Basi di Dati e Conoscenza

Progetto A.A. 2019/2020

TITOLO DEL PROGETTO

Matricola

Nome e Cognome

Indice

1. Descrizione del Minimondo

Customer Relationship Management

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
6	Azienda	Sistema	Omonimia tra azienda marketing e azienda CRM
9	Società	Sistema	Ambiguità tra società e azienda e omonimia con azienda marketing
10	Di questi	Società	Ambiguità tra clienti e società(rif. linea 9)
12	Azienda di CRM	Azienda	Omonimia tra azienda marketing e azienda CRM
12	Funzionario	Operatore	Consistenza
13	Utente aziendale del sistema	Operatore dell'azienda	Consistenza
15	Azienda di CRM	azienda	Ambiguità tra azienda marketing e azienda CRM
15	Operatori dell'azienda di CRM	Operatori dell'azienda	Ambiguità tra azienda marketing e azienda CRM
16	Recapiti	Contatti	Ambiguità con recapiti telefonici(rif. linea 8)
16	Utente	Operatore	Consistenza
17	Interazione	Conversazione	Consistenza (rif. linea 37)
18	Positiva	Affermativa	Consistenza con risposte affermative
21	Sede	Sala riunione	Semanticamente piu' corretto
32	Dati dell'azienda	Dati della società	Consistenza
32	Referente aziendale	Referente della società	Consistenza
34	Servizi consulenza acquistati	Proposte commerciali accettate	Consistenza (rif. linea 18)
35	Utente	Operatore	Consistenza
36	Note	Note testuali	Consistenza
37	Note	Note testuali	Consistenza
38	Registrata	Inserita	Consistenza (rif. linea 16)
38	Utente	Operatore	Consistenza
39	Cliente	Manager	Ambiguità con cliente
41	Nota descrittiva	Proposta commerciale	Il campo proposta commerciale rimanda al motivo dell'appuntamento in sede.
43	Utente	Operatore	Consistenza
44	Servizi di consulenza	Proposte commerciali	Consistenza
46	Utenti	Operatori	Consistenza

Specifica disambiguata

Customer Relationship Management

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 del sistema 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 dal sistema di CRM. Della 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 in questione è di dimensione elevata ed ha a disposizione vari operatori che interagiscono con i clienti. A ciascun operatori dell'azienda viene assegnato un sottoinsieme di clienti da gestire.

Su base periodica,gli operatori dell'azienda contattano i clienti mediante uno dei contatti forniti. In questa fase operativa, il funzionario deve inserire una nota testuale in cui viene riportato un breve resoconto della conversazione con il cliente, annotando anche possibili risposte affermative alle proposte commerciali. Una risposta affermativa 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 riunione, nello stesso giorno ed alla stessa ora, a più di un cliente. Agli appuntamenti partecipano i clienti e gli operatori.

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 della società e del referente della società, con tutti i dettagli e le caratteristiche, l'elenco delle note cliente e l'elenco delle proposte commerciali accettate.
* Possibilità di visualizzare l'elenco clienti a cui un operatore è assegnato.
* Gestione delle note testuali del cliente: ogni volta che un cliente viene contattato deve essere possibile registrare/modificare/cancellare una o più note testuali relative alla conversazione

avvenuta e dell' operatore che l'ha inserita.

* Gestione delle opportunità: per ogni operatore deve essere possibile inserire una nuova opportunità, cioè una proposta commerciale accettata dal cliente.

* Gestione degli appuntamenti: deve essere possibile inserire un appuntamento con una proposta commerciale, una data/ora e un cliente a cui è riferito.

* Visualizzazione dell'agenda degli appuntamenti per un operatore.

* Possibilità di inserire nuove proposte commerciali (riservata ai manager).

* Possibilità di inserire nuovi clienti (riservata al settore commerciale).

* Possibilità di inserire nuovi operatori dell'applicativo web (riservata ai manager).

Glossario dei Termini

Termine	Descrizione	Sinonimi	Collegamenti
Cliente	Cliente dell'azienda. Può essere una società		Società,membro settore sociale,appuntamento, operatore, conversazione
Società	Cliente dell'azienda con partita IVA, referente di una società	Referente società	Cliente
Operatore	Operatore dell'azienda. Interagisce con un sottoinseme di clienti		Cliente, appuntamento, manager, conversazione
Proposta commerciale	Proposta commerciale che l'azienda offre o ha offerto, inserita da un Manager		Manager, prop. commer. disponibile, prop. commer. Terminata, appuntamento
Proposta commerciale disponibile	Proposta commerciale che un operatore può offrire a un cliente		Prop. commerciale, manager, apputamento
Proposta commerciale terminata	Proposta commerciale che non può più esser fornita a un cliente. Eliminata da un Manager		Prop. commer. ,manager
Sede	Sede dell'azienda identificata da un indirizzo		Sala riunione

Sala riunione	Presenti in una sede. Dove avviene l'appuntamento		Appuntamento, sede
Appuntamento	Avviene in una sala riunione tra un operatore e un cliente in una data e ora		Operatore , cliente, sala riunione, proposta commerciale
Manager	Manager dell'azienda. Inserisce e elimina proposte commerciali nel sistema CRM		Proposta commerciale, operatore
Membro settore commerciale	Membri dell'azienda. Reclutano nuovi clienti e li inseriscono nel sistema		Cliente
Conversazione	Interazione tra operatore e cliente , iniziata tramite uno dei contatti forniti. Può aver associate risposte positive a proposte commerciali		Operatore , cliente

Raggruppamento dei requisiti in insiemi omogenei

Frasi relative a cliente
La base di dati del sistema informativo del sistema 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. Di tutti i clienti interessa sapere qual è la data di registrazione nel sistema di CRM. Agli appuntamenti partecipano i clienti e gli operatori. * Visualizzare il singolo cliente, eventualmente con i dati dell'azienda e dell'operatore, con tutti i dettagli e le caratteristiche, l'elenco delle note cliente e l'elenco delle proposte commerciali accettate. * Possibilità di visualizzare l'elenco clienti a cui un operatore è assegnato. * Gestione delle note testuali del cliente: ogni volta che un cliente viene contattato deve essere possibile registrare/modificare/cancellare una o più note testuali relative alla conversazione avvenuta e dell'operatore che l'ha inserita. * Gestione degli appuntamenti: deve essere possibile inserire un appuntamento con una proposta commerciale, una data/ora e un cliente a cui è riferito (riservata all'operatore). * Possibilità di inserire nuovi clienti (riservata al settore commerciale).

Frasi relative a operatore
L'azienda in questione è di dimensione elevata ed ha a disposizione vari operatori che interagiscono con i clienti. A ciascun operatore dell'azienda viene assegnato un sottoinsieme di clienti da gestire. Su base periodica, gli operatore dell'azienda contattano i clienti mediante uno dei contatti forniti. In questa fase operativa, l'operatore deve inserire una nota testuale in cui viene riportato un breve

resoconto della conversazione con il cliente, annotando anche possibili risposte affermative alle proposte commerciali.
Una risposta affermativa di accettazione di una proposta commerciale può essere associata ad un appuntamento in sede.Agli appuntamenti partecipano i clienti e gli operatori.
* Visualizzare il singolo cliente, eventualmente con i dati dell'azienda e dell' operatore, con tutti i dettagli e le caratteristiche, l'elenco delle note cliente e l'elenco delle proposte commerciali accettate.
* Possibilità di visualizzare l'elenco clienti a cui un operatore è assegnato.
* Gestione delle note testuali del cliente: ogni volta che un cliente viene contattato deve essere possibile registrare/modificare/cancellare una o più note testuali relative alla conversazione avvenuta e dell' operatore che l'ha inserita.
* Gestione degli appuntamenti: deve essere possibile inserire un appuntamento con una proposta commerciale, una data/ora e un cliente a cui è riferito.
* Visualizzazione dell'agenda degli appuntamenti per un operatore.
* Possibilità di inserire nuovi operatori nell'applicativo web (riservata ai manager).

Frasi relative a società

Alcuni dei clienti sono società che ricevono servizi dal sistema di CRM. Della società interessa anche mantenere il numero di partita IVA.

Frasi relative a proposta commerciale

L' operatore deve inserire una nota testuale in cui viene riportato un breve resoconto della conversazione con il cliente, annotando anche possibili risposte affermative alle proposte commerciali. Una risposta affermativa di accettazione di una proposta commerciale può essere associata ad un appuntamento in sede.
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.
* Gestione delle opportunità: per ogni manager deve essere possibile inserire una nuova opportunità, cioè una proposta commerciale.

Frasi relative a 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 riunione, nello stesso giorno ed alla stessa ora, a più di un cliente. Agli appuntamenti partecipano i clienti e gli operatori.

Frasi relative a sala 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 riunione, nello stesso giorno ed alla stessa ora, a più di un cliente.

5

Frasi relative a appuntamento

Una risposta affermativa di accettazione di una proposta commerciale può essere associata ad un appuntamento in sede.Agli appuntamenti partecipano i clienti e gli operatori.

* Gestione degli appuntamenti: deve essere possibile inserire un appuntamento con una proposta commerciale, una data/ora e un cliente a cui è riferito.
* Visualizzazione dell'agenda degli appuntamenti per un operatore.

Frasi relative a manager

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.
* Gestione delle opportunità: per ogni manager deve essere possibile inserire una nuova opportunità, cioè una proposta commerciale.
* Possibilità di inserire nuovi servizi di consulenza (riservata ai manager).
* Possibilità di inserire nuovi operatori dell'applicativo web (riservata ai manager).

Frasi relative a membro settore commerciale

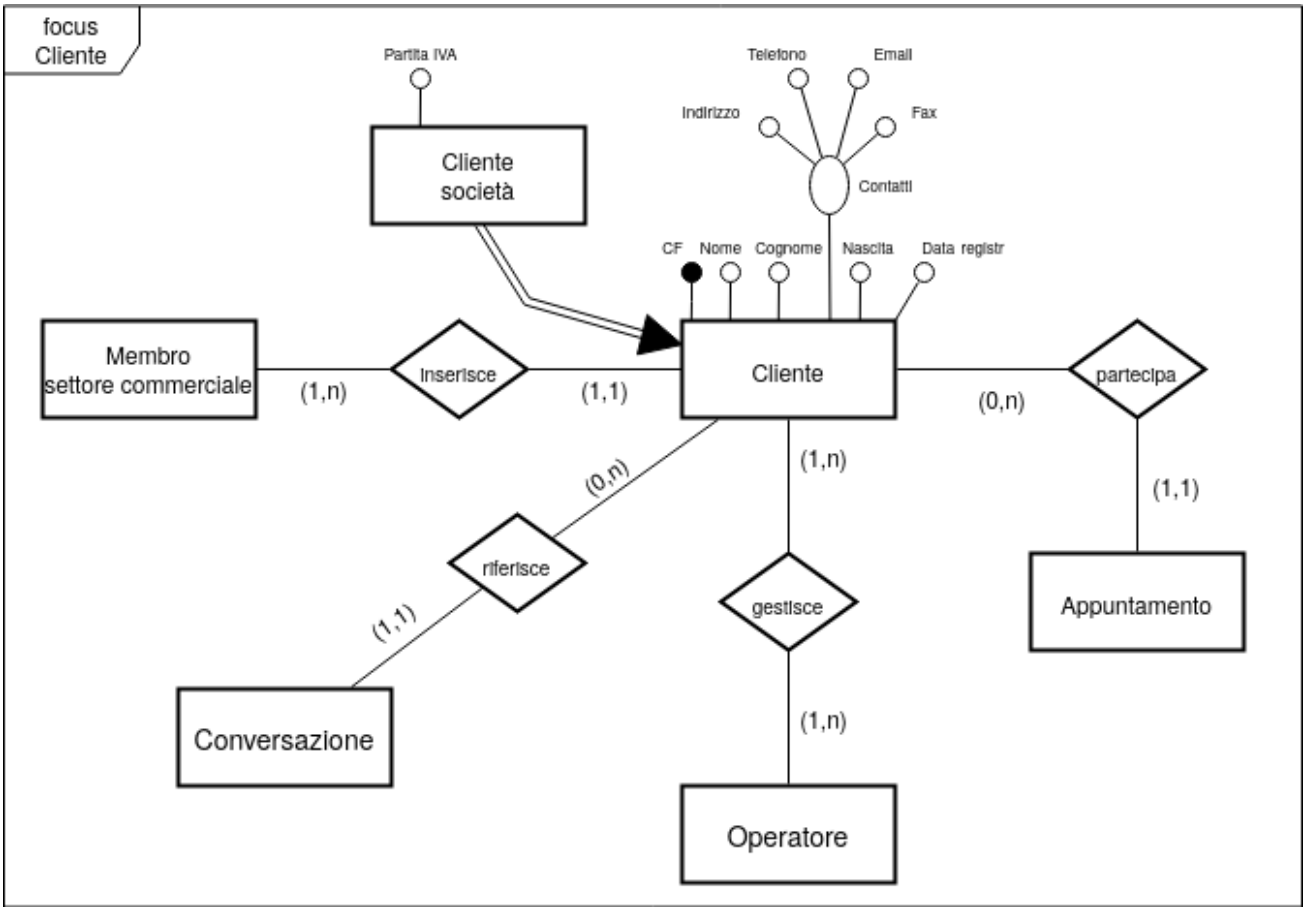
Infine, l'azienda ha un settore commerciale i cui membri reclutano nuovi clienti e li inseriscono all'interno del sistema.
* Possibilità di inserire nuovi clienti (riservata al settore commerciale).

Frasi relative a conversazione

L' operatore deve inserire una nota testuale in cui viene riportato un breve resoconto della conversazione con il cliente, annotando anche possibili risposte affermative alle proposte commerciali.
* Visualizzare il singolo cliente, eventualmente con i dati dell'azienda e dell' operatore, con tutti i dettagli e le caratteristiche, l'elenco delle note cliente e l'elenco delle proposte commerciali accettate.
* Gestione delle note testuali del cliente: ogni volta che un cliente viene contattato deve essere possibile registrare/modificare/cancellare una o più note testuali relative alla conversazione avvenuta e dell' operatore che l'ha inserita.

3. Progettazione concettuale

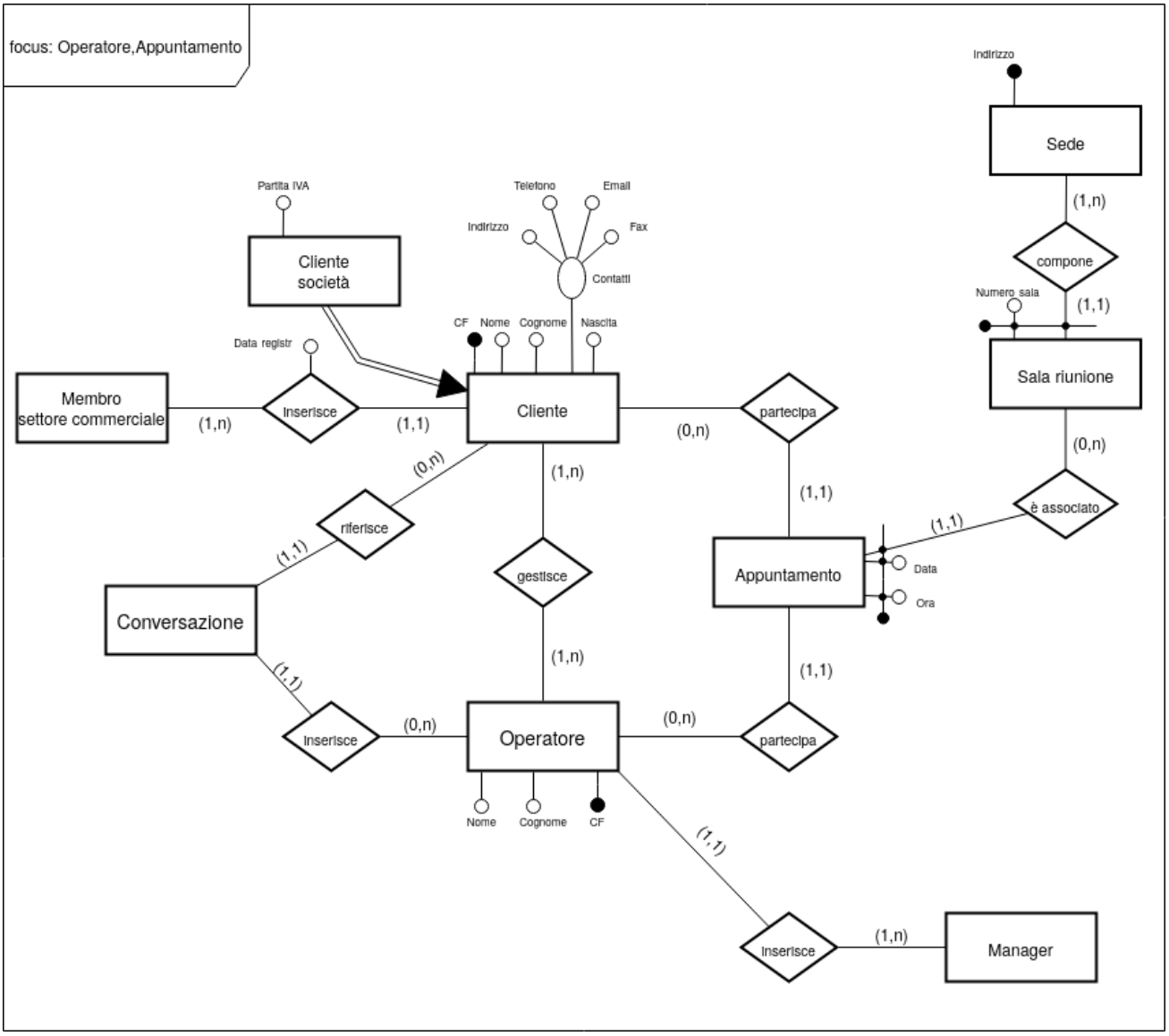
Costruzione dello schema E-R



Il primo passo effettuato è un focus sull'entità Cliente, quindi i suoi attributi il suo identificatore e tutte le eventuali relazioni con le altre entità del minimondo.

Il referente di una società (Cliente società) è stato rappresentato come un'entità a parte che specializza Cliente.

Si è assunto che un cliente possa venir gestito da più operatori, ma almeno 1. Di conseguenza si è assunto anche il fatto che I sottoinsiemi di gestione dei clienti da parte deglil operatori siano sottoinsiemi non disgiunti.

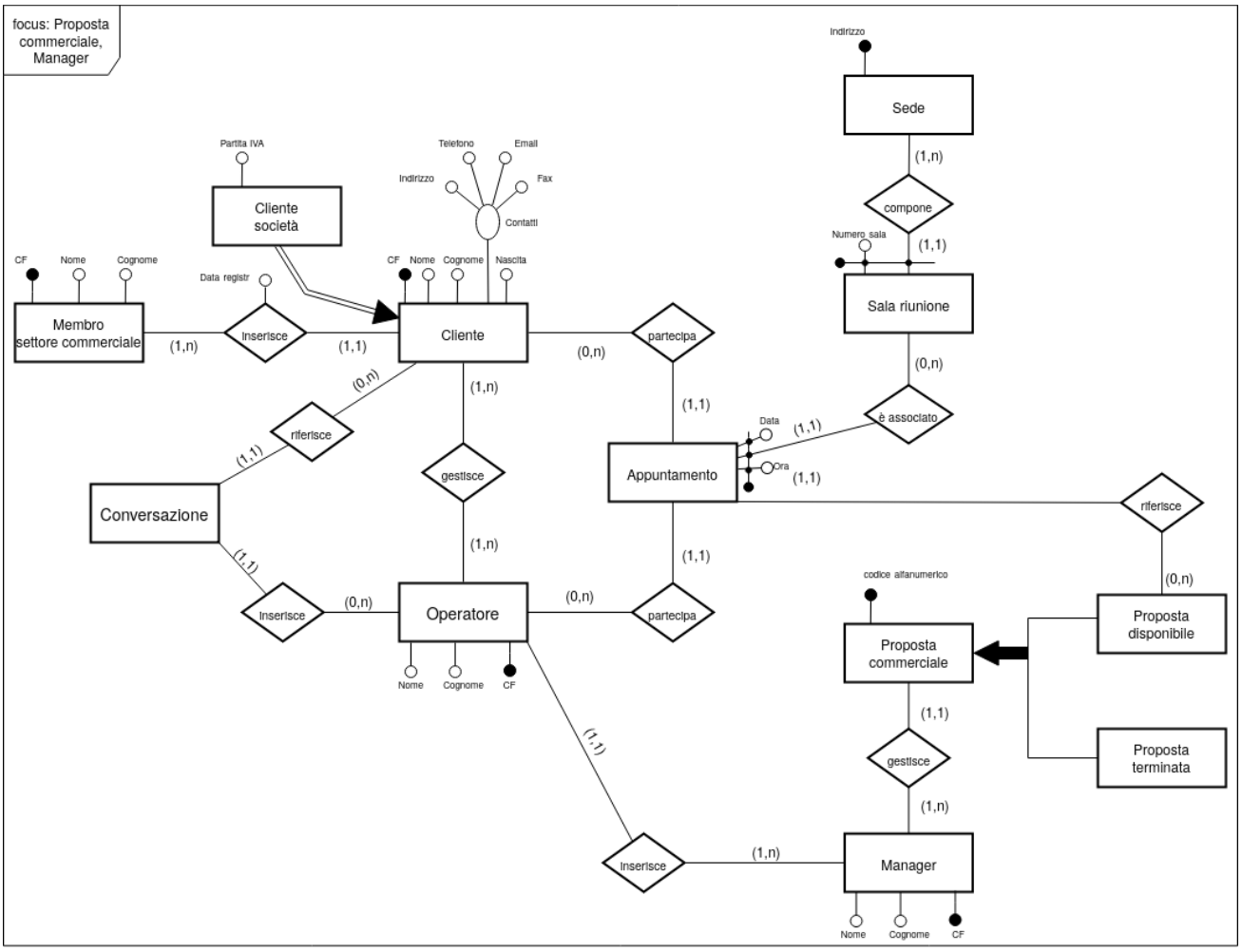


Il secondo passo effettuato è un focus sia sull'entità Operatore che sull'entità Appuntamento e mostrando , a macchia d'olio, le loro relazioni con altre entità del minimondo.

Si è assunto che in una sala riunione ad una data e un'ora ci può essere un solo appuntamento.

Si è assunto inoltre che ogni utente del minimondo che non sia un cliente (quindi operatore, manager, membro settore commerciale) sia identificato da un codice fiscale e si vuole salvare il suo nome e cognome.

Notare come l'attributo del cliente 'Data registrazione' è stato tolto e impostato come attributo della relazione 'inserisce'.

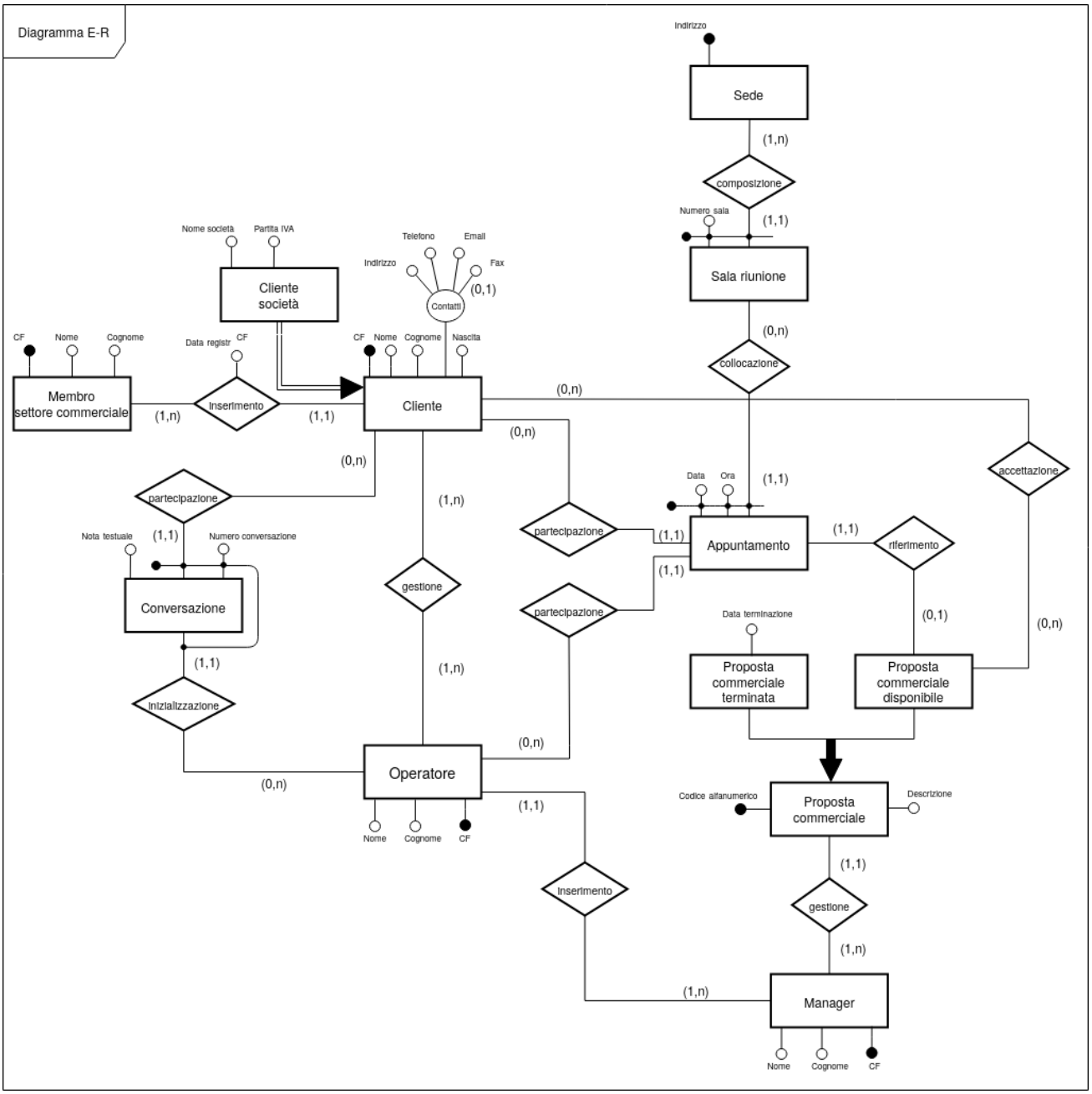


Il terzo passo effettuato è un focus sulle entità Proposta commerciale e Manager e le relazioni con le entità già descritte.

Si è descritto il fatto che un appuntamento deve riferire una proposta commerciale disponibile e si è assunto che il riferimento deve essere obbligatorio (scelta implementativa non descritta nella specifica).

Si vogliono salvare anche le proposte commerciali terminate.

Integrazione finale



5 L’ultimo passo è un focus sull’entità Conversazione e le sue relazioni con il resto del minimondo e aggiunta di alcuni attributi di alcune entità.

Importante è l’assunzione fatta sugli attributi di Conversazione.

Si è pensato di identificare la conversazione con la coppia Cliente-Operatore e un numero che

specifica su quale conversazione tra i due si sta operando. Cosi facendo la conversazione è identificata dalla tupla (cliente, operatore, numero) .

Infine, importante è la scelta di design di non riferire alcuna proposta commerciale nella coversazione, sarà cura dell'operatore riferirla nella nota testuale (la specifica non è stata chiara su questo punto).

Regole aziendali

(RV1) L' operatore che interagisce con un cliente attraverso una conversazione deve avere sotto gestione il cliente.

(RV2) L' operatore che inserisce un appuntamento con un cliente deve avere sotto gestione il cliente.

Dizionario dei dati

Entità	Descrizione	Attributi	Identificatori
Cliente	Cliente dell'azienda	CF, cognome, nome, nascita, contatti(indirizzo, telefono, email, fax)	CF
Cliente società	Cliente dell'azienda con partita IVA	CF, cognome, nome, nascita, contatti(indirizzo, telefono, email, fax), partita IVA, nome società	CF
Operatore	Operatore dell'azienda	CF, cognome, nome	CF
Appuntamento	Appuntamento tra cliente e operatore	Data, ora	Data, ora, sala riunione
Sede	Sede dell'azienda	Indirizzo	Indirizzo
Sala riunione	Sala riunione di una sede dove si svolge un appuntamento	Numero sala	Numero sala, sede
Membro settore commerciale	Membro del settore commerciale dell'azienda	CF, cognome, nome	CF
Conversazione	Conversazione tra cliente e operatore	Nota testuale, numero conversazione	Numero conversazione, cliente, operatore
Proposta commerciale	Proposta commerciale dell'azienda per i clienti	Codice alfanumerico, descrizione	Codice alfanumerico
Proposta commerciale disponibile	Proposta commerciale disponibile	Codice alfanumerico, descrizione	Codice alfanumerico

Proposta comm. terminata	Proposta commerciale terminata	Codice alfanumerico, descrizione, data terminazione	Codice alfanumerico
Manager	Manager dell'azienda	CF, cognome, nome	CF

4. Progettazione logica

Volume dei dati

Concetto nello schema	Tipo	Volume atteso
Cliente	E	50 000
Cliente società	E	10 000
Operatore	E	2 000
Conversazione	E	200 000
Appuntamento	E	10 000
Sede	E	50
Sala riunione	E	1 000
Membro settore commerciale	E	500
Proposta commerciale	E	1 000
Proposta commerciale disponibile	E	400
Proposta commerciale terminata	E	600
Manager	E	500
Inserimento (Membro settore-Cliente)	R	50 000
Partecipazione	R	200 000
Inizializzazione	R	200 000
Accettazione	R	30 000
Riferimento (Appuntamento-Prop.comm.disp.)	R	10 000
Partecipazione (Cliente-Appuntamento)	R	10 000
Partecipazione (Operatore-Appuntamento)	R	10 000
Collocazione	R	10 000
Composizione	R	1 000
Gestione (Operatore-Cliente)	R	60 000
Inserimento (Manager-Operatore)	R	2 000
Gestione(Manager-Prop.comm)	R	1 000

5 Tavola delle operazioni

Cod.	Descrizione	Frequenza attesa
1	Visualizzare il singolo cliente, eventualmente con i dati della società e del referente della società, con tutti i dettagli e le caratteristiche, l’elenco delle note cliente e l’elenco delle proposte commerciali accettate.	100 000/giorno
2	Possibilità di visualizzare l’elenco clienti a cui un operatore è assegnato.	20 000/giorno

3	Gestione delle note testuali del cliente: ogni volta che un cliente viene contattato deve essere possibile registrare/modificare/cancellare una o più note testuali relative alla conversazione avvenuta e dell' operatore che l'ha inserita.	10 000/giorno
4	Inserire una nuova proposta commerciale (da parte di un manager)	50/giorno
5	Gestione degli appuntamenti: deve essere possibile inserire un appuntamento con una proposta commerciale, una data/ora e un cliente a cui è riferito.	1 000/giorno
6	Visualizzazione dell'agenda degli appuntamenti per un operatore.	4 000/giorno
7	Gestione delle opportunità: per ogni operatore deve essere possibile inserire una nuova opportunità, cioè una proposta commerciale accettata dal cliente.	200/giorno
8	Inserire un nuovo cliente nel sistema (da parte di un membro del sett. commer.)	1 000/giorno
9	Inserire un nuovo operatore nel sistema (da parte di un manager)	50/giorno

Costo delle operazioni

Op 1: (1+1+1+1)*100 000 = 400 000 accessi/giorno

Op 2: (1+1)*20 000 = 40 000 accessi/giorno

Op 3: (1+2)*10 000= 30 000 accessi/giorno

Op 4: (2)*50 = 100 accessi/giorno

Op 5: (2+1+1)*1 000 = 4 000 accessi/giorno

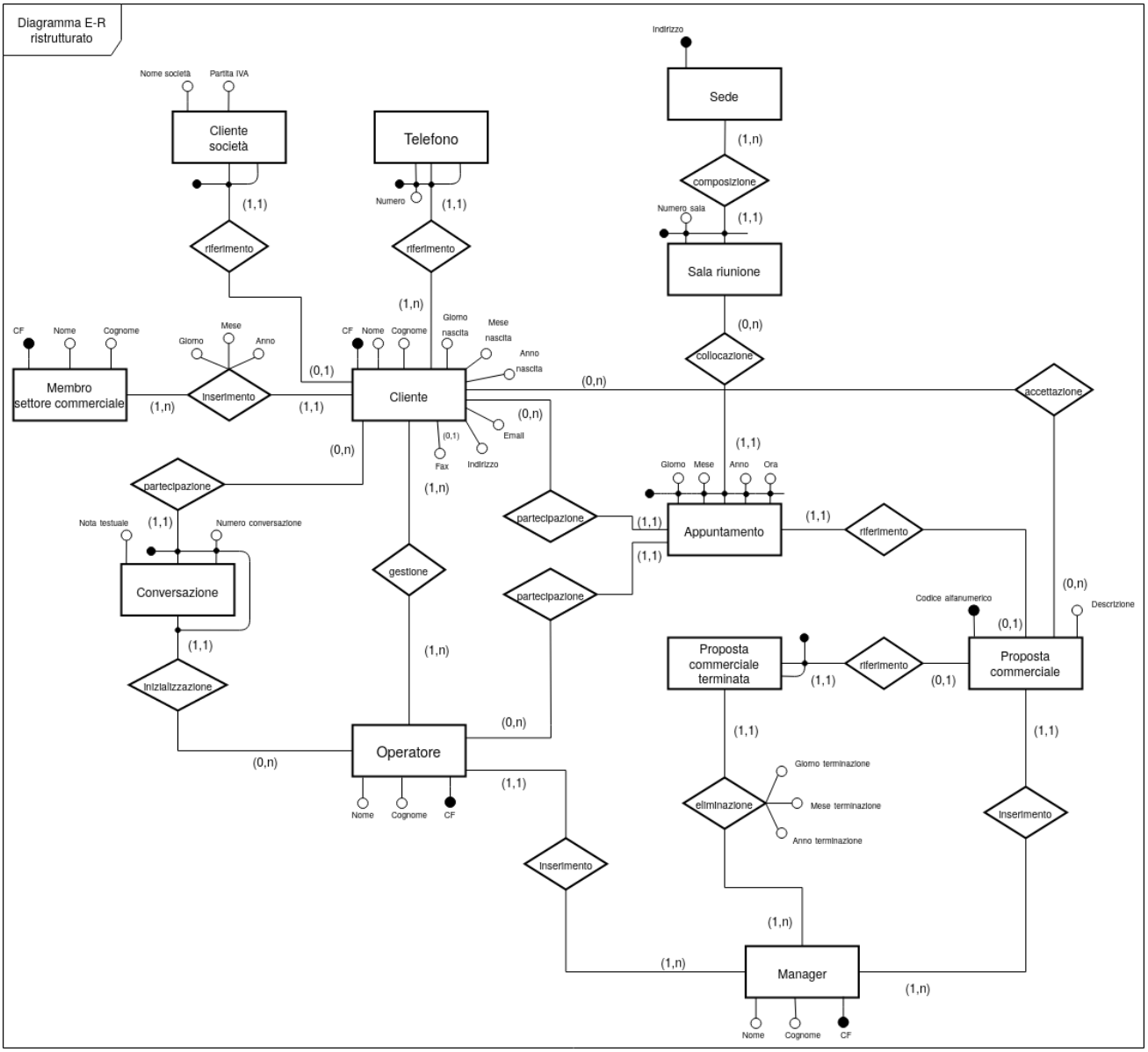
Op 6: (1 + 1)*4 000 = 8 000 accessi/giorno

Op 7: (1 + 2)*200 = 600 accessi/giorno

Op 8: (2+2+2)*1 000 = 6 000 accessi/giorno

Op 9: (2 + 2)*50 = 200 accessi/giorno

Ristrutturazione dello schema E-R



5

Le scelte di maggior impatto per la ristrutturazione dello schema E-R sono state:

- Conseguenza della frequenza dell'operazione 1 (visualizzazione del cliente), 100 000 volte al giorno, si è pensato di non aggiungere altri attributi all'entità Cliente come la data di registrazione e

il membro che ha inserito il cliente nel sistema. Si è pensato invece di delegare la responsabilità alla relazione ‘inserimento’ pena però qualche ridondanza.

- Conseguenza dello studio sul volume dei dati, in particolare del fatto che in media solo 1/5 dei clienti sono referenti di una società, si è pensato di eliminare la generalizzazione creando un entità debole ‘Cliente Società’ per Cliente, cosi da evitare valori null.

- Per la seconda generalizzazione è stato effettuato un diverso approccio. É stata eliminata l’entita ‘Proposta comerciale disponibile’ e collassata sulla piu generica ‘Proposta Commerciale’ cosi da evitare ridondaze nella base di dati. L’ entità ‘Proposta commerciale terminata’ è diventata entità debole di ‘Proposta Commerciale’.

- La scelta di identificatori primari è stata, il più delle volte, guidata dalla specifica.

Trasformazione di attributi e identificatori

L’ attributo composto Contatti della entità Cliente viene scoomposto in email, indirizzo, fax e una relazione verso l’entità Telefono.
Gli attributi multivalore Data vengono scomposti in giorno mese e anno.

Traduzione di entità e associazioni

CLIENTE(**CF**, Nome, Cognome, Giorno nascita, Mese nascita, Anno nascita, Email, Indirizzo, Fax)
CLIENTE SOCIETÀ(**Cliente**, Nome società, partita IVA)
TELEFONO(**Cliente**, **Numero**)
SEDE(**Indirizzo**)
SALA RIUNIONE(**Sede**, **Numero sala**)
MEMBRO SETTORE COMMERCIALE(**CF**, Nome, Cognome)
INSERIMENTO CLIENTE(**Cliente**, Membro sett. comm. , Giorno, Mese, Anno)
OPERATORE(**CF**,Nome,Cognome,Manager)
GESTIONE(**Operatore**, **Cliente**)
MANAGER(**CF**, Nome, Cognome)
PROPOSTA COMMERCIALE(**Codice alfanumerico**, Descrizione, Manager)

PROPOSTA COMMERCIALE TERMINATA(**Proposta comm**. , Giorno terminazione, Mese terminazione, Anno terminazione, Manager)

ACCETTAZIONE(**Cliente**, **Proposta comm**.)

CONVERSAZIONE(**Cliente**, **Operatore**, **Numero conversazione**, Nota testuale)

APPUNTAMENTO(**Sala**, **Sede**, **Giorno**, **Mese**, **Anno**, **Ora**, Cliente, Operatore, Proposta comm.)

Con vincoli di integrità referenziale:

CLIENTE SOCIETÀ(Cliente) ≤ **CLIENTE**(CF)

TELEFONO(Cliente) ≤ **CLIENTE**(CF)

SALA RIUNIONE(Sede) ≤ **SEDE**(Indirizzo)

INSERIMENTO CLIENTE(Cliente) ≤ **CLIENTE**(CF)

INSERIMENTO CLIENTE(Membro sett. comm.) ≤ **MEMBRO SETTORE COMMERCIALE**(CF)

OPERATORE(Manager) ≤ **MANAGER**(CF)

GESTIONE(Operatore) ≤ **OPERATORE**(CF)

GESTIONE(Cliente) ≤ **CLIENTE**(CF)

PROPOSTA COMMERCIALE(Manager) ≤ **MANAGER**(CF)

PROPOSTA COMMERCIALE TERMINATA(Proposta comm.) ≤ **PROPOSTA COMMERCIALE**(Codice alfanumerico)

PROPOSTA COMMERCIALE TERMINATA(Manager) ≤ **MANAGER**(CF)

ACCETTAZIONE(Cliente) ≤ **CLIENTE**(CF)

ACCETTAZIONE(Proposta comm.) ≤ **PROPOSTA COMMERCIALE**(Codice alfanumerico)

CONVERSAZIONE(Cliente) ≤ **CLIENTE**(CF)

CONVERSAZIONE(Operatore) ≤ **OPERATORE**(CF)

APPUNTAMENTO(Cliente) ≤ **CLIENTE**(CF)

APPUNTAMENTO(Operatore) ≤ **OPERATORE**(CF)

APPUNTAMENTO(Proposta comm.) ≤ **PROPOSTA COMMERCIALE**(Codice alfanumerico)

APPUNTAMENTO(Sala) ≤ **SALA RIUNIONE**(Numero sala)

APPUNTAMENTO(Sede) ≤ **SEDE**(Indirizzo)

Normalizzazione del modello relazionale

Forma 1NF:

Il modello è in 1NF infatti non ci sono attributi multivalore.

Forma 2NF:

Il modello è in 2NF infatti, prendendo in esame l'unico caso critico, la relazione CONVERSAZIONE ,dove la chiave è composta e sono presenti attributi non chiave, si può vedere che:

la nota testuale dipende completamente da ogni chiave, e non solamente da una parte di essa perchè ci possono essere più conversazioni tra la stessa coppia Cliente-Operatore, quindi in numero conversazione è necessario.

Forma 3NF:

Il modello è in 3NF perchè è in 2NF e per ogni relazione non esistono attributi di quella relazione che dipendono da altri attributi (entrambi non chiave).

5. Progettazione fisica

Utenti e privilegi

Gli utenti dell'applicazione sono: cliente, operatore, manager, membro settore commerciale e l'utente login.

Tutti gli utenti hanno privilegi di accesso ad alcune particolari routines.

Il cliente può eseguire:

- visualizza cliente
- visualizza numeri telefono
-visualizza proposte accettate

L' opernomeatore può eseguire:

- elimina nota conversazione
- inserisci appuntamento
- inserisci conversazione
- inserisci proposta accettata
- modifica nota conversazione
- elimina nota conversazione
- visualizza appuntamenti
- visualizza cliente
- visualizza elenco clienti
- visualizza numeri telefono
- visualizza proposte accettate
-visualizza conversazioni

Il manager può eseguire:

- archivia proposta
- inserisci operatore
- inserisci proposta

Il membro del settore commerciale può eseguire:

- inserisci cliente
- inserisci cliente societa

L' utente login può eseguire:

- login

Strutture di memorizzazione

Tabella <Accettazione>		
Attributo	**Tipo di dato**	**Attributi**[1]
Cliente	VARCHAR(16)	PK, NN
PropostaCommerciale	VARCHAR(45)	PK, NN

Tabella <Appuntamento>		
Attributo	**Tipo di dato**	**Attributi**
Sala	INT	PK, NN
Sede	VARCHAR(45)	PK, NN
Data	DATETIME	PK, NN
Cliente	VARCHAR(16)	NN
Operatore	VARCHAR(16)	NN
PropostaCommerciale	VARCHAR(45)	NN

5

Tabella <Cliente>		
Attributo	**Tipo di dato**	**Attributi**
Cf	VARCHAR(16)	PK, NN
Nome	VARCHAR(45)	NN
Cognome	VARCHAR(45)	NN
DataNascita	DATE	NN
Email	VARCHAR(45)	NN
Indirizzo	VARCHAR(45)	NN
Fax	VARCHAR(45)	

Tabella <ClienteSocieta>		
Attributo	**Tipo di dato**	**Attributi**
Cliente	VARCHAR(16)	PK, NN
NomeSocieta	VARCHAR(45)	NN
PartitaIVA	VARCHAR(45)	NN

Tabella <Conversazione>		
Attributo	**Tipo di dato**	**Attributi**
Cliente	VARCHAR(16)	PK, NN
Operatore	VARCHAR(16)	PK, NN

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

NumeroConversazione	INT	PK, NN
NotaTestuale	VARCHAR(100)	

Tabella <Gestione>		
Attributo	**Tipo di dato**	**Attributi**
Cliente	VARCHAR(16)	PK, NN
Operatore	VARCHAR(16)	PK, NN

Tabella <Inserimento>		
Attributo	**Tipo di dato**	**Attributi**
Cliente	VARCHAR(16)	PK, NN
MembroSettoreSociale	VARCHAR(16)	NN
DataInserimento	DATE	

Tabella <Manager>		
Attributo	**Tipo di dato**	**Attributi**
CF	VARCHAR(16)	PK, NN
Nome	VARCHAR(45)	NN
Cognome	VARCHAR(45)	NN

Tabella <MembroSettoreCommerciale>		
Attributo	**Tipo di dato**	**Attributi**
CF	VARCHAR(16)	PK, NN
Nome	VARCHAR(45)	NN
Cognome	VARCHAR(45)	NN

5

Tabella <Operatore>		
Attributo	**Tipo di dato**	**Attributi**
CF	VARCHAR(16)	PK, NN
Nome	VARCHAR(45)	NN
Cognome	VARCHAR(45)	NN
Manager	VARCHAR(16)	NN

Tabella <PropostaCommerciale>		
Attributo	**Tipo di dato**	**Attributi**
CodiceAlfanumerico	VARCHAR(45)	PK, NN
Descrizione	VARCHAR(100)	
Manager	VARCHAR(16)	NN

Tabella <PropostaCommercialeTerminata>		
Attributo	**Tipo di dato**	**Attributi**

propostaCommerciale	VARCHAR(16)	PK, NN
DataTerminazione	DATE	NN
Manager	VARCHAR(16)	NN

Tabella <Riferimento>		
Attributo	**Tipo di dato**	**Attributi**
Cliente	VARCHAR(16)	PK, NN
Operatore	VARCHAR(16)	PK, NN
NumeroConversazione	INT	PK, NN
PropostaCommerciale	VARCHAR(45)	PK, NN

Tabella <SalaRiunione>		
Attributo	**Tipo di dato**	**Attributi**
Sede	VARCHAR(45)	PK, NN
NumeroSala	INT	PK, NN

Tabella <Sede>		
Attributo	**Tipo di dato**	**Attributi**
Indirizzo	VARCHAR(45)	PK, NN

Tabella <Telefono>		
Attributo	**Tipo di dato**	**Attributi**
Cliente	VARCHAR(16)	PK, NN
Numero	INT	PK, NN

5

Tabella <Utente>		
Attributo	**Tipo di dato**	**Attributi**
Cf	VARCHAR(16)	PK, NN
Password	VARCHAR(45)	NN
Ruolo	ENUM(‘cliente’,‘operatore’,‘manager’,‘membo_settore_commerciale’)	NN

Indici

Tabella <Accettazione>	
Indice <nome>	**Tipo[2]:**
PRIMARY	PR
fk_Accettazione_1_idx (proposta commerciale)	IDX

Tabella <Appuntamento>

2 IDX = index, UQ = unique, FT = full text, PR = primary.

Indice <nome>	**Tipo:**
PRIMARY	PR
fk_Appuntamento_1_idx (cliente)	IDX
fk_Appuntamento_2_idx (operatore)	IDX
fk_Appuntamento_3_idx (sede)	IDX
fk_Appuntamento_4_idx (sala)	IDX
fk_Appuntamento_5_idx (propostaCommerciale)	IDX

Tabella <Cliente>	
Indice <nome>	**Tipo:**
PRIMARY	PR

Tabella <ClienteSocieta>	
Indice <nome>	**Tipo:**
PRIMARY	PR

Tabella <Conversazione>	
Indice <nome>	**Tipo:**
PRIMARY	PR
fk_Conversazione_1_idx (cliente)	IDX
fk_Conversazione_2_idx (operatore)	IDX
numero_conv_idx (numero)	IDX

Tabella <Gestione>	
Indice <nome>	**Tipo:**
PRIMARY	PR
fk_Gestione_1_idx (cliente)	IDX
fk_Gestione_2_idx (operatore)	IDX

5

Tabella <Inserimento>	
Indice <nome>	**Tipo:**
PRIMARY	PR
fk_Inserimento_1_idx (membro settore commerciale)	IDX

Tabella <Manager>	
Indice <nome>	**Tipo:**
PRIMARY	PR

Tabella <MembroSettoreCommerciale>

Indice <nome>	**Tipo:**
PRIMARY	PR

Tabella <Operatore>	
Indice <nome>	**Tipo:**
PRIMARY	PR
fk_Operatore_1_idx (manager)	IDX

Tabella <PropostaCommerciale>	
Indice <nome>	**Tipo:**
PRIMARY	PR
fk_PropostaCommerciale_1_idx (manager)	IDX

Tabella <PropostaCommercialeTerminata>	
Indice <nome>	**Tipo:**
PRIMARY	PR
fk_PropostaCommercialeTerminata_1_idx (manager)	IDX

Tabella <SalaRiunione>	
Indice <nome>	**Tipo:**
PRIMARY	PR
fk_SalaRiunione_1_idx (sede)	IDX

Tabella <Sede>	
Indice <nome>	**Tipo:**
PRIMARY	PR

Tabella <Telefono>	
Indice <nome>	**Tipo:**
PRIMARY	PR
fk_Telefono_1_idx (cliente)	IDX

Tabella <Utente>	
Indice <nome>	**Tipo:**
PRIMARY	PR

Trigger

Non è stato usato alcun trigger. Le regole aziendali sono state implementate attraverso controlli in alcune transazioni (vedi “inserisci_appuntamento” e “inserisci_conversazione”).

Eventi

Non sono stati usati eventi.

Viste

Non sono state usate viste.

Stored Procedures e transazioni

```
-- -------------------------------------------------
-- procedure archivia_proposta
-- -------------------------------------------------
DELIMITER $$
CREATE DEFINER=`root`@`localhost` PROCEDURE `archivia_proposta`(in codice_proposta_terminata VARCHAR(45), in codice_fiscale_manager VARCHAR(16))
BEGIN
	insert into `PropostaCommercialeTerminata`(`propostaCommerciale`, `dataTerminazione`, `manager`) values ( codice_proposta_terminata, curdate(), codice_fiscale_manager);
END$$


-- -------------------------------------------------
-- procedure create_user
-- -------------------------------------------------

DELIMITER $$
CREATE DEFINER=`root`@`localhost` PROCEDURE `create_user`(  in cf VARCHAR(16), in pass VARCHAR(45), in ruolo VARCHAR(45))
BEGIN
	insert into Utente VALUES(cf, MD5(pass), ruolo);
END$$

-- -------------------------------------------------
-- procedure elimina_nota_conversazione
-- -------------------------------------------------
```

```
DELIMITER $$
CREATE DEFINER=`root`@`localhost` PROCEDURE `elimina_nota_conversazione`(in codice_fiscale_cliente VARCHAR(45), in codice_fiscale_operatore VARCHAR(45), in num_conversazione int)
BEGIN
    UPDATE `Conversazione` set `notaTestuale` = '' where `cliente` = codice_fiscale_cliente and `operatore` = codice_fiscale_operatore and `numero` = num_conversazione;
END$$


-- -----------------------------------------------------
-- procedure inserisci_appuntamento
-- -----------------------------------------------------
DELIMITER $$
CREATE DEFINER=`root`@`localhost` PROCEDURE `inserisci_appuntamento`( in numero_sala int, in indirizzo_sede VARCHAR(45), in data_appuntamento datetime, in codice_fiscale_cliente VARCHAR(16), in codice_fiscale_operatore VARCHAR(16), in codice_proposta VARCHAR(45))
BEGIN
        declare client_manage int; #variabile per verificare se l'operatore ha
                                                #sotto gestione il cliente

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

    set transaction isolation level read committed; #evito letture sporche
        #read committed giusto compromesso
    #non mi interessa di letture non ripetibili (raramente questi dati vengono cambiati)

    start transaction;

        select count(*) into client_manage #check per vedere se l'operatore ha sotto gestione il cliente
```

```
                from `Gestione`
                where `cliente` = codice_fiscale_cliente and `operatore` = codice_fiscale_operatore;

    if client_manage = 0 then
                signal sqlstate '45002' set message_text = "Il cliente non è sotto gestione di questo operatore";
    else
                insert into `Appuntamento`(`sala`, `sede`, `data`, `cliente`, `operatore`, `propostaCommerciale`) values ( numero_sala, indirizzo_sede, data_appuntamento, codice_fiscale_cliente, codice_fiscale_operatore, codice_proposta);
            end if;
        commit;
END$$


-- -----------------------------------------------------
-- procedure inserisci_cliente
-- -----------------------------------------------------
DELIMITER $$
USE `crm_database`$$
CREATE DEFINER=`root`@`localhost` PROCEDURE `inserisci_cliente`(in codice_fiscale VARCHAR(16), in nome VARCHAR(45), in cognome VARCHAR(45), in email VARCHAR(45), in indirizzo VARCHAR(45), in fax VARCHAR(45), in data_nascita date, in numero_telefono int, in codice_fiscale_operatore VARCHAR(16), in codice_fiscale_member VARCHAR(16))
BEGIN
    declare var_num_cliente int;  #variabile per verificare la presenza del cliente

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

```
    set transaction isolation level read committed;
    #evito le letture dirty
    start transaction;
        #transazione per controllare prima se il cliente
    #che voglio aggiungere nel sistema è gia presente
    #se non e presente lo aggiungo al sistema
    select count(*)
        from `Inserimento`
    where `cliente` = codice_fiscale
    into var_num_cliente;

        if var_num_cliente > 0 then
            signal sqlstate '45001' set message_text = "The client is already registered in the CRM system";
        end if;
    #se il cliente non e gia registrato creo l'utente cliente e il rispettivo cliente con i dati passati
        call `create_user`( codice_fiscale, codice_fiscale, 'cliente');
        insert into `Cliente` (`cf`, `nome`, `cognome`, `dataNascita`, `email`, `indirizzo`, `fax`) values (codice_fiscale, nome, cognome, data_nascita, email, indirizzo, fax);

        insert into `Telefono` (`numero`, `cliente`) values (numero_telefono, codice_fiscale);

    insert into `Inserimento` values ( codice_fiscale_member, codice_fiscale, curdate());

    insert into `Gestione` values ( codice_fiscale_operatore, codice_fiscale);
        #imposto come username il codice fiscale --> potrei farlo cambiare all'utente insieme alla password


    commit;
END$$
```

```
-- -----------------------------------------------------
-- procedure inserisci_cliente_societa
-- -----------------------------------------------------

DELIMITER $$
USE `crm_database`$$
CREATE DEFINER=`root`@`localhost` PROCEDURE `inserisci_cliente_societa`(in codice_fiscale VARCHAR(16), in nome VARCHAR(45), in cognome VARCHAR(45), in email VARCHAR(45), in indirizzo VARCHAR(45), in fax VARCHAR(45), in data_nascita date, in numero_telefono int, in partitaIVA VARCHAR(45), in nome_societa VARCHAR(45), in codice_fiscale_operatore VARCHAR(16),in codice_fiscale_member VARCHAR(16))
BEGIN
        declare var_num_cliente int;  #variabile per verificare la presenza del cliente
    declare exit handler for sqlexception
    begin
        rollback;
        resignal;
    end;

    set transaction isolation level read committed;
    #evito le letture dirty
    start transaction;
        #transazione per controllare prima se il cliente
        #che voglio aggiungere nel sistema è gia presente
        #se non e presente lo aggiungo al sistema
        select count(*)
                from `Inserimento`
        where `cliente` = codice_fiscale
        into var_num_cliente;

                if var_num_cliente > 0 then
                        signal sqlstate '45001' set message_text = "The client is already registered in the CRM system";
                end if;
```

```
        #se il cliente non e gia registrato creo l'utente cliente e il rispettivo cliente con i dati passati
                call `create_user`( codice_fiscale, codice_fiscale, 'cliente');
                insert into `Cliente` (`cf`, `nome`, `cognome`, `dataNascita`, `email`, `indirizzo`, `fax`) values (codice_fiscale, nome, cognome, data_nascita, email, indirizzo, fax);

                insert into `Telefono` (`numero`, `cliente`) values (numero_telefono, codice_fiscale);

        insert into `Inserimento` values ( codice_fiscale_member, codice_fiscale, curdate());

        insert into `Gestione` values ( codice_fiscale_operatore, codice_fiscale);

            insert into `ClienteSocieta` (`cliente`,`nomeSocieta`,`partitaIVA`) values (codice_fiscale, nome_societa, partitaIVA);
            commit;
    END$$

    -- -----------------------------------------------------
    -- procedure inserisci_conversazione
    -- -----------------------------------------------------

    DELIMITER $$
    USE `crm_database`$$
    CREATE DEFINER=`root`@`localhost` PROCEDURE `inserisci_conversazione`(in codice_fiscale_cliente VARCHAR(16), in codice_fiscale_operatore VARCHAR(16), in nota_testuale VARCHAR(100))
    BEGIN
            declare client_manage int;
      declare num int;
      declare num_count int;
            declare exit handler for sqlexception
      begin
        rollback;
        resignal;
      end;
```

```
set transaction isolation level read committed;

start transaction;
        #transazione usata per effettuare un doppio controllo
    #primo: controllare se il cliente è sotto gestione dell'operatore
    #secondo: controllare se è la prima conversazione tra i due   ---> se si num_conv lo imposto a 1
    select count(*) into client_manage
                from `Gestione`
                where `cliente` = codice_fiscale_cliente and `operatore` = codice_fiscale_operatore;

    if client_manage = 0 then
                signal sqlstate '45002' set message_text = "Il cliente non è sotto gestione di questo operatore";
    else
                select count(*) into num_count
                        from `Conversazione`
            where `cliente` = codice_fiscale_cliente and `operatore` = codice_fiscale_operatore;
        if num_count = 0 then
                        #default value 1
                        set num = 1;
                else
                        #increase the num conversation between client and operator
                        select max(numero) + 1 into num
                                from `Conversazione`
                                where `cliente` = codice_fiscale_cliente and `operatore` = codice_fiscale_operatore;
                end if;
                insert into `Conversazione` (`numero`, `cliente`, `operatore`, `notaTestuale`) values ( num, codice_fiscale_cliente, codice_fiscale_operatore, nota_testuale);
    end if;
        commit;
END$$
```

```
-- -----------------------------------------------------
-- procedure inserisci_gestione_cliente
-- -----------------------------------------------------

DELIMITER $$
USE `crm_database`$$
CREATE DEFINER=`root`@`localhost` PROCEDURE `inserisci_gestione_cliente`(in
codice_fiscale_cliente VARCHAR(16), in codice_fiscale_operatore VARCHAR(16) )
BEGIN
	insert into `Gestione` values ( codice_fiscale_operatore, codice_fiscale_cliente);
END$$

-- -----------------------------------------------------
-- procedure inserisci_operatore
-- -----------------------------------------------------

DELIMITER $$
USE `crm_database`$$
CREATE DEFINER=`root`@`localhost` PROCEDURE `inserisci_operatore`(in codice_fiscale
VARCHAR(16), in nome VARCHAR(45), in cognome VARCHAR(45), in codice_fiscale_manager
VARCHAR(16))
BEGIN
	declare exit handler for sqlexception
  begin
    rollback;
    resignal;
  end;

	set transaction isolation level read committed;
  start transaction; #transazione per rispettare vincoli di foreign key
		call `create_user`(codice_fiscale, codice_fiscale,'operatore');
		insert into `Operatore` (`cf`, `nome`, `cognome`, `manager`) values (codice_fiscale,
```

```
nome, cognome, codice_fiscale_manager);

   commit;
END$$

-- -----------------------------------------------------
-- procedure inserisci_proposta
-- -----------------------------------------------------

DELIMITER $$
USE `crm_database`$$
CREATE DEFINER=`root`@`localhost` PROCEDURE `inserisci_proposta`(in codice VARCHAR(45), in descrizione VARCHAR(100), in codice_fiscale_manager VARCHAR(16))
BEGIN
        insert into `PropostaCommerciale` values (codice, descrizione, codice_fiscale_manager);

END$$

-- -----------------------------------------------------
-- procedure inserisci_proposta_accettata
-- -----------------------------------------------------

DELIMITER $$
USE `crm_database`$$
CREATE DEFINER=`root`@`localhost` PROCEDURE `inserisci_proposta_accettata`( in codice_fiscale_cliente VARCHAR(16), in codice_proposta VARCHAR(45))
BEGIN
        insert into `Accettazione` (`cliente`, `propostaCommerciale`) values ( codice_fiscale_cliente, codice_proposta);
END$$


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

```
-- procedure login
-- -----------------------------------------------------

DELIMITER $$
USE `crm_database`$$
CREATE DEFINER=`root`@`localhost` PROCEDURE `login`(in cf varchar(16), in pass varchar(45), out var_role INT)
BEGIN
	declare var_user_role ENUM('cliente', 'operatore', 'manager', 'membro_settore_commerciale');



	select u.`ruolo` into var_user_role
		from `Utente` as u
		where u.`cf` = cf
	and u.`password` = md5(pass);


	-- See the corresponding enum in the client
		if var_user_role = 'cliente' then
			set var_role = 1;
		elseif var_user_role = 'operatore' then
			set var_role = 2;
		elseif var_user_role = 'manager' then
			set var_role = 3;
		elseif var_user_role = 'membro_settore_commerciale' then
			set var_role = 4;
		else
			set var_role = 5;
		end if;
END$$
```

```
-- -----------------------------------------------------
-- procedure modifica_nota_conversazione
-- -----------------------------------------------------

DELIMITER $$
USE `crm_database`$$
CREATE DEFINER=`root`@`localhost` PROCEDURE `modifica_nota_conversazione`(in codice_fiscale_cliente VARCHAR(45), in codice_fiscale_operatore VARCHAR(45), in num_conversazione int, in nota_testuale_aggiornata VARCHAR(100))
BEGIN
	UPDATE `Conversazione` set `notaTestuale` = nota_testuale_aggiornata where `cliente` = codice_fiscale_cliente and `operatore` = codice_fiscale_operatore and `numero` = num_conversazione;
END$$


-- -----------------------------------------------------
-- procedure visualizza_appuntamenti
-- -----------------------------------------------------

DELIMITER $$
USE `crm_database`$$
CREATE DEFINER=`root`@`localhost` PROCEDURE `visualizza_appuntamenti`( in codice_fiscale_operatore VARCHAR(16))
BEGIN

	select `C`.`cf`, `C`.`nome`, `C`.`cognome`, `data`, `sede`, `sala`, `codiceAlfanumerico`, `descrizione`
		from `Appuntamento` join `Cliente` as `C` on `cliente` = `C`.`cf`
			join `PropostaCommerciale` as `P` on `propostaCommerciale` = `P`.`codiceAlfanumerico`
		where `operatore` = codice_fiscale_operatore;

END$$
```

```
-- -----------------------------------------------------
-- procedure visualizza_cliente
-- -----------------------------------------------------

DELIMITER $$
USE `crm_database`$$
CREATE DEFINER=`root`@`localhost` PROCEDURE `visualizza_cliente`(in
codice_fiscale_cliente VARCHAR(16))
BEGIN
   declare var_count int;

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

        set transaction read only;
   set transaction isolation level repeatable read;
            #voglio evitare le letture dirty e le letture non ripetibili
      # se cambia un dato voglio essere aggiornato
            select count(*) into var_count
            from `ClienteSsocieta`
            where `cliente` = codice_fiscale_cliente;

            if var_count > 0 then
                  #se ci sono entry nella tabella ClienteSsocieta il cliente è un referente di una
società
        #nella select prendo anche partita iva e nome societa
          select cliente.`cf`, cliente.`nome`, cliente.`cognome`, cliente.`email` , cliente.`dataNascita`,
cliente.`email`, cliente.`indirizzo`, cliente.`fax`, c.`partitaIVA`, c.`nomeSocieta`
                  from `Cliente` as cliente
```

```
                join `ClienteSocieta` as c on cliente.`cf` = c.`cliente`
            where cliente.`cf` = codice_fiscale_cliente;
        elseif var_count = 0 then
            #se non ci sono entry il cliente passato è un semplice cliente
            select cliente.`cf`, cliente.`nome`, cliente.`cognome`, cliente.`email` , cliente.`dataNascita`, cliente.`email`, cliente.`indirizzo`, cliente.`fax`
            from `Cliente` as cliente
            where cliente.`cf` = codice_fiscale_cliente;
        end if;
    commit;

END$$

-- -----------------------------------------------------
-- procedure visualizza_conversazioni
-- -----------------------------------------------------

DELIMITER $$
USE `crm_database`$$
CREATE DEFINER=`root`@`localhost` PROCEDURE `visualizza_conversazioni`(in codice_fiscale_operatore VARCHAR(16))
BEGIN

    select cl.`cf`, cl.`nome`, cl.`cognome`, co.`numero`, co.`notaTestuale`
        from `Conversazione` as co
            join `Cliente` as cl on co.`cliente` = cl.`cf`
        where co.`operatore` = codice_fiscale_operatore;

END$$


-- -----------------------------------------------------
-- procedure visualizza_elenco_clienti
-- -----------------------------------------------------
```

```
DELIMITER $$
USE `crm_database`$$
CREATE DEFINER=`root`@`localhost` PROCEDURE `visualizza_elenco_clienti`( in
codice_fiscale_operatore VARCHAR(16))
BEGIN

    select c.`cf`, c.`nome`, c.`cognome`
        from `Gestione` as g join `Cliente` as c on g.`cliente` = c.`cf`
        where g.`operatore` = codice_fiscale_operatore;

END$$

-- -----------------------------------------------------
-- procedure visualizza_numeri_telefono
-- -----------------------------------------------------

DELIMITER $$
USE `crm_database`$$
CREATE DEFINER=`root`@`localhost` PROCEDURE `visualizza_numeri_telefono`(in
codice_fiscale VARCHAR(16))
BEGIN

    select `numero`
        from `Telefono`
        where `cliente` = codice_fiscale;

END$$


-- -----------------------------------------------------
-- procedure visualizza_proposte_accettate
-- -----------------------------------------------------
```

```
DELIMITER $$
USE `crm_database`$$
CREATE DEFINER=`root`@`localhost` PROCEDURE `visualizza_proposte_accettate`(in codice_fiscale VARCHAR(16))
BEGIN

  select p.`codiceAlfanumerico` , p.`descrizione`
        from `Accettazione` as a
            join `PropostaCommerciale` as p on a.`propostaCommerciale` = p.`codiceAlfanumerico`
        where a.`cliente` = codice_fiscale;

END$$
```

Appendice: Implementazione

Codice SQL per instanziare il database

-- MySQL Workbench Forward Engineering

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

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

-- -----------------------------------------------------
-- Schema crm_database
-- -----------------------------------------------------
CREATE SCHEMA IF NOT EXISTS `crm_database` DEFAULT CHARACTER SET utf8mb4 COLLATE utf8mb4_0900_ai_ci ;
USE `crm_database` ;

-- -----------------------------------------------------
-- Table `crm_database`.`Utente`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `crm_database`.`Utente` ;

CREATE TABLE IF NOT EXISTS `crm_database`.`Utente` (
  `cf` VARCHAR(16) NOT NULL,
  `password` VARCHAR(45) NOT NULL,
```

```
  `ruolo` ENUM('cliente', 'operatore', 'manager', 'membro_settore_commerciale') NOT NULL,
  PRIMARY KEY (`cf`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8;


-- -----------------------------------------------------
-- Table `crm_database`.`Cliente`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `crm_database`.`Cliente` ;

CREATE TABLE IF NOT EXISTS `crm_database`.`Cliente` (
  `cf` VARCHAR(16) NOT NULL,
  `nome` VARCHAR(45) NOT NULL,
  `cognome` VARCHAR(45) NOT NULL,
  `dataNascita` DATE NOT NULL,
  `email` VARCHAR(45) NOT NULL,
  `indirizzo` VARCHAR(45) NOT NULL,
  `fax` VARCHAR(45) NULL DEFAULT NULL,
  PRIMARY KEY (`cf`),
  CONSTRAINT `fk_Cliente_1`
    FOREIGN KEY (`cf`)
    REFERENCES `crm_database`.`Utente` (`cf`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8;



-- -----------------------------------------------------
-- Table `crm_database`.`Manager`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `crm_database`.`Manager` ;

CREATE TABLE IF NOT EXISTS `crm_database`.`Manager` (
  `cf` VARCHAR(16) NOT NULL,
```

```
  `nome` VARCHAR(45) NOT NULL,
  `cognome` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`cf`),
  CONSTRAINT `fk_Manager_1`
    FOREIGN KEY (`cf`)
    REFERENCES `crm_database`.`Utente` (`cf`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8;


-- -----------------------------------------------------
-- Table `crm_database`.`PropostaCommerciale`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `crm_database`.`PropostaCommerciale` ;

CREATE TABLE IF NOT EXISTS `crm_database`.`PropostaCommerciale` (
  `codiceAlfanumerico` VARCHAR(45) NOT NULL,
  `descrizione` VARCHAR(100) NULL DEFAULT NULL,
  `manager` VARCHAR(16) NOT NULL,
  PRIMARY KEY (`codiceAlfanumerico`),
  INDEX `fk_PropostaCommerciale_1_idx` (`manager` ASC) VISIBLE,
  CONSTRAINT `fk_PropostaCommerciale_1`
    FOREIGN KEY (`manager`)
    REFERENCES `crm_database`.`Manager` (`cf`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8;


-- -----------------------------------------------------
-- Table `crm_database`.`Accettazione`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `crm_database`.`Accettazione` ;

CREATE TABLE IF NOT EXISTS `crm_database`.`Accettazione` (
```

```
  `cliente` VARCHAR(16) NOT NULL,
  `propostaCommerciale` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`cliente`, `propostaCommerciale`),
  INDEX `fk_Accettazione_2_idx` (`propostaCommerciale` ASC) VISIBLE,
  CONSTRAINT `fk_Accettazione_1`
    FOREIGN KEY (`cliente`)
    REFERENCES `crm_database`.`Cliente` (`cf`),
  CONSTRAINT `fk_Accettazione_2`
    FOREIGN KEY (`propostaCommerciale`)
    REFERENCES `crm_database`.`PropostaCommerciale` (`codiceAlfanumerico`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8;


-- -----------------------------------------------------
-- Table `crm_database`.`Operatore`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `crm_database`.`Operatore` ;

CREATE TABLE IF NOT EXISTS `crm_database`.`Operatore` (
  `cf` VARCHAR(16) NOT NULL,
  `nome` VARCHAR(45) NOT NULL,
  `cognome` VARCHAR(45) NOT NULL,
  `manager` VARCHAR(16) NOT NULL,
  PRIMARY KEY (`cf`),
  INDEX `fk_Operatore_1_idx` (`manager` ASC) VISIBLE,
  CONSTRAINT `fk_Operatore_1`
    FOREIGN KEY (`manager`)
    REFERENCES `crm_database`.`Manager` (`cf`),
  CONSTRAINT `fk_Operatore_2`
    FOREIGN KEY (`cf`)
    REFERENCES `crm_database`.`Utente` (`cf`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8;
```

```
-- -----------------------------------------------------
-- Table `crm_database`.`Sede`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `crm_database`.`Sede` ;

CREATE TABLE IF NOT EXISTS `crm_database`.`Sede` (
  `indirizzo` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`indirizzo`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8;


-- -----------------------------------------------------
-- Table `crm_database`.`SalaRiunione`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `crm_database`.`SalaRiunione` ;

CREATE TABLE IF NOT EXISTS `crm_database`.`SalaRiunione` (
  `numeroSala` INT NOT NULL,
  `sede` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`numeroSala`, `sede`),
  INDEX `fk_SalaRiunione_1_idx` (`sede` ASC) VISIBLE,
  INDEX `index3` (`numeroSala` ASC) VISIBLE,
  CONSTRAINT `fk_SalaRiunione_1`
    FOREIGN KEY (`sede`)
    REFERENCES `crm_database`.`Sede` (`indirizzo`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8;


-- -----------------------------------------------------
-- Table `crm_database`.`Appuntamento`
```

```
-- -----------------------------------------------------
DROP TABLE IF EXISTS `crm_database`.`Appuntamento` ;

CREATE TABLE IF NOT EXISTS `crm_database`.`Appuntamento` (
  `sala` INT NOT NULL,
  `sede` VARCHAR(45) NOT NULL,
  `data` DATETIME NOT NULL,
  `cliente` VARCHAR(16) NOT NULL,
  `operatore` VARCHAR(16) NOT NULL,
  `propostaCommerciale` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`sala`, `sede`, `data`),
  INDEX `fk_Appuntamento_1_idx` (`cliente` ASC) VISIBLE,
  INDEX `fk_Appuntamento_2_idx` (`operatore` ASC) VISIBLE,
  INDEX `fk_Appuntamento_3_idx` (`sede` ASC) VISIBLE,
  INDEX `fk_Appuntamento_5_idx` (`propostaCommerciale` ASC) VISIBLE,
  CONSTRAINT `fk_Appuntamento_1`
    FOREIGN KEY (`cliente`)
    REFERENCES `crm_database`.`Cliente` (`cf`),
  CONSTRAINT `fk_Appuntamento_2`
    FOREIGN KEY (`operatore`)
    REFERENCES `crm_database`.`Operatore` (`cf`),
  CONSTRAINT `fk_Appuntamento_3`
    FOREIGN KEY (`sede`)
    REFERENCES `crm_database`.`Sede` (`indirizzo`),
  CONSTRAINT `fk_Appuntamento_4`
    FOREIGN KEY (`sala`)
    REFERENCES `crm_database`.`SalaRiunione` (`numeroSala`),
  CONSTRAINT `fk_Appuntamento_5`
    FOREIGN KEY (`propostaCommerciale`)
    REFERENCES `crm_database`.`PropostaCommerciale` (`codiceAlfanumerico`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8;
```

```
-- -----------------------------------------------------
-- Table `crm_database`.`ClienteSocieta`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `crm_database`.`ClienteSocieta` ;

CREATE TABLE IF NOT EXISTS `crm_database`.`ClienteSocieta` (
  `cliente` VARCHAR(16) NOT NULL,
  `nomeSocieta` VARCHAR(45) NOT NULL,
  `partitaIVA` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`cliente`),
  CONSTRAINT `fk_ClienteSocieta_1`
    FOREIGN KEY (`cliente`)
    REFERENCES `crm_database`.`Cliente` (`cf`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8;


-- -----------------------------------------------------
-- Table `crm_database`.`Conversazione`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `crm_database`.`Conversazione` ;

CREATE TABLE IF NOT EXISTS `crm_database`.`Conversazione` (
  `numero` INT NOT NULL,
  `operatore` VARCHAR(16) NOT NULL,
  `notaTestuale` VARCHAR(100) NULL DEFAULT NULL,
  `cliente` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`numero`, `operatore`, `cliente`),
  INDEX `fk_Conversazione_2_idx` (`operatore` ASC) VISIBLE,
  INDEX `fk_Conversazione_1_idx` (`cliente` ASC) VISIBLE,
  INDEX `numero_conv_idx` (`numero` ASC) VISIBLE,
  CONSTRAINT `fk_Conversazione_1`
    FOREIGN KEY (`cliente`)
    REFERENCES `crm_database`.`Cliente` (`cf`)
```

```
    ON DELETE RESTRICT
    ON UPDATE RESTRICT,
  CONSTRAINT `fk_Conversazione_2`
    FOREIGN KEY (`operatore`)
    REFERENCES `crm_database`.`Operatore` (`cf`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8;


-- -----------------------------------------------------
-- Table `crm_database`.`Gestione`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `crm_database`.`Gestione` ;

CREATE TABLE IF NOT EXISTS `crm_database`.`Gestione` (
  `operatore` VARCHAR(16) NOT NULL,
  `cliente` VARCHAR(16) NOT NULL,
  PRIMARY KEY (`operatore`, `cliente`),
  INDEX `fk_Gestione_1_idx` (`cliente` ASC) VISIBLE,
  CONSTRAINT `fk_Gestione_1`
    FOREIGN KEY (`cliente`)
    REFERENCES `crm_database`.`Cliente` (`cf`),
  CONSTRAINT `fk_Gestione_2`
    FOREIGN KEY (`operatore`)
    REFERENCES `crm_database`.`Operatore` (`cf`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8;


-- -----------------------------------------------------
-- Table `crm_database`.`MembroSettoreCommerciale`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `crm_database`.`MembroSettoreCommerciale` ;
```

```
CREATE TABLE IF NOT EXISTS `crm_database`.`MembroSettoreCommerciale` (
  `cf` VARCHAR(16) NOT NULL,
  `nome` VARCHAR(45) NOT NULL,
  `cognome` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`cf`),
  CONSTRAINT `fk_MembroSettoreCommerciale_1`
    FOREIGN KEY (`cf`)
    REFERENCES `crm_database`.`Utente` (`cf`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8;


-- -----------------------------------------------------
-- Table `crm_database`.`Inserimento`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `crm_database`.`Inserimento` ;

CREATE TABLE IF NOT EXISTS `crm_database`.`Inserimento` (
  `membroSettoreCommerciale` VARCHAR(16) NOT NULL,
  `cliente` VARCHAR(16) NOT NULL,
  `dataInserimento` DATE NULL DEFAULT NULL,
  PRIMARY KEY (`cliente`),
  INDEX `fk_Inserimento_2_idx` (`membroSettoreCommerciale` ASC) VISIBLE,
  CONSTRAINT `fk_Inserimento_1`
    FOREIGN KEY (`cliente`)
    REFERENCES `crm_database`.`Cliente` (`cf`),
  CONSTRAINT `fk_Inserimento_2`
    FOREIGN KEY (`membroSettoreCommerciale`)
    REFERENCES `crm_database`.`MembroSettoreCommerciale` (`cf`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8;


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

```
-- Table `crm_database`.`PropostaCommercialeTerminata`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `crm_database`.`PropostaCommercialeTerminata` ;

CREATE TABLE IF NOT EXISTS `crm_database`.`PropostaCommercialeTerminata` (
  `propostaCommerciale` VARCHAR(45) NOT NULL,
  `dataTerminazione` DATE NOT NULL,
  `manager` VARCHAR(16) NOT NULL,
  PRIMARY KEY (`propostaCommerciale`),
  INDEX `fk_PropostaCommercialeTerminata_2_idx` (`manager` ASC) VISIBLE,
  CONSTRAINT `fk_PropostaCommercialeTerminata_1`
    FOREIGN KEY (`propostaCommerciale`)
    REFERENCES `crm_database`.`PropostaCommerciale` (`codiceAlfanumerico`),
  CONSTRAINT `fk_PropostaCommercialeTerminata_2`
    FOREIGN KEY (`manager`)
    REFERENCES `crm_database`.`Manager` (`cf`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8;


-- -----------------------------------------------------
-- Table `crm_database`.`Telefono`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `crm_database`.`Telefono` ;

CREATE TABLE IF NOT EXISTS `crm_database`.`Telefono` (
  `numero` INT NOT NULL,
  `cliente` VARCHAR(16) NOT NULL,
  PRIMARY KEY (`numero`, `cliente`),
  INDEX `fk_Telefono_1_idx` (`cliente` ASC) VISIBLE,
  CONSTRAINT `fk_Telefono_1`
    FOREIGN KEY (`cliente`)
    REFERENCES `crm_database`.`Cliente` (`cf`))
ENGINE = InnoDB
```

```
DEFAULT CHARACTER SET = utf8;

USE `crm_database` ;

GRANT EXECUTE ON procedure `crm_database`.`visualizza_cliente` TO 'cliente';
GRANT EXECUTE ON procedure `crm_database`.`visualizza_numeri_telefono` TO 'cliente';
GRANT EXECUTE ON procedure `crm_database`.`visualizza_proposte_accettate` TO 'cliente';
GRANT EXECUTE ON procedure `crm_database`.`elimina_nota_conversazione` TO 'operatore';
GRANT EXECUTE ON procedure `crm_database`.`modifica_nota_conversazione` TO 'operatore';
GRANT EXECUTE ON procedure `crm_database`.`inserisci_conversazione` TO 'operatore';
GRANT EXECUTE ON procedure `crm_database`.`inserisci_appuntamento` TO 'operatore';
GRANT EXECUTE ON procedure `crm_database`.`inserisci_gestione_cliente` TO 'operatore';
GRANT EXECUTE ON procedure `crm_database`.`inserisci_proposta_accettata` TO 'operatore';
GRANT EXECUTE ON procedure `crm_database`.`visualizza_appuntamenti` TO 'operatore';
GRANT EXECUTE ON procedure `crm_database`.`visualizza_cliente` TO 'operatore';
GRANT EXECUTE ON procedure `crm_database`.`visualizza_conversazioni` TO 'operatore';
GRANT EXECUTE ON procedure `crm_database`.`visualizza_elenco_clienti` TO 'operatore';
GRANT EXECUTE ON procedure `crm_database`.`visualizza_numeri_telefono` TO 'operatore';
GRANT EXECUTE ON procedure `crm_database`.`visualizza_proposte_accettate` TO 'operatore';
GRANT EXECUTE ON procedure `crm_database`.`archivia_proposta` TO 'manager';
GRANT EXECUTE ON procedure `crm_database`.`create_user` TO 'manager';
GRANT EXECUTE ON procedure `crm_database`.`inserisci_operatore` TO 'manager';
GRANT EXECUTE ON procedure `crm_database`.`inserisci_proposta` TO 'manager';
GRANT EXECUTE ON procedure `crm_database`.`create_user` TO
'membro_settore_commerciale';
GRANT EXECUTE ON procedure `crm_database`.`inserisci_cliente` TO
'membro_settore_commerciale';
GRANT EXECUTE ON procedure `crm_database`.`inserisci_cliente_societa` TO
'membro_settore_commerciale';
GRANT EXECUTE ON procedure `crm_database`.`login` TO 'login';

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

Codice del Front-End

```
/*
 *
 ************** MAIN ******************
 *
 */

#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <string.h>
#include <unistd.h>

#include "defines.h"

typedef enum {
        CLIENT = 1,
        OPERATOR,
        MANAGER,
        COMMERCIAL_SECTOR_MEMBER,
        FAILED_LOGIN
} role_t;

struct configuration    conf;           //struct for config of db
char                                    cf[16]; //codice fiscale of the running user
static MYSQL                            *conn;          //struct fot connection to db


static role_t attempt_login(MYSQL *conn, char *cf, char *password) {
        MYSQL_STMT *login_procedure;
```

```
MYSQL_BIND param[3];                    //Used both for input and output
int role = 0;
// Prepare the statement avoid injection
if(!setup_prepared_stmt(&login_procedure, "call login(?, ?, ?)", conn)) {
        print_stmt_error(login_procedure, "Unable to initialize login statement\n");
        goto err2;
}

// Prepare parameters
memset(param, 0, sizeof(param));

param[0].buffer_type = MYSQL_TYPE_VAR_STRING;    // IN
param[0].buffer = cf;
param[0].buffer_length = strlen(cf);

param[1].buffer_type = MYSQL_TYPE_VAR_STRING;    // IN
param[1].buffer = password;
param[1].buffer_length = strlen(password);

param[2].buffer_type = MYSQL_TYPE_LONG;          // OUT
param[2].buffer = &role;
param[2].buffer_length = sizeof(role);


// Bind the parameters with the procedure
if (mysql_stmt_bind_param(login_procedure, param) != 0) { // Note _param
        print_stmt_error(login_procedure, "Could not bind parameters for login");
        goto err;
}

// Run procedure
if (mysql_stmt_execute(login_procedure) != 0) {
        print_stmt_error(login_procedure, "Could not execute login procedure");
```

```
            goto err;
        }

        // Prepare output parameters
5       memset(param, 0, sizeof(param));
        param[0].buffer_type = MYSQL_TYPE_LONG; // OUT
        param[0].buffer = &role;
        param[0].buffer_length = sizeof(role);

10

        if(mysql_stmt_bind_result(login_procedure, param)) {
            print_stmt_error(login_procedure, "Could not retrieve output parameter");
            goto err;
        }
15
        // Retrieve output parameter
        if(mysql_stmt_fetch(login_procedure)) {
            print_stmt_error(login_procedure, "Could not buffer results");
            goto err;
20      }


        mysql_stmt_close(login_procedure);

25      return role;

    err:
        mysql_stmt_close(login_procedure);

30  err2:
        return FAILED_LOGIN;
}

int main(void)
```

```
{
        role_t role;
        // Parse the .json file for take info for db config
        if(!parse_config("users/login.json", &conf)) {
                fprintf(stderr, "Unable to load login configuration\n");
                exit(EXIT_FAILURE);
        }

        conn = mysql_init (NULL);
        if (conn == NULL) {
                fprintf (stderr, "mysql_init() failed \n");
                exit(EXIT_FAILURE);
        }
        // Connect with config parameters to the db
        if (mysql_real_connect(conn, conf.host, conf.db_username, conf.db_password, conf.database,
conf.port, NULL, CLIENT_MULTI_STATEMENTS | CLIENT_MULTI_RESULTS) == NULL) {
                fprintf (stderr, "mysql_real_connect() failed\n");
                mysql_close (conn);
                exit(EXIT_FAILURE);
        }
        // Take login data from input
        printf("Codice fiscale: ");
        getInput(16, cf, false);
        printf("Password: ");
        getInput(45, conf.password, true);
        // Switch through the role
        role = attempt_login(conn, cf, conf.password);
        switch(role) {
                case CLIENT:
                        start_client_view(conn);
                        break;

                case OPERATOR:
                        start_operator_view(conn);
```

```
                    break;

                case MANAGER:
                    start_manager_view(conn);
                    break;

                case COMMERCIAL_SECTOR_MEMBER:
                    start_member_view(conn);
                    break;

                case FAILED_LOGIN:
                    fprintf(stderr, "Invalid credentials\n");
                    exit(EXIT_FAILURE);
                    break;

                default:
                    fprintf(stderr, "Invalid condition at %s:%d\n", __FILE__, __LINE__);
                    abort();
            }
            printf("Exiting from the app...\n");

            // Close the db connection
            mysql_close (conn);

            return 0;

        }
```

```
/*****************************************************************
 *
 ************** CLIENT MAIN ******************
 *
 */

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

#include "defines.h"



void show_phone_numbers(){
        MYSQL *conn;
        conn = connection_db();
        MYSQL_STMT *prepared_stmt;
        MYSQL_BIND param[1];

        // Prepare stored procedure call
        if(!setup_prepared_stmt(&prepared_stmt, "call visualizza_numeri_telefono(?)", conn)) {
                finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize phone number
statement\n", false);
        }

        // Prepare parameters
        memset(param, 0, sizeof(param));

        param[0].buffer_type = MYSQL_TYPE_VAR_STRING; // IN
        param[0].buffer = cf;  //use cf of the current user
        param[0].buffer_length = strlen(cf);
```

```
        // Bind parameters and store procedure
        if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
                finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters for phone number\n", true);
        }

        // Run procedure
        if (mysql_stmt_execute(prepared_stmt) != 0) {
                print_stmt_error(prepared_stmt, "An error occurred while retrieving the phone number.");
                goto out;
        }

        // Dump the result set
        dump_result_set(conn, prepared_stmt, "\nPhone numbers");

        out:
        mysql_stmt_close(prepared_stmt);
}

void show_accepted_proposals(){
        MYSQL *conn;
        conn = connection_db();
        MYSQL_STMT *prepared_stmt;
        MYSQL_BIND param[1];


        // Prepare stored procedure call
        if(!setup_prepared_stmt(&prepared_stmt, "call visualizza_proposte_accettate(?)", conn)) {
                finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize proposal statement\n", false);
        }

        // Prepare parameters
```

```
        memset(param, 0, sizeof(param));

        param[0].buffer_type = MYSQL_TYPE_VAR_STRING;    //IN
        param[0].buffer = cf;
        param[0].buffer_length = strlen(cf);

        if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
                finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters for accepted proposals\n", true);
        }

        // Run procedure
        if (mysql_stmt_execute(prepared_stmt) != 0) {
                print_stmt_error(prepared_stmt, "An error occurred while retrieving the accepted proposals.");
                goto out;
        }

        // Dump the result set
        dump_result_set(conn, prepared_stmt, "\nAccepted proposals");

        out:
        mysql_stmt_close(prepared_stmt);

}

void show_client(){
        MYSQL *conn;
        conn = connection_db();
        MYSQL_STMT *prepared_stmt;
        MYSQL_BIND param[1];
```

```
        // Prepare stored procedure call
        if(!setup_prepared_stmt(&prepared_stmt, "call visualizza_cliente(?)", conn)) {
                finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize client statement\n",
5   false);
        }

        // Prepare parameters
        memset(param, 0, sizeof(param));
10
        param[0].buffer_type = MYSQL_TYPE_VAR_STRING;    //IN
        param[0].buffer = cf;
        param[0].buffer_length = strlen(cf);

15      if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
                finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters for client
    report\n", true);
        }

20      // Run procedure
        if (mysql_stmt_execute(prepared_stmt) != 0) {
                print_stmt_error(prepared_stmt, "An error occurred while retrieving the client
    report.");
                goto out;
25      }

        // Dump the result set
        dump_result_set(conn, prepared_stmt, "\nClient's credentials");

30      out:
        mysql_stmt_close(prepared_stmt);
```

```
}

void start_client_view(MYSQL *conn){
	// Client main
	char options[2] = {'1','2'};
	char op;

	printf("Welcome in the system!\n");

	if(!parse_config("users/cliente.json", &conf)) {
		fprintf(stderr, "Unable to load client configuration\n");
		exit(EXIT_FAILURE);
	}

	if(mysql_change_user(conn, conf.db_username, conf.db_password, conf.database)) {
		fprintf(stderr, "mysql_change_user() failed\n");
		exit(EXIT_FAILURE);
	}
	printf("\033[2J\033[H");
	while(1) {
		printf("********** CLIENT VIEW **********\n\n");
		printf("*** What should I do for you? ***\n\n");
		printf("1) Show my features\n");
		printf("2) Quit\n");

		op = multiChoice("Select an option", options, 2);

		switch(op) {
			case '1':
				show_client();
				show_phone_numbers();
```

```
                        show_accepted_proposals();
                        break;
                case '2':
                        return;
        default:
                        fprintf(stderr, "Invalid condition at %s:%d\n", __FILE__, __LINE__);
                        abort();
                }
                getchar();
        }

}
```

```
/************************************************************
 *
 ************** MANAGER MAIN ******************
 *
 */

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

#include "defines.h"


void add_operator(MYSQL *conn){
        MYSQL_STMT *prepared_stmt;
        MYSQL_BIND param[4];

        char cf_operator[16];
        char name[45];
        char surname[45];

        //Get paramenters
        printf("\nOperator CF: ");
        getInput(16, cf_operator, false);
        printf("\nOperator name: ");
        getInput(45, name, false);
        printf("\nOperator surname: ");
        getInput(45, surname, false);

        // Prepare stored procedure call
        if(!setup_prepared_stmt(&prepared_stmt, "call inserisci_operatore(?, ?, ?, ?)", conn)) {
                finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize operator insertion statement\n", false);
```

```
        }

        // Prepare parameters
        memset(param, 0, sizeof(param));

        param[0].buffer_type = MYSQL_TYPE_VAR_STRING;   //IN
        param[0].buffer = cf_operator;
        param[0].buffer_length = strlen(cf_operator);

        param[1].buffer_type = MYSQL_TYPE_VAR_STRING;   //IN
        param[1].buffer = name;
        param[1].buffer_length = strlen(name);

        param[2].buffer_type = MYSQL_TYPE_VAR_STRING;   //IN
        param[2].buffer = surname;
        param[2].buffer_length = strlen(surname);

        param[3].buffer_type = MYSQL_TYPE_VAR_STRING;   //IN
        param[3].buffer = cf; //cf of the current user --> manager
        param[3].buffer_length = strlen(cf);

        if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
                finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters for operator
    insertion\n", true);
        }

        // Run procedure
        if (mysql_stmt_execute(prepared_stmt) != 0) {
                print_stmt_error(prepared_stmt, "An error occurred while adding the operator.");
                goto out;
        }

        printf("Operator %s correctly added!\n", cf_operator);
```

```
    out:
        mysql_stmt_close(prepared_stmt);

}

void add_proposal(MYSQL *conn){
        MYSQL_STMT *prepared_stmt;
        MYSQL_BIND param[3];

        char code[45];
        char description[100];

        //Get paramenters
        printf("\nProposal's code: ");
        getInput(45, code, false);
        printf("\nInsert a description for this proposal: \n");
        getInput(100, description, false);

        // Prepare stored procedure call
        if(!setup_prepared_stmt(&prepared_stmt, "call inserisci_proposta(?, ?, ?)", conn)) {
                finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize proposal insertion statement\n", false);
        }

        memset(param, 0, sizeof(param));

        param[0].buffer_type = MYSQL_TYPE_VAR_STRING;   //IN
        param[0].buffer = code;
        param[0].buffer_length = strlen(code);

        param[1].buffer_type = MYSQL_TYPE_VAR_STRING;   //IN
        param[1].buffer = description;
        param[1].buffer_length = strlen(description);
```

```
        param[2].buffer_type = MYSQL_TYPE_VAR_STRING;    //IN
        param[2].buffer = cf;
        param[2].buffer_length = strlen(cf);

        if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
                finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters for proposal insertion\n", true);
        }

        // Run procedure
        if (mysql_stmt_execute(prepared_stmt) != 0) {
                print_stmt_error(prepared_stmt, "An error occurred while adding the proposal.");
                goto out;
        }

        printf("Proposal %s correctly added!\n", code);

    out:
        mysql_stmt_close(prepared_stmt);
}


void archive_proposal(MYSQL *conn){
        MYSQL_STMT *prepared_stmt;
        MYSQL_BIND param[2];

        char code[45];

        //Get paramenters
        printf("\nProposal's code you want to archive: ");
        getInput(45, code, false);

        // Prepare stored procedure call
```

```
        if(!setup_prepared_stmt(&prepared_stmt, "call archivia_proposta(?, ?)", conn)) {
                finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize proposal archive statement\n", false);
        }

        memset(param, 0, sizeof(param));

        param[0].buffer_type = MYSQL_TYPE_VAR_STRING;    //IN
        param[0].buffer = code;
        param[0].buffer_length = strlen(code);

        param[1].buffer_type = MYSQL_TYPE_VAR_STRING;    //IN
        param[1].buffer = cf;
        param[1].buffer_length = strlen(cf);

        if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
                finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters for proposal archive\n", true);
        }

        // Run procedure
        if (mysql_stmt_execute(prepared_stmt) != 0) {
                print_stmt_error(prepared_stmt, "An error occurred while archiving the proposal.");
                goto out;
        }

        printf("Proposal %s correctly archived\n", code);

    out:
        mysql_stmt_close(prepared_stmt);

}
```

```
void start_manager_view(MYSQL *conn){

        char options[4] = {'1','2','3','4'};
        char op;

        printf("Welcome in the system!\n");

        if(!parse_config("users/manager.json", &conf)) {
                fprintf(stderr, "Unable to load manager configuration\n");
                exit(EXIT_FAILURE);
        }

        if(mysql_change_user(conn, conf.db_username, conf.db_password, conf.database)) {
                fprintf(stderr, "mysql_change_user() failed\n");
                exit(EXIT_FAILURE);
        }
        printf("\033[2J\033[H");
        while(1) {
                printf("********** MANAGER VIEW *********\n\n");
                printf("*** What should I do for you? ***\n\n");
                printf("1) Add new operator\n");
                printf("2) Add new business proposal\n");
                printf("3) Archive a business proposal\n");
                printf("4) Quit\n");

                op = multiChoice("Select an option", options, 4);

                switch(op) {
                        case '1':
                                add_operator(conn);
                                break;
                        case '2':
                                add_proposal(conn);
                                break;
```

```
                        case '3':
                                archive_proposal(conn);
                                break;
                        case '4':
                                return;


                        default:
                                fprintf(stderr, "Invalid condition at %s:%d\n", __FILE__, __LINE__);
                                abort();
                }

                getchar();
        }
}
```

```
/*****************************************************************************
 *
 ************** COMMERCIAL SECTOR MEMBER MAIN *****************
 *
 */

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

#include "defines.h"

void add_client( MYSQL *conn){
        MYSQL_STMT *prepared_stmt;
        MYSQL_BIND param[10];

        char cf_client[16];
        char cf_operator[16];
        char name[45];
        char surname[45];
        char email[45];
        char address[45];
        char fax[45];
        int num_phone;
        MYSQL_TIME date;



        //Get paramenters
        printf("\nClient CF: ");
        getInput(16, cf_client, false);
        printf("\nOperatore CF: ");
        getInput(16, cf_operator, false);
```

```
        printf("\nClient name: ");
        getInput(45, name, false);
        printf("\nClient surname: ");
        getInput(45, surname, false);
        printf("\nClient email: ");
        getInput(45, email, false);
        printf("\nClient address: ");
        getInput(45, address, false);
        printf("\nClient fax: ");
        getInput(45, fax, false);

        printf("\nDate: \n");
        printf("\nYear: ");
        scanf("%d", &(date.year) );
        printf("\nMonth: ");
        scanf("%d", &(date.month) );
        printf("\nDay: ");
        scanf("%d", &(date.day) );

        printf("\nPhone number: ");
        scanf("%d", &num_phone );

        // Prepare stored procedure call
        if(!setup_prepared_stmt(&prepared_stmt, "call inserisci_cliente(?, ?, ?, ?, ?, ?, ?, ?, ?, ?)",
conn)) {
                finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize client insertion
statement\n", false);
        }

        // Prepare parameters
        memset(param, 0, sizeof(param));

        param[0].buffer_type = MYSQL_TYPE_VAR_STRING;    //IN
        param[0].buffer = cf_client;
```

```
        param[0].buffer_length = strlen(cf_client);

        param[1].buffer_type = MYSQL_TYPE_VAR_STRING;   //IN
        param[1].buffer = name;
        param[1].buffer_length = strlen(name);

        param[2].buffer_type = MYSQL_TYPE_VAR_STRING;   //IN
        param[2].buffer = surname;
        param[2].buffer_length = strlen(surname);

        param[3].buffer_type = MYSQL_TYPE_VAR_STRING;   //IN
        param[3].buffer = email;
        param[3].buffer_length = strlen(email);

        param[4].buffer_type = MYSQL_TYPE_VAR_STRING;   //IN
        param[4].buffer = address;
        param[4].buffer_length = strlen(address);

        param[5].buffer_type = MYSQL_TYPE_VAR_STRING;   //IN
        param[5].buffer = fax;
        param[5].buffer_length = strlen(fax);

        param[6].buffer_type = MYSQL_TYPE_DATE;         //IN
        param[6].buffer = &date;
        param[6].buffer_length = sizeof(date);

        param[7].buffer_type = MYSQL_TYPE_LONG;         //IN
        param[7].buffer = &num_phone;
        param[7].buffer_length = sizeof(num_phone);

        param[8].buffer_type = MYSQL_TYPE_VAR_STRING;   //IN
        param[8].buffer = cf_operator;
        param[8].buffer_length = strlen(cf_operator);
```

```
        param[9].buffer_type = MYSQL_TYPE_VAR_STRING;    //IN
        param[9].buffer = cf;
        param[9].buffer_length = strlen(cf);

        if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
                finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters for client insertion\n", true);
        }

        // Run procedure
        if (mysql_stmt_execute(prepared_stmt) != 0) {
                print_stmt_error(prepared_stmt, "An error occurred while adding the client.");
                goto out;
        }

        printf("Client %s correctly added!\n", cf_client);

    out:
        mysql_stmt_close(prepared_stmt);

}

void add_company_client( MYSQL *conn){
        MYSQL_STMT *prepared_stmt;
        MYSQL_BIND param[12];

        char cf_client[16];
        char cf_operator[16];
        char name[45];
        char surname[45];
        char email[45];
        char address[45];
        char fax[45];
        char company_name[45];
```

```
        char iva[45];
        int num_phone;
        MYSQL_TIME date;


5


        //Get paramenters
        printf("\nClient CF: ");
        getInput(16, cf_client, false);
10      printf("\nOperatore CF: ");
        getInput(16, cf_operator, false);
        printf("\nClient name: ");
        getInput(45, name, false);
        printf("\nClient surname: ");
15      getInput(45, surname, false);
        printf("\nClient email: ");
        getInput(45, email, false);
        printf("\nClient address: ");
        getInput(45, address, false);
20      printf("\nClient fax: ");
        getInput(45, fax, false);
        printf("\nPartita IVA: ");
        getInput(45, iva, false);
        printf("\nCompany name: ");
25      getInput(45, company_name, false);


        printf("\nDate: \n");
        printf("\nYear: ");
        scanf("%d", &(date.year) );
30      printf("\nMonth: ");
        scanf("%d", &(date.month) );
        printf("\nDay: ");
        scanf("%d", &(date.day) );
```

```
        printf("\nPhone number: ");
        scanf("%d", &num_phone );


        // Prepare stored procedure call
        if(!setup_prepared_stmt(&prepared_stmt, "call inserisci_cliente_societa(?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)", conn)) {
                finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize client insertion statement\n", false);
        }

        // Prepare parameters
        memset(param, 0, sizeof(param));

        param[0].buffer_type = MYSQL_TYPE_VAR_STRING;   //IN
        param[0].buffer = cf_client;
        param[0].buffer_length = strlen(cf_client);

        param[1].buffer_type = MYSQL_TYPE_VAR_STRING;   //IN
        param[1].buffer = name;
        param[1].buffer_length = strlen(name);

        param[2].buffer_type = MYSQL_TYPE_VAR_STRING;   //IN
        param[2].buffer = surname;
        param[2].buffer_length = strlen(surname);

        param[3].buffer_type = MYSQL_TYPE_VAR_STRING;   //IN
        param[3].buffer = email;
        param[3].buffer_length = strlen(email);

        param[4].buffer_type = MYSQL_TYPE_VAR_STRING;   //IN
        param[4].buffer = address;
        param[4].buffer_length = strlen(address);
```

```
        param[5].buffer_type = MYSQL_TYPE_VAR_STRING;   //IN
        param[5].buffer = fax;
        param[5].buffer_length = strlen(fax);

        param[6].buffer_type = MYSQL_TYPE_DATE;         //IN
        param[6].buffer = &date;
        param[6].buffer_length = sizeof(date);

        param[7].buffer_type = MYSQL_TYPE_LONG;         //IN
        param[7].buffer = &num_phone;
        param[7].buffer_length = sizeof(num_phone);

        param[8].buffer_type = MYSQL_TYPE_VAR_STRING;   //IN
        param[8].buffer = iva;
        param[8].buffer_length = strlen(iva);

        param[9].buffer_type = MYSQL_TYPE_VAR_STRING;   //IN
        param[9].buffer = company_name;
        param[9].buffer_length = strlen(company_name);

        param[10].buffer_type = MYSQL_TYPE_VAR_STRING; //IN
        param[10].buffer = cf_operator;
        param[10].buffer_length = strlen(cf_operator);

        param[11].buffer_type = MYSQL_TYPE_VAR_STRING; //IN
        param[11].buffer = cf;
        param[11].buffer_length = strlen(cf);

        if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
                finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters for client insertion\n", true);
        }

        // Run procedure
```

```
        if (mysql_stmt_execute(prepared_stmt) != 0) {
                print_stmt_error(prepared_stmt, "An error occurred while adding the client.");
                goto out;
        }

        printf("Company contact %s correctly added!\n", cf_client);

    out:
        mysql_stmt_close(prepared_stmt);

}

void start_member_view(MYSQL *conn){

        char options[3] = {'1','2','3'};
        char op;

        printf("Welcome in the system!\n");

        if(!parse_config("users/membro settore commerciale.json", &conf)) {
                fprintf(stderr, "Unable to load member configuration\n");
                exit(EXIT_FAILURE);
        }

        if(mysql_change_user(conn, conf.db_username, conf.db_password, conf.database)) {
                fprintf(stderr, "mysql_change_user() failed\n");
                exit(EXIT_FAILURE);
        }
        printf("\033[2J\033[H");
        while(1) {
                printf("********** COMMERCIAL SECTOR MEMBER VIEW **********\n\n");
                printf("*** What should I do for you? ***\n\n");
                printf("1) Add new client\n");
                printf("2) Add new company contact\n");
```

```
                printf("3) Quit\n");

                op = multiChoice("Select an option", options, 3);

                switch(op) {
                        case '1':
                                add_client(conn);
                                break;
                        case '2':
                                add_company_client(conn);
                                break;
                        case '3':
                                return;


                        default:
                                fprintf(stderr, "Invalid condition at %s:%d\n", __FILE__, __LINE__);
                                abort();
                }

                getchar();
        }


}
```

```
/*****************************************************************
 *
 ************** OPERATOR MAIN *****************
 *
 */

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

#include "defines.h"

void add_appointment(MYSQL *conn){
        MYSQL_STMT *prepared_stmt;
        MYSQL_BIND param[6];

        int num_sala;
        char address[45];
        MYSQL_TIME date;
        date.second_part = 0;
        char cf_client[16];
        char codice_proposta[45];


        //Get paramenters

        printf("\nClient's CF: ");
        getInput(16, cf_client, false);
        printf("\nProposal code: ");
        getInput(45, codice_proposta, false);
        printf("\nAddress : ");
        getInput(45, address, false);
        printf("\nNumber sala: ");
```

```
        scanf("%d", &num_sala);

        bzero(&date, sizeof(MYSQL_TIME));       //init struct MYSQL_TIME
        date.time_type = MYSQL_TYPE_DATETIME;

        printf("\nDate: \n");
        printf("\nYear: ");
        scanf("%d", &(date.year) );
        printf("\nMonth: ");
        scanf("%d", &(date.month) );
        printf("\nDay: ");
        scanf("%d", &(date.day) );



        printf("\nTime: \n");
        printf("\nHour: ");
        scanf("%d", &(date.hour) );
        printf("\nMinute: ");
        scanf("%d", &(date.minute) );

        // Prepare stored procedure call
        if(!setup_prepared_stmt(&prepared_stmt, "call inserisci_appuntamento(?, ?, ?, ?, ?, ?)", conn)) {
                finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize operator insertion statement\n", false);
        }

        // Prepare parameters
        memset(param, 0, sizeof(param));

        param[0].buffer_type = MYSQL_TYPE_LONG;                 //IN
        param[0].buffer = &num_sala;
        param[0].buffer_length = sizeof(num_sala);
```

```
        param[1].buffer_type = MYSQL_TYPE_VAR_STRING;    //IN
        param[1].buffer = address;
        param[1].buffer_length = strlen(address);
5
        param[2].buffer_type = MYSQL_TYPE_DATETIME;              //IN
        param[2].buffer = (char *)&date;
        param[2].buffer_length = sizeof(date);
        param[2].is_null=0;
10
        param[3].buffer_type = MYSQL_TYPE_VAR_STRING;    //IN
        param[3].buffer = cf_client;
        param[3].buffer_length = strlen(cf_client);

15      param[4].buffer_type = MYSQL_TYPE_VAR_STRING;    //IN
        param[4].buffer = cf;
        param[4].buffer_length = strlen(cf);

        param[5].buffer_type = MYSQL_TYPE_VAR_STRING;    //IN
20      param[5].buffer = codice_proposta;
        param[5].buffer_length = strlen(codice_proposta);


        if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
25              finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters for
    appointment insertion\n", true);
        }

        // Run procedure
30      if (mysql_stmt_execute(prepared_stmt) != 0) {
                print_stmt_error(prepared_stmt, "An error occurred while adding the appointment.");
                goto out;
        }
```

```
        printf("Appointment correctly added!\n");

    out:
        mysql_stmt_close(prepared_stmt);
5
}

void show_appointments(){
        MYSQL *conn;
10      conn = connection_db();
        MYSQL_STMT *prepared_stmt;
        MYSQL_BIND param[1];

        // Prepare stored procedure call
15      if(!setup_prepared_stmt(&prepared_stmt, "call visualizza_appuntamenti(?)", conn)) {
                finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize operator display statement\n", false);
        }

20      // Prepare parameters
        memset(param, 0, sizeof(param));

        param[0].buffer_type = MYSQL_TYPE_VAR_STRING;   //IN
        param[0].buffer = cf;
25      param[0].buffer_length = strlen(cf);


        if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
                finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters for
30  appointments display\n", true);
        }

        // Run procedure
        if (mysql_stmt_execute(prepared_stmt) != 0) {
```

```
            print_stmt_error(prepared_stmt, "An error occurred while display the appointments.");
            goto out;
        }

        // Dump the result set
        dump_result_set(conn, prepared_stmt, "\nList of appointments assigned to you");


        out:
        mysql_stmt_close(prepared_stmt);
}

void add_conversation(MYSQL *conn){
        MYSQL_STMT *prepared_stmt;
        MYSQL_BIND param[3];


        char cf_client[16];
        char note[100];


        //Get paramenters
        printf("\nClient's CF : ");
        getInput(45, cf_client, false);
        printf("\nNote: \n");
        getInput(100, note, false);


        // Prepare stored procedure call
        if(!setup_prepared_stmt(&prepared_stmt, "call inserisci_conversazione(?, ?, ?)", conn)) {
            finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize operator insertion statement\n", false);
        }
```

```
        // Prepare parameters
        memset(param, 0, sizeof(param));

        param[0].buffer_type = MYSQL_TYPE_VAR_STRING;    //IN
        param[0].buffer = cf_client;
        param[0].buffer_length = strlen(cf_client);

        param[1].buffer_type = MYSQL_TYPE_VAR_STRING;    //IN
        param[1].buffer = cf;
        param[1].buffer_length = strlen(cf);

        param[2].buffer_type = MYSQL_TYPE_VAR_STRING;    //IN
        param[2].buffer = note;
        param[2].buffer_length = strlen(note);


        if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
                finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters for
conversation insertion\n", true);
        }

        // Run procedure
        if (mysql_stmt_execute(prepared_stmt) != 0) {
                print_stmt_error(prepared_stmt, "An error occurred while adding the conversation.");
                goto out;
        }

        printf("Conversation correctly added!\n");

    out:
        mysql_stmt_close(prepared_stmt);

}
```

```
void edit_note(MYSQL *conn){
        MYSQL_STMT *prepared_stmt;
        MYSQL_BIND param[4];
5

        char cf_client[16];
        char note[100];
        int num = 0;
10
        //Get paramenters
        printf("\nClient's CF : ");
        getInput(45, cf_client, false);
        printf("\nNote: \n");
15      getInput(100, note, false);
        printf("\nConversation number: \n");
        scanf("%d", &num);


20      // Prepare stored procedure call
        if(!setup_prepared_stmt(&prepared_stmt, "call modifica_nota_conversazione(?, ?, ?, ?)",
conn)) {
                finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize operator
modification statement\n", false);
25      }

        // Prepare parameters
        memset(param, 0, sizeof(param));

30      param[0].buffer_type = MYSQL_TYPE_VAR_STRING;   //IN
        param[0].buffer = cf_client;
        param[0].buffer_length = strlen(cf_client);

        param[1].buffer_type = MYSQL_TYPE_VAR_STRING;   //IN
```

```
        param[1].buffer = cf;
        param[1].buffer_length = strlen(cf);

        param[2].buffer_type = MYSQL_TYPE_LONG;                    //IN
        param[2].buffer = &num;
        param[2].buffer_length = sizeof(num);

        param[3].buffer_type = MYSQL_TYPE_VAR_STRING;   //IN
        param[3].buffer = note;
        param[3].buffer_length = strlen(note);


        if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
                finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters for conversation modification\n", true);
        }

        // Run procedure
        if (mysql_stmt_execute(prepared_stmt) != 0) {
                print_stmt_error(prepared_stmt, "An error occurred while modifing the conversation.");
                goto out;
        }

        printf("Conversation correctly modified!\n");

    out:
        mysql_stmt_close(prepared_stmt);

}

void delete_note(MYSQL *conn){
        MYSQL_STMT *prepared_stmt;
        MYSQL_BIND param[3];
```

```
        char cf_client[16];
        int num = 0;

        //Get paramenters
        printf("\nClient's CF : ");
        getInput(45, cf_client, false);
        printf("\nConversation number: \n");
        scanf("%d", &num);


        // Prepare stored procedure call
        if(!setup_prepared_stmt(&prepared_stmt, "call elimina_nota_conversazione(?, ?, ?)", conn))
{
                finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize operator modification statement\n", false);
        }

        // Prepare parameters
        memset(param, 0, sizeof(param));

        param[0].buffer_type = MYSQL_TYPE_VAR_STRING;   //IN
        param[0].buffer = cf_client;
        param[0].buffer_length = strlen(cf_client);

        param[1].buffer_type = MYSQL_TYPE_VAR_STRING;   //IN
        param[1].buffer = cf;
        param[1].buffer_length = strlen(cf);

        param[2].buffer_type = MYSQL_TYPE_LONG;                 //IN
        param[2].buffer = &num;
        param[2].buffer_length = sizeof(num);
```

```
        if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
            finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters for conversation modification\n", true);
        }

        // Run procedure
        if (mysql_stmt_execute(prepared_stmt) != 0) {
            print_stmt_error(prepared_stmt, "An error occurred while modifing the conversation.");
            goto out;
        }

        printf("Conversation correctly modified!\n");

    out:
        mysql_stmt_close(prepared_stmt);

}

void show_list(){
        MYSQL *conn;
        conn = connection_db();
        MYSQL_STMT *prepared_stmt;
        MYSQL_BIND param[1];

        // Prepare stored procedure call
        if(!setup_prepared_stmt(&prepared_stmt, "call visualizza_elenco_clienti(?)", conn)) {
            finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize operator display statement\n", false);
        }
```

```
        // Prepare parameters
        memset(param, 0, sizeof(param));

        param[0].buffer_type = MYSQL_TYPE_VAR_STRING;    //IN
        param[0].buffer = cf;
        param[0].buffer_length = strlen(cf);



        if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
                finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters for clients display\n", true);
        }

        // Run procedure
        if (mysql_stmt_execute(prepared_stmt) != 0) {
                print_stmt_error(prepared_stmt, "An error occurred while display the clients.");
                goto out;
        }

        // Dump the result set
        dump_result_set(conn, prepared_stmt, "\nList of clients assigned to you");



        out:
        mysql_stmt_close(prepared_stmt);
}

void get_phone(char *cf_client){
        MYSQL *conn;
        conn = connection_db();
        MYSQL_STMT *prepared_stmt;
        MYSQL_BIND param[1];
```

```
        // Prepare stored procedure call
        if(!setup_prepared_stmt(&prepared_stmt, "call visualizza_numeri_telefono(?)", conn)) {
                finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize phone number statement\n", false);
        }

        // Prepare parameters
        memset(param, 0, sizeof(param));

        param[0].buffer_type = MYSQL_TYPE_VAR_STRING;    //IN
        param[0].buffer = cf_client;
        param[0].buffer_length = strlen(cf_client);

        if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
                finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters for phone number\n", true);
        }

        // Run procedure
        if (mysql_stmt_execute(prepared_stmt) != 0) {
                print_stmt_error(prepared_stmt, "An error occurred while retrieving the phone number.");
                goto out;
        }

        // Dump the result set
        dump_result_set(conn, prepared_stmt, "\nPhone numbers");


        out:
        mysql_stmt_close(prepared_stmt);
}

void get_proposals(char *cf_client){
```

```
        MYSQL *conn;
        conn = connection_db();
        MYSQL_STMT *prepared_stmt;
        MYSQL_BIND param[1];

        // Prepare stored procedure call
        if(!setup_prepared_stmt(&prepared_stmt, "call visualizza_proposte_accettate(?)", conn)) {
                finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize proposal statement\
n", false);
        }

        // Prepare parameters
        memset(param, 0, sizeof(param));

        param[0].buffer_type = MYSQL_TYPE_VAR_STRING;    //IN
        param[0].buffer = cf_client;
        param[0].buffer_length = strlen(cf_client);
        if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
                finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters for accepted
proposals\n", true);
        }

        // Run procedure
        if (mysql_stmt_execute(prepared_stmt) != 0) {
                print_stmt_error(prepared_stmt, "An error occurred while retrieving the accepted
proposals.");
                goto out;
        }

        // Dump the result set
        dump_result_set(conn, prepared_stmt, "\nAccepted proposals");

        out:
```

```
        mysql_stmt_close(prepared_stmt);



}
5
void show_client_features(){
        MYSQL *conn;
        conn = connection_db();
        MYSQL_STMT *prepared_stmt;
10      MYSQL_BIND param[1];

        char cf_client[16];

        printf("\nClient CF: ");
15      getInput(16, cf_client, false);



        // Prepare stored procedure call
        if(!setup_prepared_stmt(&prepared_stmt, "call visualizza_cliente(?)", conn)) {
20              finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize client statement\n",
false);
        }

        // Prepare parameters
25      memset(param, 0, sizeof(param));

        param[0].buffer_type = MYSQL_TYPE_VAR_STRING;   //IN
        param[0].buffer = cf_client;
        param[0].buffer_length = strlen(cf_client);
30
        if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
                finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters for client
report\n", true);
        }
```

```
        // Run procedure
        if (mysql_stmt_execute(prepared_stmt) != 0) {
                print_stmt_error(prepared_stmt, "An error occurred while retrieving the client
5   report.");
                goto out;
        }

        // Dump the result set
10      dump_result_set(conn, prepared_stmt, "\nClient's credentials");

        get_phone(cf_client);
        get_proposals(cf_client);

15      out:
        mysql_stmt_close(prepared_stmt);

}

20  void add_accepted_proposal(MYSQL *conn){
        MYSQL_STMT *prepared_stmt;
        MYSQL_BIND param[2];


25      char cf_client[16];
        char proposal_code[45];


        //Get paramenters
30      printf("\nClient CF: ");
        getInput(16, cf_client, false);
        printf("\nProposal code: \n");
        getInput(45, proposal_code, false);
```

```
        // Prepare stored procedure call
        if(!setup_prepared_stmt(&prepared_stmt, "call inserisci_proposta_accettata(?, ?)", conn)) {
                finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize accepted proposal
5   insertion statement\n", false);
        }

        // Prepare parameters
        memset(param, 0, sizeof(param));
10
        param[0].buffer_type = MYSQL_TYPE_VAR_STRING;    //IN
        param[0].buffer = cf_client;
        param[0].buffer_length = strlen(cf_client);

15      param[1].buffer_type = MYSQL_TYPE_VAR_STRING;    //IN
        param[1].buffer = proposal_code;
        param[1].buffer_length = strlen(proposal_code);



20



        if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
                finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters for accepted
25  proposal insertion\n", true);
        }

        // Run procedure
        if (mysql_stmt_execute(prepared_stmt) != 0) {
30              print_stmt_error(prepared_stmt, "An error occurred while adding the accepted
    proposal.");
                goto out;
        }
```

```
        printf("Accepted proposal correctly added!\n");

    out:
        mysql_stmt_close(prepared_stmt);
}

void add_managment(MYSQL *conn){
        MYSQL_STMT *prepared_stmt;
        MYSQL_BIND param[2];

        char cf_client[16];

        //Get paramenters
        printf("\nClient CF: ");
        getInput(16, cf_client, false);

        // Prepare stored procedure call
        if(!setup_prepared_stmt(&prepared_stmt, "call inserisci_gestione_cliente(?, ?)", conn)) {
                finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize managment insertion statement\n", false);
        }

        // Prepare parameters
        memset(param, 0, sizeof(param));

        param[0].buffer_type = MYSQL_TYPE_VAR_STRING;    //IN
        param[0].buffer = cf_client;
        param[0].buffer_length = strlen(cf_client);
```

```
        param[1].buffer_type = MYSQL_TYPE_VAR_STRING;    //IN
        param[1].buffer = cf;
        param[1].buffer_length = strlen(cf);
5
10      if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
            finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters for managment insertion\n", true);
        }

15      // Run procedure
        if (mysql_stmt_execute(prepared_stmt) != 0) {
            print_stmt_error(prepared_stmt, "An error occurred while adding the managment.");
            goto out;
        }
20
        printf("Managment with %s correctly added!\n", cf_client);

    out:
        mysql_stmt_close(prepared_stmt);
25  }


    void show_conversations(){
        MYSQL *conn;
30      conn = connection_db();
        MYSQL_STMT *prepared_stmt;
        MYSQL_BIND param[1];
```

```
        // Prepare stored procedure call
        if(!setup_prepared_stmt(&prepared_stmt, "call visualizza_conversazioni(?)", conn)) {
                finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize operator statement\
n", false);
        }

        // Prepare parameters
        memset(param, 0, sizeof(param));

        param[0].buffer_type = MYSQL_TYPE_VAR_STRING;    //IN
        param[0].buffer = cf;
        param[0].buffer_length = strlen(cf);

        if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
                finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters for
conversations report\n", true);
        }

        // Run procedure
        if (mysql_stmt_execute(prepared_stmt) != 0) {
                print_stmt_error(prepared_stmt, "An error occurred while retrieving the conversations
report.");
                goto out;
        }

        // Dump the result set
        dump_result_set(conn, prepared_stmt, "\nConversations");

        out:
        mysql_stmt_close(prepared_stmt);


}
```

```
void start_operator_view(MYSQL *conn){

        char options[11] = {'0', '1','2','3','4','5','6','7','8','9','a'};
        char op;

        printf("Welcome in the system!\n");

        if(!parse_config("users/operatore.json", &conf)) {
                fprintf(stderr, "Unable to load operator configuration\n");
                exit(EXIT_FAILURE);
        }

        if(mysql_change_user(conn, conf.db_username, conf.db_password, conf.database)) {
                fprintf(stderr, "mysql_change_user() failed\n");
                exit(EXIT_FAILURE);
        }
        printf("\033[2J\033[H");
        while(1) {
                printf("********** OPERATOR VIEW **********\n\n");
                printf("*** What should I do for you? ***\n\n");
                printf("0) Enter appointment\n");
                printf("1) Show appointments\n");
                printf("2) Enter conversation\n");
                printf("3) Edit conversation note\n");
                printf("4) Delete conversation note\n");
                printf("5) Show clients list\n");
                printf("6) Show client features\n");
                printf("7) Enter accepted proposal\n");
                printf("8) Show your conversations\n");
                printf("9) Enter client's managment\n");
                printf("a) Quit\n");

                op = multiChoice("Select an option", options, 11);
```

```
            switch(op) {
                case '0':
                    add_appointment(conn);
                    break;
                case '1':
                    show_appointments();
                    break;
                case '2':
                    add_conversation(conn);
                    break;
                case '3':
                    edit_note(conn);
                    break;
                case '4':
                    delete_note(conn);
                    break;
                case '5':
                    show_list();
                    break;
                case '6':
                    show_client_features();
                    break;
                case '7':
                    add_accepted_proposal(conn);
                    break;
                case '8':
                    show_conversations();
                    break;
                case '9':
                    add_managment(conn);
                    break;
                case 'a':
                    return;
```

```
                default:
                    fprintf(stderr, "Invalid condition at %s:%d\n", __FILE__, __LINE__);
                    abort();
        }

        getchar();
    }
}
```

```
/*************************************************
 *
 ************** UTILITY ******************
 *
 * here there're implemented some useful
 * functions for the app
 */

#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <termios.h>
#include <sys/ioctl.h>
#include <pthread.h>
#include <signal.h>
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "defines.h"

// Per la gestione dei segnali
static volatile sig_atomic_t signo;
typedef struct sigaction sigaction_t;
static void handler(int s);

MYSQL* connection_db(){
        MYSQL *conn;
        conn = mysql_init (NULL);
        if (conn == NULL) {
```

```
                fprintf (stderr, "mysql_init() failed \n");
                exit(EXIT_FAILURE);
        }

        if (mysql_real_connect(conn, conf.host, conf.db_username, conf.db_password, conf.database,
conf.port, NULL, CLIENT_MULTI_STATEMENTS | CLIENT_MULTI_RESULTS) == NULL) {
                fprintf (stderr, "mysql_real_connect() failed\n");
                mysql_close (conn);
                exit(EXIT_FAILURE);
        }
        return conn;
}


void print_stmt_error (MYSQL_STMT *stmt, char *message)
{
        fprintf (stderr, "%s\n", message);
        if (stmt != NULL) {
                fprintf (stderr, "Error %u (%s): %s\n",
                        mysql_stmt_errno (stmt),
                        mysql_stmt_sqlstate(stmt),
                        mysql_stmt_error (stmt));
        }
}


void print_error(MYSQL *conn, char *message)
{
        fprintf (stderr, "%s\n", message);
        if (conn != NULL) {
                #if MYSQL_VERSION_ID >= 40101
                fprintf (stderr, "Error %u (%s): %s\n",
                mysql_errno (conn), mysql_sqlstate(conn), mysql_error (conn));
                #else
```

```
                fprintf (stderr, "Error %u: %s\n",
                mysql_errno (conn), mysql_error (conn));
                #endif
        }
}


char *getInput(unsigned int lung, char *stringa, bool hide)
{
        char c;
        unsigned int i;

        // Dichiara le variabili necessarie ad un possibile mascheramento dell'input
        sigaction_t sa, savealrm, saveint, savehup, savequit, saveterm;
        sigaction_t savetstp, savettin, savettou;
        struct termios term, oterm;

        if(hide) {
                // Svuota il buffer
                (void) fflush(stdout);

                // Cattura i segnali che altrimenti potrebbero far terminare il programma, lasciando l'utente senza output sulla shell
                sigemptyset(&sa.sa_mask);
                sa.sa_flags = SA_INTERRUPT; // Per non resettare le system call
                sa.sa_handler = handler;
                (void) sigaction(SIGALRM, &sa, &savealrm);
                (void) sigaction(SIGINT, &sa, &saveint);
                (void) sigaction(SIGHUP, &sa, &savehup);
                (void) sigaction(SIGQUIT, &sa, &savequit);
                (void) sigaction(SIGTERM, &sa, &saveterm);
                (void) sigaction(SIGTSTP, &sa, &savetstp);
                (void) sigaction(SIGTTIN, &sa, &savettin);
                (void) sigaction(SIGTTOU, &sa, &savettou);
```

```
        // Disattiva l'output su schermo
        if (tcgetattr(fileno(stdin), &oterm) == 0) {
                (void) memcpy(&term, &oterm, sizeof(struct termios));
                term.c_lflag &= ~(ECHO|ECHONL);
                (void) tcsetattr(fileno(stdin), TCSAFLUSH, &term);
        } else {
                (void) memset(&term, 0, sizeof(struct termios));
                (void) memset(&oterm, 0, sizeof(struct termios));
        }
}

// Acquisisce da tastiera al più lung - 1 caratteri
for(i = 0; i < lung; i++) {

        (void) fread(&c, sizeof(char), 1, stdin);
        if(c == '\n') {
                stringa[i] = '\0';
                break;
        } else
                stringa[i] = c;

        // Gestisce gli asterischi
        if(hide) {
                if(c == '\b') // Backspace
                        (void) write(fileno(stdout), &c, sizeof(char));
                else
                        (void) write(fileno(stdout), "*", sizeof(char));
        }

}

if( strlen(stringa) == 0){
```

```
                printf("Att: non sono consentiti valori nulli!\nInserisci di nuovo: ");
                getInput(lung, stringa, hide);
                return stringa;

        }

        // Controlla che il terminatore di stringa sia stato inserito
        if(i == lung - 1)
                stringa[i] = '\0';

        // Se sono stati digitati più caratteri, svuota il buffer della tastiera
        if(strlen(stringa) >= lung) {
                // Svuota il buffer della tastiera
                do {
                        c = getchar();
                } while (c != '\n');
        }

        if(hide) {
                //L'a capo dopo l'input
                (void) write(fileno(stdout), "\n", 1);

                // Ripristina le impostazioni precedenti dello schermo
                (void) tcsetattr(fileno(stdin), TCSAFLUSH, &oterm);

                // Ripristina la gestione dei segnali
                (void) sigaction(SIGALRM, &savealrm, NULL);
                (void) sigaction(SIGINT, &saveint, NULL);
                (void) sigaction(SIGHUP, &savehup, NULL);
                (void) sigaction(SIGQUIT, &savequit, NULL);
                (void) sigaction(SIGTERM, &saveterm, NULL);
                (void) sigaction(SIGTSTP, &savetstp, NULL);
                (void) sigaction(SIGTTIN, &savettin, NULL);
                (void) sigaction(SIGTTOU, &savettou, NULL);
```

```
                // Se era stato ricevuto un segnale viene rilanciato al processo stesso
                if(signo)
                        (void) raise(signo);
        }

        return stringa;
}


//Per preparare i prepared statement
bool setup_prepared_stmt(MYSQL_STMT **stmt, char *statement, MYSQL *conn)
{
        _Bool update_length = true;

        *stmt = mysql_stmt_init(conn);
        if (*stmt == NULL)
        {
                print_error(conn, "Could not initialize statement handler");
                return false;
        }

        if (mysql_stmt_prepare (*stmt, statement, strlen(statement)) != 0) {
                print_stmt_error(*stmt, "Could not prepare statement");
                return false;
        }

        mysql_stmt_attr_set(*stmt, STMT_ATTR_UPDATE_MAX_LENGTH, &update_length);

        return true;
}


// Per la gestione dei segnali
```

```
static void handler(int s) {
	signo = s;
}


char multiChoice(char *domanda, char choices[], int num)
{

	// Genera la stringa delle possibilità
	char *possib = malloc(2 * num * sizeof(char));
	int i, j = 0;
	for(i = 0; i < num; i++) {
		possib[j++] = choices[i];
		possib[j++] = '/';
	}
	possib[j-1] = '\0'; // Per eliminare l'ultima '/'

	// Chiede la risposta
	while(true) {
		// Mostra la domanda
		printf("%s [%s]: ", domanda, possib);

		char c;
		getInput(1, &c, false);

		// Controlla se è un carattere valido
		for(i = 0; i < num; i++) {
			if(c == choices[i])
				return c;
		}
	}
}
```

```
void finish_with_error(MYSQL *conn, char *message)
{
        print_error(conn, message);
        mysql_close(conn);
        exit(EXIT_FAILURE);
}


void finish_with_stmt_error(MYSQL *conn, MYSQL_STMT *stmt, char *message, bool
close_stmt)
{
        print_stmt_error(stmt, message);
        if(close_stmt)  mysql_stmt_close(stmt);
        mysql_close(conn);
        exit(EXIT_FAILURE);
}


static void print_dashes(MYSQL_RES *res_set)
{
        MYSQL_FIELD *field;
        unsigned int i, j;

        mysql_field_seek(res_set, 0);
        putchar('+');
        for (i = 0; i < mysql_num_fields(res_set); i++) {
                field = mysql_fetch_field(res_set);
                for (j = 0; j < field->max_length + 2; j++)
                        putchar('-');
                putchar('+');
        }
        putchar('\n');
}
```

```
static void dump_result_set_header(MYSQL_RES *res_set)
{
        MYSQL_FIELD *field;
5       unsigned long col_len;
        unsigned int i;

        /* determine column display widths -- requires result set to be */
        /* generated with mysql_store_result(), not mysql_use_result() */
10
        mysql_field_seek (res_set, 0);

        for (i = 0; i < mysql_num_fields (res_set); i++) {
                field = mysql_fetch_field (res_set);
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))
20                      col_len = 4; /* 4 = length of the word "NULL" */
                field->max_length = col_len; /* reset column info */
        }

        print_dashes(res_set);
25      putchar('|');
        mysql_field_seek (res_set, 0);
        for (i = 0; i < mysql_num_fields(res_set); i++) {
                field = mysql_fetch_field(res_set);
                printf(" %-*s |", (int)field->max_length, field->name);
30      }
        putchar('\n');

        print_dashes(res_set);
}
```

```
void dump_result_set(MYSQL *conn, MYSQL_STMT *stmt, char *title)
{
    int i;
    int status;
    int num_fields;       /* number of columns in result */
    MYSQL_FIELD *fields;  /* for result set metadata */
    MYSQL_BIND *rs_bind;  /* for output buffers */
    MYSQL_RES *rs_metadata;
    MYSQL_TIME *date;
    size_t attr_size;

    /* Prefetch the whole result set. This in conjunction with
     * STMT_ATTR_UPDATE_MAX_LENGTH set in `setup_prepared_stmt`
     * updates the result set metadata which are fetched in this
     * function, to allow to compute the actual max length of
     * the columns.
     */
    if (mysql_stmt_store_result(stmt)) {
        fprintf(stderr, " mysql_stmt_execute(), 1 failed\n");
        fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
        exit(0);
    }

    /* the column count is > 0 if there is a result set */
    /* 0 if the result is only the final status packet */
    num_fields = mysql_stmt_field_count(stmt);

    if (num_fields > 0) {
        /* there is a result set to fetch */
        printf("%s\n", title);

        if((rs_metadata = mysql_stmt_result_metadata(stmt)) == NULL) {
```

```
            finish_with_stmt_error(conn, stmt, "Unable to retrieve result metadata\n", true);
        }

        dump_result_set_header(rs_metadata);

        fields = mysql_fetch_fields(rs_metadata);

        rs_bind = (MYSQL_BIND *)malloc(sizeof (MYSQL_BIND) * num_fields);
        if (!rs_bind) {
            finish_with_stmt_error(conn, stmt, "Cannot allocate output buffers\n", true);
        }
        memset(rs_bind, 0, sizeof (MYSQL_BIND) * num_fields);

        /* set up and bind result set output buffers */
        for (i = 0; i < num_fields; ++i) {

            // Properly size the parameter buffer
            switch(fields[i].type) {
                case MYSQL_TYPE_DATE:
                case MYSQL_TYPE_TIMESTAMP:
                case MYSQL_TYPE_TIME:
                    attr_size = sizeof(MYSQL_TIME);
                    break;
                case MYSQL_TYPE_DATETIME:

                    attr_size = sizeof(MYSQL_TIME);
                    break;

                case MYSQL_TYPE_FLOAT:
                    attr_size = sizeof(float);
                    break;
                case MYSQL_TYPE_DOUBLE:
                    attr_size = sizeof(double);
```

```
                        break;
                    case MYSQL_TYPE_TINY:
                        attr_size = sizeof(signed char);
                        break;
                    case MYSQL_TYPE_SHORT:
                    case MYSQL_TYPE_YEAR:
                        attr_size = sizeof(short int);
                        break;
                    case MYSQL_TYPE_LONG:
                    case MYSQL_TYPE_INT24:
                        attr_size = sizeof(int);
                        break;
                    case MYSQL_TYPE_LONGLONG:
                        attr_size = sizeof(int);
                        break;
                    default:
                        attr_size = fields[i].max_length;
                        break;
                }

                // Setup the binding for the current parameter
                rs_bind[i].buffer_type = fields[i].type;
                rs_bind[i].buffer = malloc(attr_size + 1);
                rs_bind[i].buffer_length = attr_size + 1;

                if(rs_bind[i].buffer == NULL) {
                    finish_with_stmt_error(conn, stmt, "Cannot allocate output buffers\n",
true);
                }
            }

            if(mysql_stmt_bind_result(stmt, rs_bind)) {
                finish_with_stmt_error(conn, stmt, "Unable to bind output parameters\n",
true);
```

```
        }

        /* fetch and display result set rows */
        while (true) {
            status = mysql_stmt_fetch(stmt);

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

            putchar('|');

            for (i = 0; i < num_fields; i++) {

                if (rs_bind[i].is_null_value) {
                    printf (" %-*s |", (int)fields[i].max_length, "NULL");
                    continue;
                }

                switch (rs_bind[i].buffer_type) {

                    case MYSQL_TYPE_DATETIME:
                        date = (MYSQL_TIME *)rs_bind[i].buffer;

                        printf(" %d-%02d-%02d %02d:%02d:%02d |", date->year, date->month, date->day, date->hour, date->minute, date->second);

                        break;

                    case MYSQL_TYPE_TIME:
                        date = (MYSQL_TIME *)rs_bind[i].buffer;
                        printf(" %02d:%02d:%02d |", date->hour, date->minute, date->second);
                        break;
```

```
                case MYSQL_TYPE_DATE:
                case MYSQL_TYPE_TIMESTAMP:
                        date = (MYSQL_TIME *)rs_bind[i].buffer;
                        printf(" %d-%02d-%02d |", date->year, date->month, date->day);
                        break;
                case MYSQL_TYPE_VAR_STRING:
                case MYSQL_TYPE_STRING:
                        printf(" %-*s |", (int)fields[i].max_length, (char *)rs_bind[i].buffer);
                        break;

                case MYSQL_TYPE_FLOAT:
                case MYSQL_TYPE_DOUBLE:
                        printf(" %.02f |", *(float *)rs_bind[i].buffer);
                        break;

                case MYSQL_TYPE_LONG:
                case MYSQL_TYPE_SHORT:
                case MYSQL_TYPE_TINY:
                        printf(" %-*d |", (int)fields[i].max_length, *(int *)rs_bind[i].buffer);
                        break;

                case MYSQL_TYPE_NEWDECIMAL:
                        printf(" %-*.02lf |", (int)fields[i].max_length, *(float*) rs_bind[i].buffer);
                        break;

                default:
                                printf("ERROR: Unhandled type (%d)\n", rs_bind[i].buffer_type);
                  abort();
```

```
                        }
                }
                putchar('\n');
                print_dashes(rs_metadata);
        }

        mysql_free_result(rs_metadata); /* free metadata */

        /* free output buffers */
        for (i = 0; i < num_fields; i++) {
                free(rs_bind[i].buffer);
        }
        free(rs_bind);
    }
}
```

```
/**************************************************
 *
 ************** PARSER *****************
 *
 * provide the parsing
 * of a right formuled .json file
 */

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

#include "defines.h"

#define BUFF_SIZE 4096

// The final config struct will point into this
static char config[BUFF_SIZE];

/**
 * JSON type identifier. Basic types are:
 *      o Object
 *      o Array
 *      o String
 *      o Other primitive: number, boolean (true/false) or null
 */
typedef enum {
        JSMN_UNDEFINED = 0,
        JSMN_OBJECT = 1,
        JSMN_ARRAY = 2,
        JSMN_STRING = 3,
        JSMN_PRIMITIVE = 4
```

```
} jsmntype_t;

enum jsmnerr {
        /* Not enough tokens were provided */
        JSMN_ERROR_NOMEM = -1,
        /* Invalid character inside JSON string */
        JSMN_ERROR_INVAL = -2,
        /* The string is not a full JSON packet, more bytes expected */
        JSMN_ERROR_PART = -3
};

/**
 * JSON token description.
 * type         type (object, array, string etc.)
 * start start position in JSON data string
 * end          end position in JSON data string
 */
typedef struct {
        jsmntype_t type;
        int start;
        int end;
        int size;
#ifdef JSMN_PARENT_LINKS
        int parent;
#endif
} jsmntok_t;

/**
 * JSON parser. Contains an array of token blocks available. Also stores
 * the string being parsed now and current position in that string
 */
typedef struct {
        unsigned int pos; /* offset in the JSON string */
        unsigned int toknext; /* next token to allocate */
```

```
	int toksuper; /* superior token node, e.g parent object or array */
} jsmn_parser;

/**
 * Allocates a fresh unused token from the token pool.
 */
static jsmntok_t *jsmn_alloc_token(jsmn_parser *parser, jsmntok_t *tokens, size_t num_tokens) {
	jsmntok_t *tok;
	if (parser->toknext >= num_tokens) {
		return NULL;
	}
	tok = &tokens[parser->toknext++];
	tok->start = tok->end = -1;
	tok->size = 0;
#ifdef JSMN_PARENT_LINKS
	tok->parent = -1;
#endif
	return tok;
}

/**
 * Fills token type and boundaries.
 */
static void jsmn_fill_token(jsmntok_t *token, jsmntype_t type,
			int start, int end) {
	token->type = type;
	token->start = start;
	token->end = end;
	token->size = 0;
}

/**
 * Fills next available token with JSON primitive.
 */
```

```
static int jsmn_parse_primitive(jsmn_parser *parser, const char *js,
		size_t len, jsmntok_t *tokens, size_t num_tokens) {
	jsmntok_t *token;
	int start;

	start = parser->pos;

	for (; parser->pos < len && js[parser->pos] != '\0'; parser->pos++) {
		switch (js[parser->pos]) {
#ifndef JSMN_STRICT
			/* In strict mode primitive must be followed by "," or "}" or "]" */
			case ':':
#endif
			case '\t' : case '\r' : case '\n' : case ' ' :
			case ','  : case ']'  : case '}' :
				goto found;
		}
		if (js[parser->pos] < 32 || js[parser->pos] >= 127) {
			parser->pos = start;
			return JSMN_ERROR_INVAL;
		}
	}
#ifdef JSMN_STRICT
	/* In strict mode primitive must be followed by a comma/object/array */
	parser->pos = start;
	return JSMN_ERROR_PART;
#endif

found:
	if (tokens == NULL) {
		parser->pos--;
		return 0;
	}
	token = jsmn_alloc_token(parser, tokens, num_tokens);
```

```
        if (token == NULL) {
                parser->pos = start;
                return JSMN_ERROR_NOMEM;
        }
        jsmn_fill_token(token, JSMN_PRIMITIVE, start, parser->pos);
#ifdef JSMN_PARENT_LINKS
        token->parent = parser->toksuper;
#endif
        parser->pos--;
        return 0;
}

/**
 * Fills next token with JSON string.
 */
static int jsmn_parse_string(jsmn_parser *parser, const char *js,
                size_t len, jsmntok_t *tokens, size_t num_tokens) {
        jsmntok_t *token;

        int start = parser->pos;

        parser->pos++;

        /* Skip starting quote */
        for (; parser->pos < len && js[parser->pos] != '\0'; parser->pos++) {
                char c = js[parser->pos];

                /* Quote: end of string */
                if (c == '\"') {
                        if (tokens == NULL) {
                                return 0;
                        }
                        token = jsmn_alloc_token(parser, tokens, num_tokens);
                        if (token == NULL) {
```

```
                parser->pos = start;
                return JSMN_ERROR_NOMEM;
            }
            jsmn_fill_token(token, JSMN_STRING, start+1, parser->pos);
#ifdef JSMN_PARENT_LINKS
            token->parent = parser->toksuper;
#endif
            return 0;
        }

        /* Backslash: Quoted symbol expected */
        if (c == '\\' && parser->pos + 1 < len) {
            int i;
            parser->pos++;
            switch (js[parser->pos]) {
                /* Allowed escaped symbols */
                case '\"': case '/' : case '\\' : case 'b' :
                case 'f' : case 'r' : case 'n'  : case 't' :
                    break;
                /* Allows escaped symbol \uXXXX */
                case 'u':
                    parser->pos++;
                    for(i = 0; i < 4 && parser->pos < len && js[parser->pos] != '\
0'; i++) {
                        /* If it isn't a hex character we have an error */
                        if(!((js[parser->pos] >= 48 && js[parser->pos] <= 57) ||
/* 0-9 */
                                (js[parser->pos] >= 65 &&
js[parser->pos] <= 70) || /* A-F */
                                (js[parser->pos] >= 97 &&
js[parser->pos] <= 102))) { /* a-f */
                            parser->pos = start;
                            return JSMN_ERROR_INVAL;
                        }
```

```
                                        parser->pos++;
                                }
                                parser->pos--;
                                break;
        /* Unexpected symbol */
                        default:
                                parser->pos = start;
                                return JSMN_ERROR_INVAL;
                        }
                }
        }
        parser->pos = start;
        return JSMN_ERROR_PART;
}

/**
 * Parse JSON string and fill tokens.
 */
static int jsmn_parse(jsmn_parser *parser, const char *js, size_t len, jsmntok_t *tokens, unsigned int
num_tokens) {
        int r;
        int i;
        jsmntok_t *token;
        int count = parser->toknext;

        for (; parser->pos < len && js[parser->pos] != '\0'; parser->pos++) {
                char c;
                jsmntype_t type;

                c = js[parser->pos];
                switch (c) {
                        case '{': case '[':
                                count++;
                                if (tokens == NULL) {
```

```
                                break;
                        }
                        token = jsmn_alloc_token(parser, tokens, num_tokens);
                        if (token == NULL)
                                return JSMN_ERROR_NOMEM;
                        if (parser->toksuper != -1) {
                                tokens[parser->toksuper].size++;
#ifdef JSMN_PARENT_LINKS
                                token->parent = parser->toksuper;
#endif
                        }
                        token->type = (c == '{' ? JSMN_OBJECT : JSMN_ARRAY);
                        token->start = parser->pos;
                        parser->toksuper = parser->toknext - 1;
                        break;
                case '}': case ']':
                        if (tokens == NULL)
                                break;
                        type = (c == '}' ? JSMN_OBJECT : JSMN_ARRAY);
#ifdef JSMN_PARENT_LINKS
                        if (parser->toknext < 1) {
                                return JSMN_ERROR_INVAL;
                        }
                        token = &tokens[parser->toknext - 1];
                        for (;;) {
                                if (token->start != -1 && token->end == -1) {
                                        if (token->type != type) {
                                                return JSMN_ERROR_INVAL;
                                        }
                                        token->end = parser->pos + 1;
                                        parser->toksuper = token->parent;
                                        break;
                                }
                                if (token->parent == -1) {
```

```
                                if(token->type != type || parser->toksuper == -1) {
                                        return JSMN_ERROR_INVAL;
                                }
                                break;
                        }
                        token = &tokens[token->parent];
                }
#else
                for (i = parser->toknext - 1; i >= 0; i--) {
                        token = &tokens[i];
                        if (token->start != -1 && token->end == -1) {
                                if (token->type != type) {
                                        return JSMN_ERROR_INVAL;
                                }
                                parser->toksuper = -1;
                                token->end = parser->pos + 1;
                                break;
                        }
                }
                /* Error if unmatched closing bracket */
                if (i == -1) return JSMN_ERROR_INVAL;
                for (; i >= 0; i--) {
                        token = &tokens[i];
                        if (token->start != -1 && token->end == -1) {
                                parser->toksuper = i;
                                break;
                        }
                }
#endif
                break;
        case '\"':
                r = jsmn_parse_string(parser, js, len, tokens, num_tokens);
                if (r < 0) return r;
                count++;
```

```
                        if (parser->toksuper != -1 && tokens != NULL)
                                tokens[parser->toksuper].size++;
                        break;
                case '\t' : case '\r' : case '\n' : case ' ':
                        break;
                case ':':
                        parser->toksuper = parser->toknext - 1;
                        break;
                case ',':
                        if (tokens != NULL && parser->toksuper != -1 &&
                                        tokens[parser->toksuper].type != JSMN_ARRAY &&
                                        tokens[parser->toksuper].type != JSMN_OBJECT) {
#ifdef JSMN_PARENT_LINKS
                                parser->toksuper = tokens[parser->toksuper].parent;
#else
                                for (i = parser->toknext - 1; i >= 0; i--) {
                                        if (tokens[i].type == JSMN_ARRAY || tokens[i].type
== JSMN_OBJECT) {
                                                if (tokens[i].start != -1 && tokens[i].end == -1)
{
                                                        parser->toksuper = i;
                                                        break;
                                                }
                                        }
                                }
#endif
                        }
                        break;
#ifdef JSMN_STRICT
                /* In strict mode primitives are: numbers and booleans */
                case '-': case '0': case '1' : case '2': case '3' : case '4':
                case '5': case '6': case '7' : case '8': case '9':
                case 't': case 'f': case 'n' :
                        /* And they must not be keys of the object */
```

```
                        if (tokens != NULL && parser->toksuper != -1) {
                                jsmntok_t *t = &tokens[parser->toksuper];
                                if (t->type == JSMN_OBJECT ||
                                                (t->type == JSMN_STRING && t->size != 0)) {
                                        return JSMN_ERROR_INVAL;
                                }
                        }
#else
                        /* In non-strict mode every unquoted value is a primitive */
                        default:
#endif
                                r = jsmn_parse_primitive(parser, js, len, tokens, num_tokens);
                                if (r < 0) return r;
                                count++;
                                if (parser->toksuper != -1 && tokens != NULL)
                                        tokens[parser->toksuper].size++;
                                break;

#ifdef JSMN_STRICT
                        /* Unexpected char in strict mode */
                        default:
                                return JSMN_ERROR_INVAL;
#endif
                }
        }

        if (tokens != NULL) {
                for (i = parser->toknext - 1; i >= 0; i--) {
                        /* Unmatched opened object or array */
                        if (tokens[i].start != -1 && tokens[i].end == -1) {
                                return JSMN_ERROR_PART;
                        }
                }
        }
```

```
        return count;
}

/**
 * Creates a new parser based over a given  buffer with an array of tokens
 * available.
 */
static void jsmn_init(jsmn_parser *parser) {
        parser->pos = 0;
        parser->toknext = 0;
        parser->toksuper = -1;
}

static int jsoneq(const char *json, jsmntok_t *tok, const char *s)
{
        if (tok->type == JSMN_STRING
           && (int) strlen(s) == tok->end - tok->start
           && strncmp(json + tok->start, s, tok->end - tok->start) == 0) {
                return 0;
        }
        return -1;
}

static size_t load_file(char *filename)
{
        FILE *f = fopen(filename, "rb");
        if(f == NULL) {
                fprintf(stderr, "Unable to open file %s\n", filename);
                exit(1);
        }

        fseek(f, 0, SEEK_END);
        size_t fsize = ftell(f);
```

```
        fseek(f, 0, SEEK_SET);  //same as rewind(f);

        if(fsize >= BUFF_SIZE) {
                fprintf(stderr, "Configuration file too large\n");
                abort();
        }

        fread(config, fsize, 1, f);
        fclose(f);

        config[fsize] = 0;
        return fsize;
}

int parse_config(char *path, struct configuration *conf)
{
        int i;
        int r;
        jsmn_parser p;
        jsmntok_t t[128]; /* We expect no more than 128 tokens */

        load_file(path);

        jsmn_init(&p);
        r = jsmn_parse(&p, config, strlen(config), t, sizeof(t)/sizeof(t[0]));
        if (r < 0) {
                printf("Failed to parse JSON: %d\n", r);
                return 0;
        }

        /* Assume the top-level element is an object */
        if (r < 1 || t[0].type != JSMN_OBJECT) {
                printf("Object expected\n");
                return 0;
```

```
        }

        /* Loop over all keys of the root object */
        for (i = 1; i < r; i++) {
            if (jsoneq(config, &t[i], "host") == 0) {
                /* We may use strndup() to fetch string value */
                conf->host = strndup(config + t[i+1].start, t[i+1].end-t[i+1].start);
                i++;
            } else if (jsoneq(config, &t[i], "username") == 0) {
                conf->db_username = strndup(config + t[i+1].start, t[i+1].end-t[i+1].start);
                i++;
            } else if (jsoneq(config, &t[i], "password") == 0) {
                conf->db_password = strndup(config + t[i+1].start, t[i+1].end-t[i+1].start);
                i++;
            } else if (jsoneq(config, &t[i], "port") == 0) {
                conf->port = strtol(config + t[i+1].start, NULL, 10);
                i++;
            } else if (jsoneq(config, &t[i], "database") == 0) {
                conf->database = strndup(config + t[i+1].start, t[i+1].end-t[i+1].start);
                i++;
            } else {
                printf("Unexpected key: %.*s\n", t[i].end-t[i].start, config + t[i].start);
            }
        }
        return 1;
    }
```

```
/************************************************
 *
 ************** DEFINES *****************
 *
 */
#pragma once

#include <mysql.h>

struct configuration{
        char *host;
        char *db_username;
        char *db_password;
        unsigned int port;
        char *database;
        char password[45];
};
extern char                             cf[16];         /*codice fiscale of the current user*/
extern struct configuration     conf;


extern MYSQL*           connection_db();
                                                /*provide connection to the db*/
extern int                      parse_config(char *path, struct configuration *conf);
                        /*parse a .json, take username, pass, port, db*/
extern char*            getInput(unsigned int lung, char *stringa, bool hide);
                /*provide a safe scan of strings with upperbound lung  */
extern char             multiChoice(char *domanda, char choices[], int num);
                /*provide the choice in the gui*/
extern bool             setup_prepared_stmt(MYSQL_STMT **stmt, char *statement, MYSQL
*conn);         /*setup MYSQL_STMT struct*/
extern void             print_error (MYSQL *conn, char *message);
                                /*print error message for the connection  */
```

```
extern void print_stmt_error (MYSQL_STMT *stmt, char *message);
                /*print error message for the statement  */
extern void dump_result_set(MYSQL *conn, MYSQL_STMT *stmt, char *title);
            /*print with the right measure the output of a select call*/
extern void finish_with_error(MYSQL *conn, char *message);
                /*exit the process after a error in the connection   */
extern void finish_with_stmt_error(MYSQL *conn, MYSQL_STMT *stmt, char
*message, bool close_stmt);     /*exit the process after a error in the statement   */
extern void start_client_view(MYSQL *conn);     /* run with client's
privileges */
extern void start_operator_view(MYSQL *conn);     /* run with operator's
privileges */
extern void start_manager_view(MYSQL *conn);     /* run with manager's
privileges */
extern void start_member_view(MYSQL *conn);     /* run with
commercial sector member's privileges */
```