inserisci il cliente:", size_cf);
    MYSQL_TIME data;

    memset(&data, 0, sizeof(data));
    data.day = atoi(takeInput("Inserisci il giorno"));
    data.month = atoi(takeInput("Inserisci il mese"));
    data.year = atoi(takeInput("Inserisci l'anno"));

    MYSQL_STMT *stmt;
    stmt = mysql_stmt_init(conn);
    if(stmt == NULL){
        fprintf(stderr,"Errore nella init dello statement\n");
        return;
    }
    char* stmtStr = "CALL inserisci_contatto(?,?,?,?);";
    if(mysql_stmt_prepare(stmt, stmtStr,strlen(stmtStr))!=0){
        stmtError(stmt,"Errore nella prepare statement\n", true);
        return;
    }

    MYSQL_BIND ps_params[4];
    memset(ps_params, 0, sizeof (ps_params));
    ps_params[0].buffer_type = MYSQL_TYPE_STRING;
    ps_params[0].buffer = nota;
    ps_params[0].buffer_length = strlen(nota);

    ps_params[1].buffer_type = MYSQL_TYPE_STRING;
    ps_params[1].buffer = cf;
    ps_params[1].buffer_length = strlen(cf);

    ps_params[2].buffer_type = MYSQL_TYPE_STRING;
    ps_params[2].buffer = loggedUser;
    ps_params[2].buffer_length = strlen(loggedUser);

    ps_params[3].buffer_type = MYSQL_TYPE_DATE;
    ps_params[3].buffer = (char*)&data;
```

```
    ps_params[3].buffer_length = sizeof(data);


    if( mysql_stmt_bind_param(stmt, ps_params) != 0 ){
        stmtError(stmt, "Errore nel binding dei parametri per Inserisci Contatto",true);
        return;
    }
    if (mysql_stmt_execute(stmt) != 0){
        stmtPrintError(stmt, "Non è stato possibile inserire il contatto");
        checkError(stmt, "Il cliente non esiste\n", "");
    }
    else{
        printf("\nContatto inserito con successo!\n");
    }

    mysql_stmt_close(stmt);
}

void cmdInserisciAppuntamento(MYSQL *conn,char *loggedUser){
    char *day;
    char *month;
    char *year;
    MYSQL_TIME data_inizio;
    MYSQL_TIME data_fine;

    memset(&data_inizio, 0, sizeof(data_inizio));
    printf("Data appuntamento:\n");
    day = takeInput("Inserisci il giorno");
    data_inizio.day = atoi(day);
    month = takeInput("Inserisci il mese");
    data_inizio.month = atoi(month);
    year = takeInput("Inserisci l'anno"),
    data_inizio.year = atoi(year);

    printf("Orario di inizio dell'appuntamento:\n");
    data_inizio.hour = atoi(takeInput("Inserisci l'ora (senza i minuti):"));
    data_inizio.minute = atoi(takeInput("Inserisci i minuti:"));

    memset(&data_fine, 0, sizeof(data_fine));
    data_fine.day = atoi(day);
    data_fine.month = atoi(month);
    data_fine.year = atoi(year);

    printf("Orario di fine dell'appuntamento (l'appuntamento può durare massimo un'ora):\n");
    data_fine.hour = atoi(takeInput("Inserisci l'ora (senza i minuti):"));
    data_fine.minute = atoi(takeInput("Inserisci i minuti:"));

    cmdVisualizzaSalaSede(conn);

    char *numero_sala = takeCustomInput("Inserisci il numero della sala:", size_sala);
```

```
char* via_sede = takeCustomInput("Inserisci la via della sede:", size_via);
char* città_sede = takeCustomInput("Inserisci la città della sede:", size_città);
char* cf = takeCustomInput("Inserisci il codice fiscale del cliente:", size_cf);


MYSQL_STMT *stmt;
stmt = mysql_stmt_init(conn);
if(stmt == NULL){
    fprintf(stderr,"Errore nella init dello statement\n");
    return;
}
char* stmtStr = "CALL inserisci_appuntamento(?,?,?,?,?,?,?);";
if(mysql_stmt_prepare(stmt, stmtStr,strlen(stmtStr))!=0){
    stmtError(stmt,"Errore nella prepare statement\n", true);
    return;
}

MYSQL_BIND ps_params[7];
memset(ps_params, 0, sizeof (ps_params));
ps_params[0].buffer_type = MYSQL_TYPE_DATETIME;
ps_params[0].buffer = (char*)&data_inizio;
ps_params[0].buffer_length = sizeof(data_inizio);

ps_params[1].buffer_type = MYSQL_TYPE_VAR_STRING;
ps_params[1].buffer = numero_sala;
ps_params[1].buffer_length = strlen(numero_sala);

ps_params[2].buffer_type = MYSQL_TYPE_VAR_STRING;
ps_params[2].buffer = via_sede;
ps_params[2].buffer_length = strlen(via_sede);

ps_params[3].buffer_type = MYSQL_TYPE_VAR_STRING;
ps_params[3].buffer = città_sede;
ps_params[3].buffer_length = strlen(città_sede);

ps_params[4].buffer_type = MYSQL_TYPE_VAR_STRING;
ps_params[4].buffer = loggedUser;
ps_params[4].buffer_length = strlen(loggedUser);

ps_params[5].buffer_type = MYSQL_TYPE_VAR_STRING;
ps_params[5].buffer = cf;
ps_params[5].buffer_length = strlen(cf);
ps_params[5].length = 0;
ps_params[5].is_null = 0;

ps_params[6].buffer_type = MYSQL_TYPE_DATETIME;
ps_params[6].buffer = (char*)&data_fine;
ps_params[6].buffer_length = sizeof(data_fine);

if( mysql_stmt_bind_param(stmt, ps_params) != 0 ){
```

```
        stmtError(stmt, "Errore nel binding dei parametri per Inserisci Appuntamento",true);
        return;
    }
    if (mysql_stmt_execute(stmt) != 0){
        stmtPrintError(stmt, "Non è stato possibile inserire l'appuntamento");
        checkError(stmt,"Controllare se la via e la città sono corretti, oppure se il codice fiscale del
cliente è corretto\n", "");
    }
    else{
        printf("\n\nAppuntamento inserito con successo!\n");
    }

    mysql_stmt_close(stmt);
}

void cmdVisualizzaSalaSede(){
    char* stmtStr = "CALL visualizza_sala_sede();";

    MYSQL_STMT *stmt;
    if(!init_stmt(&stmt,stmtStr)){
        return;
    }

    if (mysql_stmt_execute(stmt)!= 0){
        stmtError(stmt, "Non è possibile visualizzare l'elenco delle sale", true);
        return;
    }
    if (mysql_stmt_store_result(stmt)) {
        stmtError(stmt, "Impossibile salvare lo statement", true);
        return;
    }

    printResult(stmt);
    mysql_stmt_close(stmt);
}


void cmdVisualizzaClienti(MYSQL *conn){
    bool cont = true;

    char* stmtStr = "CALL visualizza_clienti();";

    visualizza_clienti:

    MYSQL_STMT *stmt;
    if(!init_stmt(&stmt,stmtStr)){
        return;
    }

    if (mysql_stmt_execute(stmt)!= 0){
```

```
        stmtError(stmt, "Non è possibile visualizzare l'elenco dei clienti", true);
        return;
    }
    if (mysql_stmt_store_result(stmt)) {
        stmtError(stmt, "Impossibile salvare lo statement", true);
        return;
    }

    printResult(stmt);
    mysql_stmt_close(stmt);

    while(cont){
        printf("\n\nI comandi disponibili sono: \n - inserisci cliente \n - back \n");
        char *cmd = takeInput("Inserisci il comando da eseguire");

        if(strcmp(cmd,"back")==0){
            cont=false;
        }
        else if(strcmp(cmd,"inserisci cliente") == 0){
            cmdInserisciCliente(conn);
            goto visualizza_clienti;
        }
        else{
            printf("\nComando non riconosciuto!\n");
        }
    }
}

void cmdInserisciCliente(MYSQL *conn){

    inserisci_cf:
    char *cf = takeCustomInput("Inserisci il codice fiscale (Devono essere 16 caratteri):", size_cf);
    if (strlen(cf) < 16){
        printf("\nCodice fiscale non valido: troppo piccolo\n");
        goto inserisci_cf;
    }

    char *nome = takeCustomInput("Inserisci il nome:",size_nome);
    char *cognome = takeCustomInput("Inserisci il cognome:", size_cognome);
    MYSQL_TIME data_nascita;

    printf("Data di nascita:\n");
    memset(&data_nascita, 0, sizeof(data_nascita));
    data_nascita.day = atoi(takeInput("Inserisci il giorno"));
    data_nascita.month = atoi(takeInput("Inserisci il mese"));
    data_nascita.year = atoi(takeInput("Inserisci l'anno"));

    char *email = takeCustomInput("Inserisci l'email:", size_email);
    char *fax = takeCustomInput("Inserisci il fax:", size_fax);
```

```
    inserisci_piva:
    char *piva = takeCustomInput("Inserisci la partita iva (Devono essere 11 caratteri)\t(premere invio se non si tratta di una società):",size_partitaiva );
    if (strlen(piva) > 0 && strlen(piva) < 11){
        printf("\nPartita Iva non valida: troppo piccola\n");
        goto inserisci_piva;
    }
    bool is_null_piva = true;
    char *nome_società = "";
    bool is_null_nome_società = true;

    if (strcmp(piva, "") != 0){
        is_null_piva = false;
        inserisci_società:
        nome_società = takeCustomInput("Inserisci il nome della società", size_società);
        if (strcmp(nome_società, "") == 0){
            printf("Attenzione: inserire il nome della società\n");
            goto inserisci_società;
        }
        is_null_nome_società = false;
    }

    MYSQL_STMT *stmt;
    stmt = mysql_stmt_init(conn);
    if(stmt == NULL){
        fprintf(stderr,"Errore nella init dello statement\n");
        return;
    }
    char* stmtStr = "CALL inserisci_cliente(?,?,?,?,?,?,?,?);";
    if(mysql_stmt_prepare(stmt, stmtStr,strlen(stmtStr))!=0){
        stmtError(stmt,"Errore nella prepare statement\n", true);
        return;
    }

    MYSQL_BIND ps_params[8];
    memset(ps_params, 0, sizeof (ps_params));
    ps_params[0].buffer_type = MYSQL_TYPE_STRING;
    ps_params[0].buffer = cf;
    ps_params[0].buffer_length = strlen(cf);

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

    ps_params[2].buffer_type = MYSQL_TYPE_VAR_STRING;
    ps_params[2].buffer = cognome;
    ps_params[2].buffer_length = strlen(cognome);

    ps_params[3].buffer_type = MYSQL_TYPE_DATE;
    ps_params[3].buffer = (char*)&data_nascita;
```

```
    ps_params[3].buffer_length = sizeof(data_nascita);

    ps_params[4].buffer_type = MYSQL_TYPE_VAR_STRING;
    ps_params[4].buffer = email;
    ps_params[4].buffer_length = strlen(email);

    ps_params[5].buffer_type = MYSQL_TYPE_VAR_STRING;
    ps_params[5].buffer = fax;
    ps_params[5].buffer_length = strlen(fax);

    ps_params[6].buffer_type = MYSQL_TYPE_VAR_STRING;
    ps_params[6].buffer = piva;
    ps_params[6].buffer_length = strlen(piva);
    ps_params[6].is_null = &is_null_piva;

    ps_params[7].buffer_type = MYSQL_TYPE_VAR_STRING;
    ps_params[7].buffer = nome_società;
    ps_params[7].buffer_length = strlen(nome_società);
    ps_params[7].is_null = &is_null_nome_società;

    if ( mysql_stmt_bind_param(stmt, ps_params) != 0 ){
        stmtError(stmt, "Impossibile fare il bind dei parametri per Inserisci Cliente", true);
        return;
    }
    if ( mysql_stmt_execute(stmt) != 0){
        stmtPrintError(stmt, "Errore nell'inserimento del cliente");
    }
    else{
        printf("\n\nIl cliente è stato inserito con successo!\n");
    }
    mysql_stmt_close(stmt);
}



void cmdVisualizzaProposte(MYSQL *conn){
    bool cont = true;

    char* stmtStr = "CALL visualizza_proposte();";

    visualizza_proposte:

    MYSQL_STMT *stmt;
    if(!init_stmt(&stmt,stmtStr)){
        return;
    }

    if (mysql_stmt_execute(stmt)!= 0){
        stmtError(stmt, "Non è possibile visualizzare l'elenco delle proposte", true);
        return;
```

```
    }
    if (mysql_stmt_store_result(stmt)) {
        stmtError(stmt, "Impossibile salvare lo statement", true);
        return;
    }

    struct proposte *proposta;
    int max = parse_proposte(stmt, &proposta);
    mysql_stmt_data_seek(stmt,0);

    printResult(stmt);
    mysql_stmt_close(stmt);

     if(max == 0){
        printf("\n\nNon ci sono ancora proposte da visualizzare");
        return;
    }

    while(cont){
        printf("\n\nI comandi disponibili sono: \n - visualizza proposta \n - back \n");
        char *cmd = takeInput("Inserisci il comando da eseguire");

        if(strcmp(cmd,"back")==0){
            cont=false;
        }
        else if(strcmp(cmd,"visualizza proposta") == 0){
            cmdVisualizzaProposta(conn, proposta, max);
            free(proposta);
            goto visualizza_proposte;
        }
        else{
            printf("\nComando non riconosciuto!\n");
        }
    }
    free(proposta);
}

void cmdVisualizzaProposta(MYSQL *conn,struct proposte *proposta, int max){
    bool cont = true;
    char codice[11];

    inserire_numero:

    int number = atoi(takeInput("Inserisci numero della proposta:"));

    if(number <= 0 || number > max){
        printf("Il numero inserito non corrisponde a nessuna proposta\n");
        goto inserire_numero;
    }
```

```
    strcpy(codice,proposta[number-1].codice);

    char* stmtStr = "CALL visualizza_proposta(?);";
    MYSQL_STMT *stmt;
    if(!init_stmt(&stmt,stmtStr)){
        return;
    }

    MYSQL_BIND ps_params[1];
    memset(ps_params, 0, sizeof (ps_params));

    ps_params[0].buffer_type = MYSQL_TYPE_VAR_STRING;
    ps_params[0].buffer = codice;
    ps_params[0].buffer_length = strlen(codice);

     if( mysql_stmt_bind_param(stmt, ps_params) != 0 ){
        stmtError(stmt, "Errore nel binding dei parametri per Visualizza Proposta",true);
        return;
    }
    if (mysql_stmt_execute(stmt)!= 0){
        stmtError(stmt, "Non è possibile visualizzare la proposta", true);
        return;
    }
    if (mysql_stmt_store_result(stmt)) {
        stmtError(stmt, "Impossibile salvare lo statement", true);
        return;
    }

    printResult(stmt);
    mysql_stmt_close(stmt);

    while(cont){
        printf("\n\nI comandi disponibili sono: \n - modifica disponibilità \n - back \n");
        char *cmd = takeInput("Inserisci il comando da eseguire");

        if(strcmp(cmd,"back")==0){
            cont=false;
        }
        else if(strcmp(cmd,"modifica disponibilità") == 0){
            cmdModificaDisponibilità(conn, codice);
        }
        else{
            printf("\nComando non riconosciuto!\n");
        }
    }
}

void cmdModificaDisponibilità(MYSQL *conn, char *codice){
    bool disp;
    inserisci_disp:
```

```
   char *disponibilità = takeInput("Inserire 0 o 1 ( 0 = setta la proposta come 'Non disponibile', 1 =
setta la proposta come 'Disponibile'");

   if ( strcmp(disponibilità,"0") == 0){
      disp = false;
   }
   else if(strcmp(disponibilità,"1") == 0){
      disp = true;
   }
   else{
      printf("Il valore che è stato inserito non è valido\n");
      goto inserisci_disp;
   }

   MYSQL_STMT *stmt;
   stmt = mysql_stmt_init(conn);
   if(stmt == NULL){
      fprintf(stderr,"Errore nella init dello statement\n");
      return;
   }
   char* stmtStr = "CALL modifica_disponibilita_proposta(?,?);";
   if(mysql_stmt_prepare(stmt, stmtStr,strlen(stmtStr))!=0){
      stmtError(stmt,"Errore nella prepare statement\n", true);
      return;
   }

   MYSQL_BIND ps_params[2];
   memset(ps_params, 0, sizeof (ps_params));

   ps_params[0].buffer_type = MYSQL_TYPE_VAR_STRING;
   ps_params[0].buffer = codice;
   ps_params[0].buffer_length = strlen(codice);

   ps_params[1].buffer_type = MYSQL_TYPE_TINY;
   ps_params[1].buffer = &disp;
   ps_params[1].buffer_length = sizeof(&disp);


   if( mysql_stmt_bind_param(stmt, ps_params) != 0 ){
      stmtError(stmt, "Errore nel binding dei parametri per Modifica Disponibilità",true);
      return;
   }
   if (mysql_stmt_execute(stmt)!= 0){
      stmtPrintError(stmt, "Non è possibile  modificare la disponibilità");
   }
   else{
      printf("\n\nLa disponibilità è stata modifica con successo!\n");
   }

   mysql_stmt_close(stmt);
```

```
}


void cmdContaProposteAcquistate(){

    char* stmtStr = "CALL conta_proposte_acquistate();";

    MYSQL_STMT *stmt;
    if(!init_stmt(&stmt,stmtStr)){
        return;
    }

    if (mysql_stmt_execute(stmt)!= 0){
        stmtError(stmt, "Non è possibile visualizzare l'elenco delle proposte che sono state acquistate",
true);
        return;
    }
    if (mysql_stmt_store_result(stmt)) {
        stmtError(stmt, "Impossibile salvare lo statement", true);
        return;
    }

    printResult(stmt);
    mysql_stmt_close(stmt);
}

void cmdInserisciProposta(MYSQL *conn, char *loggedUser){
    char *codice = takeCustomInput("Inserisci il codice della proposta:", size_proposta);
    char *descrizione = takeCustomInput("Inserisci la descrizione (massimo 100 caratteri):",
size_descrizione);

    MYSQL_STMT *stmt;
    stmt = mysql_stmt_init(conn);
    if(stmt == NULL){
        fprintf(stderr,"Errore nella init dello statement\n");
        return;
    }
    char* stmtStr = "CALL inserisci_proposta(?,?,?);";
    if(mysql_stmt_prepare(stmt, stmtStr,strlen(stmtStr))!=0){
        stmtError(stmt,"Errore nella prepare statement\n", true);
        return;
    }

    MYSQL_BIND ps_params[3];
    memset(ps_params, 0, sizeof (ps_params));
    ps_params[0].buffer_type = MYSQL_TYPE_STRING;
    ps_params[0].buffer = codice;
    ps_params[0].buffer_length = strlen(codice);

    ps_params[1].buffer_type = MYSQL_TYPE_STRING;
```

```
    ps_params[1].buffer = descrizione;
    ps_params[1].buffer_length = strlen(descrizione);

    ps_params[2].buffer_type = MYSQL_TYPE_STRING;
    ps_params[2].buffer = loggedUser;
    ps_params[2].buffer_length = strlen(loggedUser);

    if( mysql_stmt_bind_param(stmt, ps_params) != 0 ){
        stmtError(stmt, "Errore nel binding dei parametri per Inserisci Proposta",true);
        return;
    }
    if (mysql_stmt_execute(stmt) != 0){
        stmtPrintError(stmt, "Non è stato possibile inserire la proposta");
    }
    else{
        printf("\n\nProposta inserita con successo!\n");
    }
    mysql_stmt_close(stmt);
}

void cmdVisualizzaUtenti(MYSQL *conn){
    bool cont = true;

    char* stmtStr = "CALL visualizza_utenti();";

    visualizza_utenti:

    MYSQL_STMT *stmt;
    if(!init_stmt(&stmt,stmtStr)){
        return;
    }

    if (mysql_stmt_execute(stmt)!= 0){
        stmtError(stmt, "Non è possibile visualizzare l'elenco degli utenti", true);
        return;
    }
    if (mysql_stmt_store_result(stmt)) {
        stmtError(stmt, "Impossibile salvare lo statement", true);
        return;
    }

    printResult(stmt);
    mysql_stmt_close(stmt);

    while(cont){
        printf("\n\nI comandi disponibili sono: \n - inserisci utente \n - back \n");
        char *cmd = takeInput("Inserisci il comando da eseguire");

        if(strcmp(cmd,"back")==0){
            cont=false;
```

```
        }
        else if(strcmp(cmd,"inserisci utente") == 0){
            cmdInserisciUtente(conn);
            goto visualizza_utenti;
        }
        else{
            printf("\nComando non riconosciuto!\n");
        }
    }
}

void cmdInserisciUtente(MYSQL *conn){
    char *username = takeCustomInput("Inserisci l'username (massimo 20 caratteri):",
size_username);
    char *password = takeCustomInput("Inserisci la password (massimo 32 caratteri):",
size_password);
    char *nome = takeCustomInput("Inserisci il nome:", size_nome);
    char *cognome = takeCustomInput("Inserisci il cognome:",size_cognome);
    MYSQL_TIME data_nascita;

    printf("Data di nascita:\n");
    memset(&data_nascita, 0, sizeof(data_nascita));
    data_nascita.day = atoi(takeInput("Inserisci il giorno"));
    data_nascita.month = atoi(takeInput("Inserisci il mese"));
    data_nascita.year = atoi(takeInput("Inserisci l'anno"));

    char *email = takeCustomInput("Inserisci l'email:", size_email);
    char *ruolo = takeCustomInput("Scrivi O oppure M oppure F ( O = operatore, M = manager, F =
Funzionario Commerciale:", size_ruolo);

    MYSQL_STMT *stmt;
    stmt = mysql_stmt_init(conn);
    if(stmt == NULL){
        fprintf(stderr,"Errore nella init dello statement\n");
        return;
    }
    char* stmtStr = "CALL inserisci_utente(?,?,?,?,?,?,?);";
    if(mysql_stmt_prepare(stmt, stmtStr,strlen(stmtStr))!=0){
        stmtError(stmt,"Errore nella prepare statement\n", true);
        return;
    }

    MYSQL_BIND ps_params[7];
    memset(ps_params, 0, sizeof (ps_params));
    ps_params[0].buffer_type = MYSQL_TYPE_STRING;
    ps_params[0].buffer = username;
    ps_params[0].buffer_length = strlen(username);

    ps_params[1].buffer_type = MYSQL_TYPE_STRING;
    ps_params[1].buffer = password;
```

```
    ps_params[1].buffer_length = strlen(password);

    ps_params[2].buffer_type = MYSQL_TYPE_STRING;
    ps_params[2].buffer = nome;
    ps_params[2].buffer_length = strlen(nome);

    ps_params[3].buffer_type = MYSQL_TYPE_STRING;
    ps_params[3].buffer = cognome;
    ps_params[3].buffer_length = strlen(cognome);

    ps_params[4].buffer_type = MYSQL_TYPE_DATE;
    ps_params[4].buffer = (char*)&data_nascita;
    ps_params[4].buffer_length = sizeof(data_nascita);

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

    ps_params[6].buffer_type = MYSQL_TYPE_STRING;
    ps_params[6].buffer = ruolo;
    ps_params[6].buffer_length = strlen(ruolo);

    if( mysql_stmt_bind_param(stmt, ps_params) != 0 ){
        stmtError(stmt, "Errore nel binding dei parametri per Inserisci Utente",true);
        return;
    }
    if (mysql_stmt_execute(stmt) != 0){
        stmtPrintError(stmt, "Non è stato possibile inserire l'utente");
    }
    else{
        printf("\n\nUtente inserito con successo!\n");
    }
    mysql_stmt_close(stmt);
}
```

- Structures.c

```
#include "structures.h"
#include "utility.h"

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


int parse_clienti(MYSQL_STMT *stmt, struct clienti **ret){
    int status;
    int row = 0;
    int num_fields = mysql_stmt_field_count(stmt);
    MYSQL_BIND ps_params[num_fields];
```

```
bool is_null_nome_società = false;

char cf[17];
char nome[25];
char cognome[25];
char email[45];
char nome_società[45];

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

memset(ps_params, 0, sizeof(ps_params));

ps_params[0].buffer_type = MYSQL_TYPE_STRING;
ps_params[0].buffer = cf;
ps_params[0].buffer_length = sizeof(cf);

ps_params[1].buffer_type = MYSQL_TYPE_STRING;
ps_params[1].buffer = nome;
ps_params[1].buffer_length = sizeof(nome);

ps_params[2].buffer_type = MYSQL_TYPE_STRING;
ps_params[2].buffer = cognome;
ps_params[2].buffer_length = sizeof(cognome);

ps_params[3].buffer_type = MYSQL_TYPE_VAR_STRING;
ps_params[3].buffer = email;
ps_params[3].buffer_length = sizeof(email);

ps_params[4].buffer_type = MYSQL_TYPE_VAR_STRING;
ps_params[4].buffer = nome_società;
ps_params[4].buffer_length = sizeof(nome_società);
ps_params[4].is_null = &is_null_nome_società;

if(mysql_stmt_bind_result(stmt, ps_params)) {
    stmtError(stmt, "Impossibile fare il bind dei parametri\n", true);
}

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

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

    strcpy((*ret)[row].cf, cf);
    strcpy((*ret)[row].nome, nome);
    strcpy((*ret)[row].cognome, cognome);
    strcpy((*ret)[row].email, email);

    if(is_null_nome_società) {
```

```
            strcpy((*ret)[row].società, "");
        } else {
            strcpy((*ret)[row].società, nome_società);
        }
        row++;
    }
    return row;
}

int parse_appuntamenti(MYSQL_STMT *stmt, struct appuntamenti **ret){
    int status;
    int row = 0;
    int num_fields = mysql_stmt_field_count(stmt);
    MYSQL_BIND ps_params[num_fields];

    MYSQL_TIME data_inizio;
    char sala[4];
    char via[46];
    char città[21];
    char cliente[17];

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

    memset(ps_params, 0, sizeof(ps_params));

    ps_params[0].buffer_type = MYSQL_TYPE_DATETIME;
    ps_params[0].buffer = &data_inizio;
    ps_params[0].buffer_length = sizeof(data_inizio);

    ps_params[1].buffer_type = MYSQL_TYPE_VAR_STRING;
    ps_params[1].buffer = sala;
    ps_params[1].buffer_length = sizeof(sala);

    ps_params[2].buffer_type = MYSQL_TYPE_VAR_STRING;
    ps_params[2].buffer = via;
    ps_params[2].buffer_length = sizeof(via);

    ps_params[3].buffer_type = MYSQL_TYPE_VAR_STRING;
    ps_params[3].buffer = città;
    ps_params[3].buffer_length = sizeof(città);

    ps_params[4].buffer_type = MYSQL_TYPE_VAR_STRING;
    ps_params[4].buffer = cliente;
    ps_params[4].buffer_length = sizeof(cliente);


    if(mysql_stmt_bind_result(stmt, ps_params)) {
        stmtError(stmt, "Impossibile fare il bind dei parametri\n", true);
    }
```

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

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

        memcpy(&(*ret)[row].inizio , &data_inizio, sizeof(data_inizio));

        strcpy((*ret)[row].sala, sala);
        strcpy((*ret)[row].via, via);
        strcpy((*ret)[row].città, città);
        strcpy((*ret)[row].cliente, cliente);

        row++;
    }
    return row;
}

int parse_contatti(MYSQL_STMT *stmt, struct contatti **ret){
    int status;
    int row = 0;
    int num_fields = mysql_stmt_field_count(stmt);
    MYSQL_BIND ps_params[num_fields];

    int contatto;
    char cliente[17];
    MYSQL_TIME data;

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

    memset(ps_params, 0, sizeof(ps_params));

    ps_params[0].buffer_type = MYSQL_TYPE_LONG;
    ps_params[0].buffer = &contatto;
    ps_params[0].buffer_length = sizeof(contatto);

    ps_params[1].buffer_type = MYSQL_TYPE_STRING;
    ps_params[1].buffer = cliente;
    ps_params[1].buffer_length = sizeof(cliente);

    ps_params[2].buffer_type = MYSQL_TYPE_DATE;
    ps_params[2].buffer = (char*)&data;
    ps_params[2].buffer_length = sizeof(data);

    if(mysql_stmt_bind_result(stmt, ps_params)) {
        stmtError(stmt, "Impossibile fare il bind dei parametri\n", true);
    }

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

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

        (*ret)[row].IDcontatto = contatto;
        strcpy((*ret)[row].cliente, cliente);
        memcpy(&(*ret)[row].data , &data, sizeof(data));

        row++;
    }
    return row;
}

int parse_proposte(MYSQL_STMT *stmt, struct proposte **ret){
    int status;
    int row = 0;
    int num_fields = mysql_stmt_field_count(stmt);
    MYSQL_BIND ps_params[num_fields];

    char codice[11];
    bool disponibilità;
    char descrizione[101];
    char manager[21];

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

    memset(ps_params, 0, sizeof(ps_params));

    ps_params[0].buffer_type = MYSQL_TYPE_VAR_STRING;
    ps_params[0].buffer = codice;
    ps_params[0].buffer_length = sizeof(codice);

    ps_params[1].buffer_type = MYSQL_TYPE_TINY;
    ps_params[1].buffer = &disponibilità;
    ps_params[1].buffer_length = sizeof(&disponibilità);

    ps_params[2].buffer_type = MYSQL_TYPE_VAR_STRING;
    ps_params[2].buffer = descrizione;
    ps_params[2].buffer_length = sizeof(descrizione);

    ps_params[3].buffer_type = MYSQL_TYPE_VAR_STRING;
    ps_params[3].buffer = manager;
    ps_params[3].buffer_length = sizeof(manager);

    if(mysql_stmt_bind_result(stmt, ps_params)) {
        stmtError(stmt, "Impossibile fare il bind dei parametri\n", true);
    }

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

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

        strcpy((*ret)[row].codice, codice);
        (*ret)[row].disponibilità = disponibilità;
        strcpy((*ret)[row].descrizione, descrizione);
        strcpy((*ret)[row].manager, manager);

        row++;
    }
    return row;
}
```

- Utility.c

```
#include "utility.h"

#include <stdbool.h>
#include <stdio.h>
#include <string.h>


char* takeCustomInput(char* info, int size){
    char* cmd = malloc(size);
    char ch;
    printf("%s\n",info);

    fgets(cmd, size, stdin);

    if(cmd[strlen(cmd)-1] != '\n'){
        do{
            ch = getchar();
        }
        while( ch != '\n');
    }

    cmd[strlen(cmd)-1] = '\0';

    return cmd;
}

char* takeInput(char* info){
    int size = 35;
    char* cmd = malloc(size);
    char ch;
    printf("%s\n",info);

    fgets(cmd, size, stdin);
```

```
    if(cmd[strlen(cmd)-1] != '\n'){
        do{
            ch = getchar();
        }
        while( ch != '\n');
    }


    cmd[strlen(cmd)-1] = '\0';
    return cmd;
}

void stmtPrintError(MYSQL_STMT *stmt, char *msg){
    fprintf(stderr,"\n\nErrore: %s\n", msg);
    if ( strcmp(mysql_stmt_sqlstate(stmt),"22007") == 0){
        printf("La data inserita non è valida\n");
    }
    else if( strcmp(mysql_stmt_sqlstate(stmt),"45000") == 0){
        fprintf(stderr,"Errore %u (%s): %s\n", mysql_stmt_errno(stmt), mysql_stmt_sqlstate(stmt),
mysql_stmt_error(stmt));
    }
}

void stmtError( MYSQL_STMT *stmt, char *msg, bool close_stmt){
    stmtPrintError(stmt,msg);
    if (close_stmt) mysql_stmt_close(stmt);

}

void checkError(MYSQL_STMT *stmt, char *msg, char *msg1){
    if ( strcmp(mysql_stmt_sqlstate(stmt),"23000") == 0){
        if ( mysql_stmt_errno(stmt) == 1452 ){
            printf("%s",msg);
        }
        else if( mysql_stmt_errno(stmt) == 1062 ){
            printf("%s",msg1);
        }
    }
}

void print_dashes(MYSQL_RES *res_set){
    MYSQL_FIELD *field;
    unsigned int i, j;
    mysql_field_seek(res_set, 0);
    putchar('+');
    putchar('-');
    putchar('-');
    putchar('-');
    putchar('-');
```

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

void printResultHeader(MYSQL_RES *res_set){
    MYSQL_FIELD *field;
    unsigned long col_len;
    unsigned int i;
    mysql_field_seek (res_set, 0);
    for (i = 0; i < mysql_num_fields (res_set); i++) {
        field = mysql_fetch_field (res_set);
        if (strcmp(field->name,"DISPONIBILITA'")==0){
            field->max_length = 15;
        }

        col_len = strlen(field->name);
        if (col_len < field->max_length)
            col_len = field->max_length;
        if (col_len < 4 && !IS_NOT_NULL(field->flags))
            col_len = 4;
        field->max_length = col_len;
    }
    print_dashes(res_set);
    putchar('|');
    mysql_field_seek (res_set, 0);
    printf(" %-*s |", 3, "");
    for (i = 0; i < mysql_num_fields(res_set); i++) {
        field = mysql_fetch_field(res_set);
        printf(" %-*s |", (int)field->max_length, field->name);
    }
    putchar('\n');
    print_dashes(res_set);
}

void setLenght(MYSQL_RES *res_set){
    MYSQL_FIELD *field;
    unsigned long col_len;
    unsigned int i;
    mysql_field_seek (res_set, 0);
    for (i = 0; i < mysql_num_fields (res_set); i++) {
        field = mysql_fetch_field (res_set);
        if (strcmp(field->name,"DISPONIBILITA'")==0){
            field->max_length = 15;
```

```
        }
        col_len = strlen(field->name);

        if (col_len < field->max_length)
            col_len = field->max_length;
        if (col_len < 4 && !IS_NOT_NULL(field->flags))
            col_len = 4;
        field->max_length = col_len;
    }
}

void printResult( MYSQL_STMT *stmt){
    int i;
    int status;
    int num_fields;
    MYSQL_FIELD *fields;
    MYSQL_BIND *rs_bind;
    MYSQL_RES *rs_metadata;
    MYSQL_TIME *date;
    size_t attr_size;
    bool print;
    if (mysql_stmt_num_rows(stmt) > 0){
        print = true;
    }
    else{
        print = false;
    }

    num_fields = mysql_stmt_field_count(stmt);
    bool is_null[num_fields];
    if (num_fields > 0) {
        printf("\n ");

        if((rs_metadata = mysql_stmt_result_metadata(stmt)) == NULL) {
            stmtError(stmt, "Impossibile recuperare i metadati\n",true);
        }

        if (print) printResultHeader(rs_metadata);
        fields = mysql_fetch_fields(rs_metadata);
        rs_bind = (MYSQL_BIND *)malloc(sizeof (MYSQL_BIND) * num_fields);
        if (!rs_bind) {
            stmtError(stmt, "Impossibile allocare il buffer di output\n", true);
        }
        memset(rs_bind, 0, sizeof (MYSQL_BIND) * num_fields);
        for (i = 0; i < num_fields; ++i) {
            switch(fields[i].type) {
                case MYSQL_TYPE_DATE:
                case MYSQL_TYPE_TIMESTAMP:
                case MYSQL_TYPE_DATETIME:
                case MYSQL_TYPE_TIME:
```

```
            attr_size = sizeof(MYSQL_TIME);
            break;
        case MYSQL_TYPE_FLOAT:
            attr_size = sizeof(float);
            break;
        case MYSQL_TYPE_DOUBLE:
            attr_size = sizeof(double);
            break;
        case MYSQL_TYPE_TINY:
            attr_size = sizeof(signed char);
            break;
        case MYSQL_TYPE_SHORT:
        case MYSQL_TYPE_YEAR:
            attr_size = sizeof(short int);
            break;
        case MYSQL_TYPE_LONG:
        case MYSQL_TYPE_INT24:
            attr_size = sizeof(int);
            break;
        case MYSQL_TYPE_LONGLONG:
            attr_size = sizeof(int);
            break;
        case MYSQL_TYPE_BIT:
            attr_size = 1;
            break;
        default:
            attr_size = fields[i].max_length;
            break;
    }

    rs_bind[i].buffer_type = fields[i].type;
    rs_bind[i].buffer = malloc(attr_size + 1);
    rs_bind[i].buffer_length = attr_size + 1;
    rs_bind[i].is_null = &is_null[i];
    if(rs_bind[i].buffer == NULL) {
        stmtError(stmt, "Impossibile allocare il buffer di output\n",true);
    }
}
if(mysql_stmt_bind_result(stmt, rs_bind)) {
    stmtError(stmt, "Impossibile fare il bind dei parametri\n",true);
}

int index = 1;

while (true) {
    status = mysql_stmt_fetch(stmt);
    if (status == 1 || status == MYSQL_NO_DATA)
        break;
    if(print){
        putchar('|');
```

```
            setLenght(rs_metadata);
            printf (" %-*d |", 3, index);
            for (i = 0; i < num_fields; i++) {

                if (rs_bind[i].is_null_value) {
                    printf (" %-*s |", (int)fields[i].max_length, "NULL");
                    continue;
                }
                if(*rs_bind[i].is_null){
                    printf(" %-*s |", (int)fields[i].max_length, "NULL");
                    continue;
                }
                if(rs_bind[i].buffer_type == MYSQL_TYPE_TINY) {
                    if (*(char *)rs_bind[i].buffer == 0)
                        printf(" %-*s |", (int)fields[i].max_length, "Non disponibile");
                    if (*(char *)rs_bind[i].buffer == 1)
                        printf(" %-*s |", (int)fields[i].max_length, "Disponibile");
                } else {
                    switch (rs_bind[i].buffer_type) {
                        case MYSQL_TYPE_VAR_STRING:
                            printf(" %-*s |", (int)fields[i].max_length, (char*)rs_bind[i].buffer);
                            break;
                        case MYSQL_TYPE_DATETIME:
                            date = (MYSQL_TIME *)rs_bind[i].buffer;
                            printf(" %02d-%02d-%4d %02d:%02d:%02d%-*s |", date->day, date->month,
date->year,date->hour,date->minute,date->second,((int)fields[i].max_length)-19, "");
                            break;
                        case MYSQL_TYPE_DATE:
                            date = (MYSQL_TIME *)rs_bind[i].buffer;
                            printf(" %02d-%02d-%04d%-*s |", date->day,date->month, date->year,
((int)fields[i].max_length)-10, "");
                            break;
                        case MYSQL_TYPE_TIMESTAMP:
                            date = (MYSQL_TIME *)rs_bind[i].buffer;
                            printf(" %d-%02d-%02d %-*s |", date->year,date->month, date->day, 1, "");
                            break;
                        case MYSQL_TYPE_STRING:
                            printf(" %-*s |", (int)fields[i].max_length, (char*)rs_bind[i].buffer);
                            break;
                        case MYSQL_TYPE_FLOAT:
                        case MYSQL_TYPE_DOUBLE:
                            printf(" %.02f |", *(float *)rs_bind[i].buffer);
                            break;
                        case MYSQL_TYPE_LONG:
                        case MYSQL_TYPE_LONGLONG:
                        case MYSQL_TYPE_SHORT:
                        case MYSQL_TYPE_TINY:
                            printf(" %-*d |", (int)fields[i].max_length, *(int*)rs_bind[i].buffer);
                            break;
                        case MYSQL_TYPE_NEWDECIMAL:
```

```
                        printf(" %-*.02lf |", (int)fields[i].max_length,*(float*) rs_bind[i].buffer);
                        break;
                    case MYSQL_TYPE_BIT:
                        if (*rs_bind[i].is_null)
                            printf(" %-*s |", 15, "NULL");
                        else if ( strcmp( (char*) rs_bind[i].buffer, "") == 1 ) {

                            printf(" %-*s |", 15,"Disponibile");
                        } else {
                            printf(" %-*s |", 15,"Non disponibile");
                        }

                        break;
                    default:
                        printf("ERRORE:tipo sconosciuto (%d)\n", rs_bind[i].buffer_type);
                        abort();
                }
            }
        }
        putchar('\n');
        print_dashes(rs_metadata);
    }
    index++;
  }
  mysql_stmt_next_result(stmt);
  mysql_free_result(rs_metadata);

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

- sqlFunctions.h

```
#ifndef CODICEPROGETTO_SQLFUNCTIONS_H
#define CODICEPROGETTO_SQLFUNCTIONS_H
#include <mysql.h>

MYSQL* connectToDb();
void closeConnection();
void connect_as_login();
void connect_as_operatore();
void connect_as_manager();
void connect_as_commerciale();
bool init_stmt(MYSQL_STMT **stmt, char* procedure);

#endif //CODICEPROGETTO_SQLFUNCTIONS_H
```

- functions.h

```
#ifndef CODICEPROGETTO_FUNCTIONS_H
#define CODICEPROGETTO_FUNCTIONS_H
#include <mysql.h>
#include "structures.h"

int cmdLogin(MYSQL *conn, char *username, char *password);

void cmdVisualizzaProposteAccettate(char *cf);
void cmdVisualizzaProposteAcquistate(char *cf);
void cmdVisualizzaContattiCliente(char *cf);
void cmdVisualizzaCliente(struct clienti *cli, int max);
void cmdVisualizzaClientiOperatore(char *loggedUser);

void cmdModificaNotaContatto(MYSQL *conn, int IDcontatto);
void cmdEliminaNotaContatto(MYSQL *conn, int IDcontatto);
void cmdInserisciPropostaAccettata(MYSQL *conn, int IDcontatto);
void cmdVisualizzaContatto(MYSQL *conn,struct contatti *contatto, int max);
void cmdVisualizzaContattiOperatore(MYSQL *conn, char *loggedUser);

void cmdModificaResoconto(MYSQL *conn, MYSQL_TIME data_inizio, char *sala, char *via,
char *città);
void cmdInserisciPropostaAcquistata(MYSQL *conn, MYSQL_TIME data_inizio, char *sala, char
*via, char *città, char *cliente);
bool cmdModificaDataAppuntamento(MYSQL *conn, MYSQL_TIME data_inizio, char *sala, char
*via, char *città);
void cmdVisualizzaAppuntamento(MYSQL *conn, struct appuntamenti *app, int max);
void cmdVisualizzaAppuntamentiOperatore(MYSQL *conn, char *loggedUser);

void cmdVisualizzaSalaSede();
void cmdInserisciAppuntamento(MYSQL *conn, char *loggedUser);

void cmdInserisciContatto(MYSQL *conn, char *loggedUser);

void cmdVisualizzaProposteDisponibili();



void cmdModificaDisponibilità(MYSQL *conn, char *codice);
void cmdVisualizzaProposta(MYSQL *conn,struct proposte *proposta, int max);
void cmdVisualizzaProposte(MYSQL *conn);

void cmdContaProposteAcquistate();

void cmdInserisciProposta(MYSQL *conn, char *loggedUser);

void cmdInserisciCliente(MYSQL *conn);
```

```
void cmdVisualizzaClienti(MYSQL *conn);



void cmdInserisciUtente(MYSQL *conn);
void cmdVisualizzaUtenti(MYSQL *conn);


#endif //CODICEPROGETTO_FUNCTIONS_H
```

- structures.h

```
#ifndef CODICEPROGETTO_STRUCTURES_H
#define CODICEPROGETTO_STRUCTURES_H
#include <mysql.h>

struct clienti{
    char cf[17];
    char nome[26];
    char cognome[26];
    char nascita[11];
    char registrazione[11];
    char email[46];
    char fax[46];
    char piva[12];
    char società[46];
    char operatore[21];
};


struct appuntamenti{
    MYSQL_TIME inizio;
    char sala[4];
    char via[46];
    char città[21];
    char resoconto[301];
    char operatore[21];
    char cliente[17];
    MYSQL_TIME fine;

};


struct contatti{
    int IDcontatto;
    char nota[301];
    char cliente[17];
    char operatore[21];
    MYSQL_TIME data;
```

```
};

struct proposte{
   char codice[11];
   bool disponibilità;
   char descrizione[101];
   char manager[21];
};

extern int parse_clienti(MYSQL_STMT *stmt, struct clienti **ret);
extern int parse_appuntamenti(MYSQL_STMT *stmt, struct appuntamenti **ret);
extern int parse_contatti(MYSQL_STMT *stmt, struct contatti **ret);
extern int parse_proposte(MYSQL_STMT *stmt, struct proposte **ret);

#endif //CODICEPROGETTO_SQLFUNCTIONS_H
```

- utility.h

```
#ifndef CODICEPROGETTO_UTILITY_H
#define CODICEPROGETTO_UTILITY_H
#include <mysql.h>

char* takeCustomInput(char* info, int size);
char* takeInput(char* info);

void checkError(MYSQL_STMT *stmt, char *msg, char *msg1);
void stmtPrintError(MYSQL_STMT *stmt, char *msg);
void stmtError(MYSQL_STMT *stmt, char *msg, bool close_stmt);

void printError(MYSQL *conn, char *msg);

void printResult( MYSQL_STMT *stmt);


#endif //CODICEPROGETTO_UTILITY_H
```

- Makefile

```
all:

        gcc -g *.c -Wall -Wextra -o client `mysql_config --cflags --include --libs)`;


clean:

        -rm client
```