Basi di Dati e Conoscenza

Progetto A.A. 2020/2021

Customer Relationship Management

0251679

Simone Festa

Indice

1. Descrizione del Minimondo

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

servizi di consulenza acquistati.
* Possibilità di visualizzare l'elenco clienti a cui un utente è assegnato.
* Gestione delle note cliente: ogni volta che un cliente viene contattato deve essere possibile registrare/modificare/cancellare una o più note relative alla conversazione avvenuta e dell'utente che l'ha registrata.
* Gestione delle opportunità: per ogni cliente deve essere possibile inserire una nuova opportunità, cioè una proposta commerciale.
* Gestione degli appuntamenti: deve essere possibile inserire un appuntamento con una nota descrittiva, una data/ora e un cliente a cui è riferito.
* Visualizzazione dell'agenda degli appuntamenti per un utente.
* Possibilità di inserire nuovi servizi di consulenza (riservata ai manager).
* Possibilità di inserire nuovi clienti (riservata al settore commerciale).
* Possibilità di inserire nuovi utenti dell'applicativo web (riservata ai manager).

2. Analisi dei Requisiti

Identificazione dei termini ambigui e correzioni possibili

Linea	Termine	Nuovo termine	Motivo correzione
12	Utente Aziendale	Funzionario	Utente aziendale non identifica chiaramente colui che interagisce con i clienti. Anche un membro del settore commerciale o un manager potrebbe essere identificato come utente aziendale, mentre il "funzionario" descrive con più precisione il ruolo.
14, 22	Operatore dell'azienda di CRM	Funzionario	Nel testo, l'Operatore dell'azienda di CRM è colui che interagisce con i clienti e svolge il ruolo del funzionario. Elimino l'ambiguità rinominandolo funzionario per mantenere la coerenza.
15, 34, 37, 42, 45	Utente	Funzionario	Nel testo, l'utente è usato come sinonimo di Operatore dell'azienda di CRM. Per transitività anche utente viene rinominato Funzionario, che identifica colui che interagisce con i clienti.
17, 18, 23, 39	Proposta Commerciale	Proposta	Proposta Commerciale viene utilizzata una sola volta al posto di "Proposta", sostituendola con quest'ultima uniformiamo la tipologia di proposte riferite al nostro minimondo di riferimento.
28	Membri	Promoter	Il termine "Membro" è molto generale, mentre nel testo lo si riferisce ad una determinata tipologia di dipendente operante nel settore commerciale. Sostituendo tale termine con Promoter, aumenta la leggibilità del testo, fornendo maggior chiarezza.
33, 43	Servizi di consulenza	Proposta commerciale	L'azienda offre proposte commerciali che sono alla base della consulenza tra funzionari e clienti. Nella riga 33 viene scritto "servizi di consulenza acquistati", riferito alla sottoscrizione di una proposta commerciale.
38,39	Opportunità	Proposta	Nella medesima riga viene chiaramente specificata la sinonimia con "Proposta commerciale". Per transitività "Opportunità" viene sostituita da "Proposta".
31	Referente aziendale	Rappresentante	Viene usato "Referente Aziendale" nel caso di una azienda cliente. Usiamo "Rappresentante" per non confonderlo con un elemento del nostro sistema.

Specifica disambiguata

Un sistema di Customer Relationship Management (o gestione delle relazioni con i clienti) è un sistema informativo che verte sulla fidelizzazione del cliente. Si vuole realizzare un sistema CRM per un'azienda marketing-oriented che intende realizzare relazioni durevoli di breve e lungo periodo con i propri clienti, massimizzando quindi il valore degli stessi.
La base di dati del sistema informativo dell'azienda di CRM deve poter memorizzare le informazioni su tutti i clienti di interesse dell'azienda, caratterizzati da nome, cognome, codice fiscale, data di nascita ed un insieme di contatti, sia in forma di indirizzi che di recapiti telefonici, e-mail, 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 funzionario del sistema viene assegnato un sottoinsieme di clienti da gestire.
Su base periodica, i funzionari contattano i clienti mediante uno dei recapiti forniti. In questa fase operativa, il funzionario deve inserire una nota testuale in cui viene riportato un breve resoconto dell'interazione con il cliente, annotando anche possibili risposte affermative alle proposte. Una risposta positiva di accettazione di una proposta 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 i funzionari.
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 promoter 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 rappresentante, con tutti i dettagli e le caratteristiche, l'elenco delle note cliente e l'elenco delle proposte acquistate.

* Possibilità di visualizzare l'elenco clienti a cui un funzionario 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 del funzionario che l'ha registrata.
* Gestione delle proposte: per ogni cliente deve essere possibile inserire una nuova proposta.
* 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 funzionario.
* Possibilità di inserire nuovi proposte (riservata ai manager).
* Possibilità di inserire nuovi clienti (riservata al settore commerciale).
* Possibilità di inserire nuovi funzionari dell'applicativo web (riservata ai manager).

Glossario dei Termini

Termine	Descrizione	Sinonimi	Collegamenti
Funzionario	Entità dell'azienda che si interfaccia con i clienti, avanzando proposte.	Utente aziendale, operatore dell'azienda, utente.	Clienti Proposta commerciale
Cliente	Entità esterna all'azienda che può sottoscrivere proposte mediante l'interazione col funzionario.		Funzionario, Proposta commerciale
Proposta	Contratto stipulato tra il cliente e l'azienda rappresentata dal funzionario.	Servizio di consulenza	Cliente Funzionario
Sede	Indirizzo dove avviene l'incontro tra il funzionario ed il cliente.		Funzionario Cliente
Sala riunione	Luogo fisico, interno alla sede, dove avviene l'incontro tra funzionario e cliente.		Funzionario Cliente
Appuntamento	Intersezione tra proposta, luogo di incontro e data stabilita.		Funzionario Cliente

Raggruppamento dei requisiti in insiemi omogenei

Frasi relative a: Funzionario
Su base periodica, i funzionari contattano i clienti mediante uno dei recapiti forniti. In questa fase operativa, il funzionario deve inserire una nota testuale in cui viene riportato un breve resoconto dell'interazione con il cliente, annotando anche possibili risposte affermative alle proposte commerciali. Agli appuntamenti partecipano i clienti e i funzionari. * Possibilità di visualizzare l'elenco clienti a cui un funzionario 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 del funzionario che l'ha registrata. * Visualizzazione dell'agenda degli appuntamenti per un funzionario.

Frasi relative a: Cliente
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, e-mail, 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. Su base periodica, i funzionari contattano i clienti mediante uno dei recapiti forniti. In questa fase operativa, il funzionario 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. In ciascuna sede sono presenti una o più sale riunione, in cui è possibile ricevere i clienti. Non è possibile assegnare una stessa sala, nello stesso giorno ed alla stessa ora, a più di un cliente. Agli appuntamenti partecipano i clienti e i funzionari. 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 promoter reclutano nuovi clienti e li inseriscono all'interno del sistema. * Visualizzare il singolo cliente, eventualmente con i dati dell'azienda e del rappresentante, con tutti i dettagli e le caratteristiche, l'elenco delle note cliente e l'elenco delle proposte commerciali acquistati. * Possibilità di visualizzare l'elenco clienti a cui un funzionario 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 del funzionario che l'ha registrata. * Gestione delle opportunità: per ogni cliente deve essere possibile inserire una nuova 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. * Possibilità di inserire nuovi clienti (riservata al settore commerciale).

Frasi relative a: Proposta
Il funzionario 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 può essere associata ad un appuntamento in sede. L'azienda ha anche un gruppo di manager che definisce quali sono le proposte 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 commerciali e di segnalare che alcune proposte già presenti nel sistema sono terminate, ovverosia che non possono più essere fornite ai clienti. * 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 delle proposte acquistate. * Gestione delle proposte: per ogni cliente deve essere possibile inserire una nuova proposta. * Possibilità di inserire nuove proposte (riservata ai manager).

Frasi relative a: Appuntamento
Una risposta positiva di accettazione di una proposta può essere associata ad un appuntamento in sede. Agli appuntamenti partecipano i clienti e i funzionari. 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 funzionario.

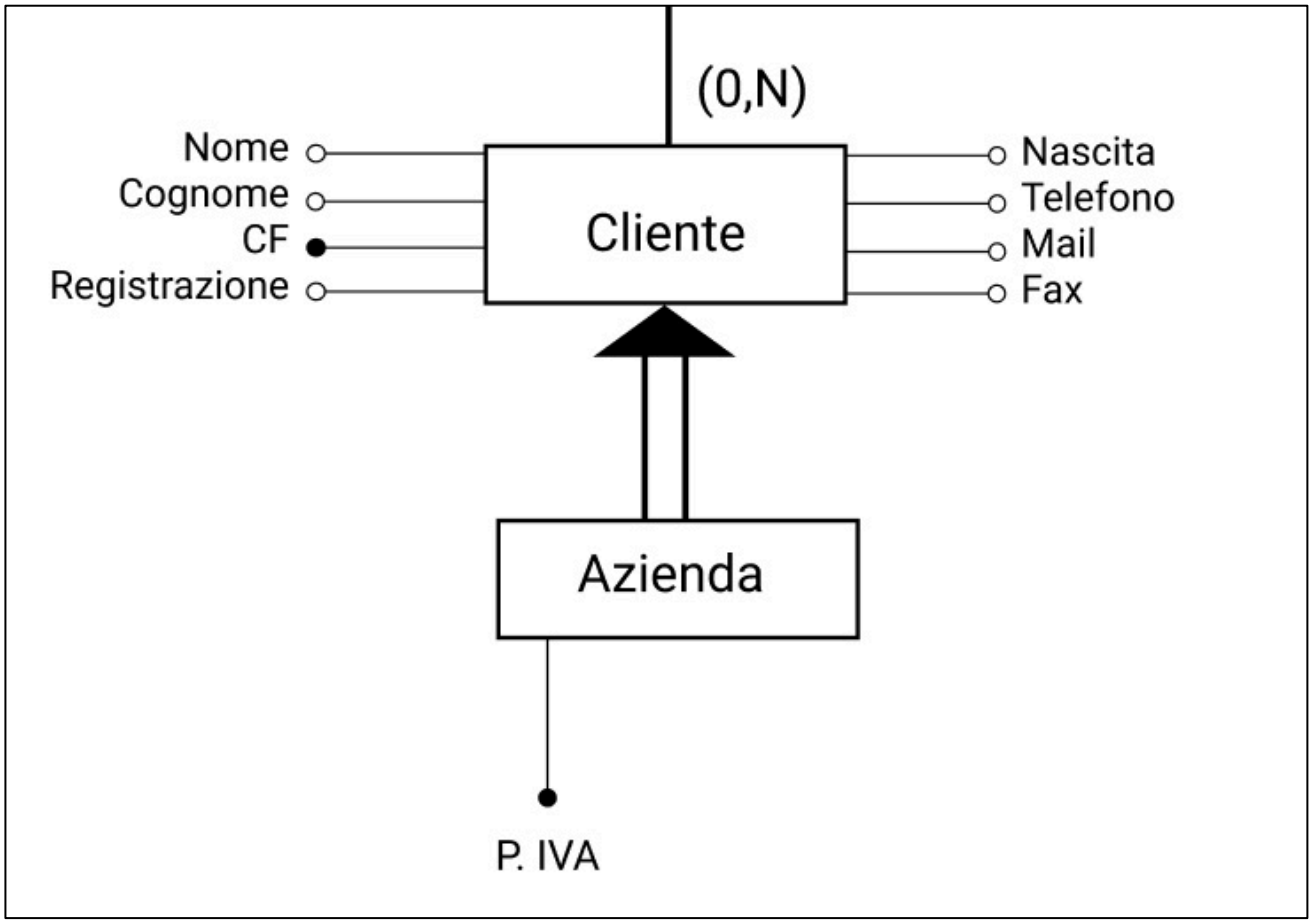
Frasi relative a: Sede
Una risposta positiva di accettazione di una proposta può essere associata ad un appuntamento in sede. L'azienda ha più sedi, ciascuna caratterizzata da un indirizzo. In ciascuna sede sono presenti una o più sale riunione, in cui è possibile ricevere i clienti. Non è possibile assegnare una stessa sede, nello stesso giorno ed alla stessa ora, a più di un cliente.

Frasi relative a: Sala
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.

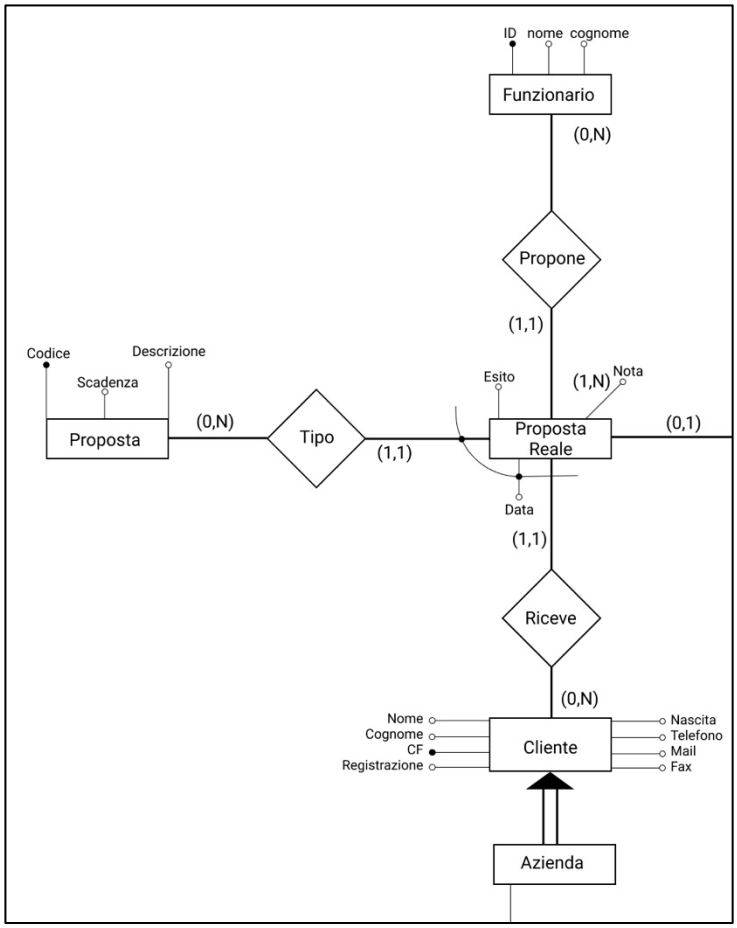
3. Progettazione concettuale

Costruzione dello schema E-R

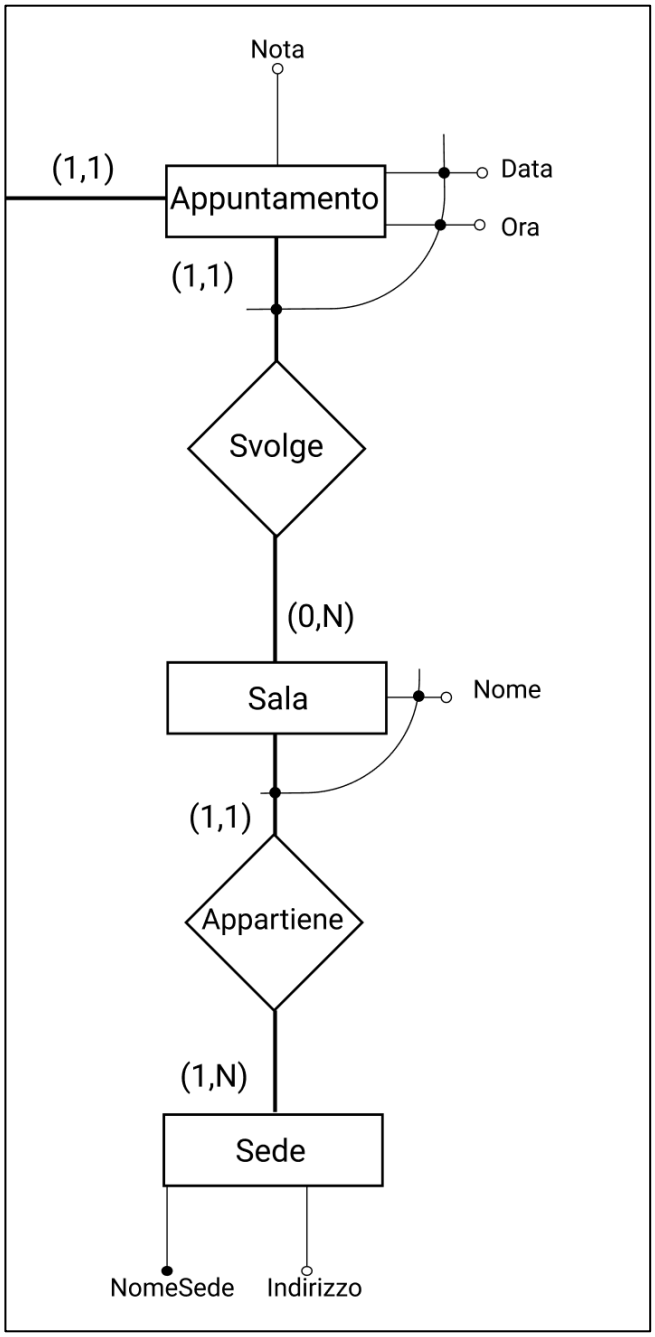
Risolvo “Cliente” e “Azienda”.
Come si evince dal testo, il cliente, inteso come persona fisica ha gli attributi specificati in figura.Nel caso di Azienda “interessa mantenere il numero di partita IVA” [vedi riga 9 della specifica disambiguata].
Tra le richieste troviamo anche “trovare singolo cliente, eventualmente con i dati dell’azienda e del rappresentante” [riga 30]. Il testo ci informa che un’azienda esterna si interfaccia al nostro sistema mediante un rappresentante che condivide gli stessi attributi di un normale cliente, oltre ad avere un attributo “P.IVA” che lo etichetta come un rappresentante di un’azienda.



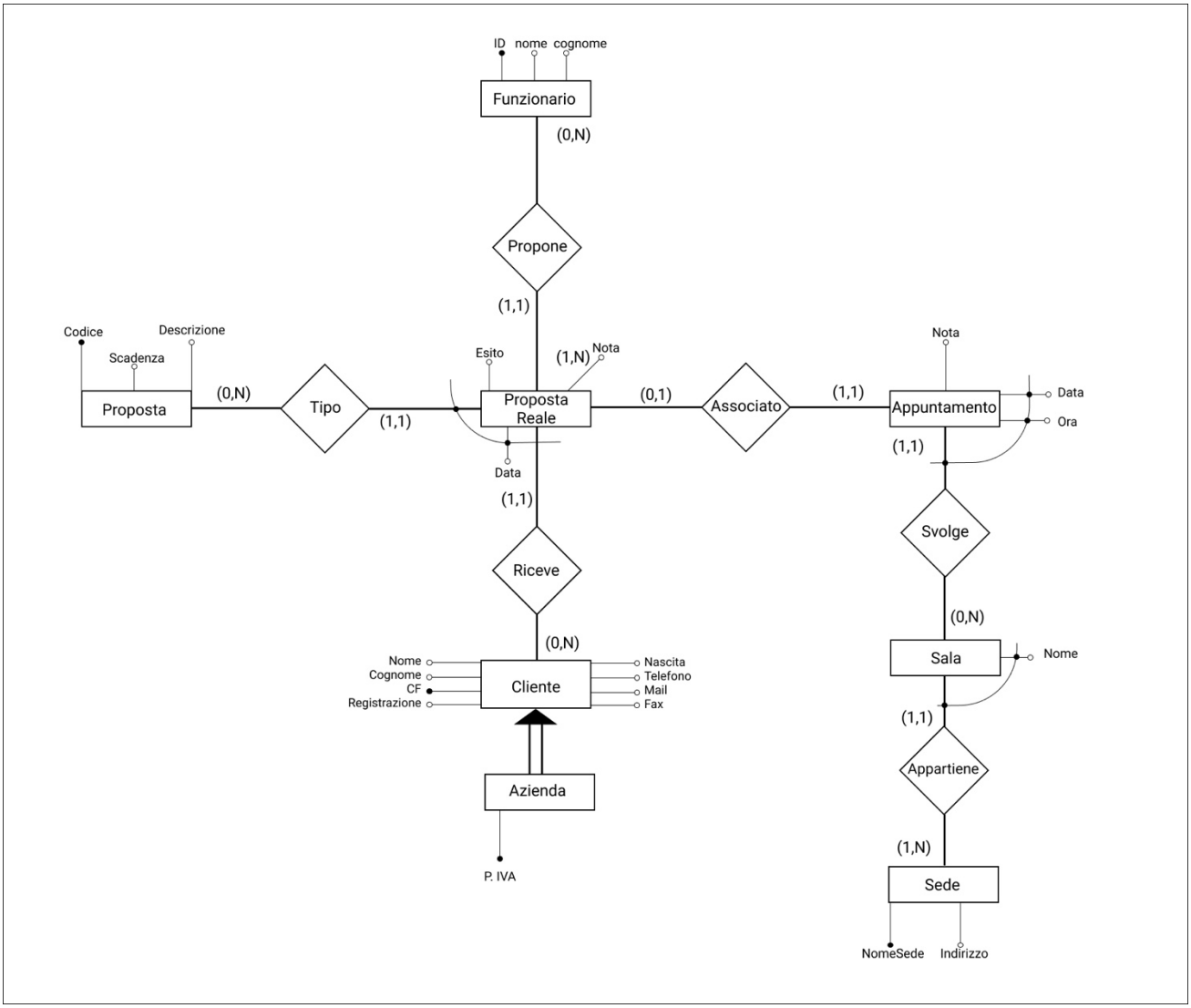
Esamino il rapporto tra la proposta e gli intermediari.
Ad una proposta è possibile associare nessuna o più proposte reali (e.g. nel primo caso considero una proposta appena aggiunta nel sistema, nel secondo una proposta offerta a più clienti). Ad una proposta reale è associata una ed una sola proposta (la proposta reale è basata sulla proposta commerciale ed è unica per l'interazione funzionario-cliente.) basata sul pattern "is a kind of ". Un cliente può ricevere una o più proposta reale nel corso del tempo.
Ad una proposta reale è associata almeno una nota, la quale è un resoconto dell'interazione con il cliente.



Specifico appuntamento e sede.
Una proposta reale può portare ad un appuntamento.
Un appuntamento può essere svolto in una ed una sola sala, in una specifica data.
Una sala può presenziare appuntamenti diversi, purchè si svolgano in momenti temporali diversi.
In una sede deve esserci almeno una sala., o possono svolgersi più appuntamenti.
Una sede ospita almeno una sala e può ospitare più sale.
Una sala è identificata dal suo nome e dalla sede di appartenenza.


Nota
(1,1)
Appuntamento
Data
Ora
(1,1)
Svolge
(0,N)
Sala
Nome
(1,1)
Appartiene
(1,N)
Sede
NomeSede
Indirizzo

Integrazione finale



Regole Aziendali

1) Un cliente deve essere gestito da un solo funzionario.
2) Un cliente viene inserito nel sistema solo se maggiorenne.
3) Un cliente non deve sottoscrivere una proposta scaduta.
4) Un appuntamento è organizzato solo se l'esito di una proposta è positivo.
5) La data fissata per l'appuntamento deve essere successiva a quella della proposta reale.
6) La data di un appuntamento deve essere successiva alla data di inserimento.
7) Una sala di una specifica sede non deve essere assegnata a clienti diversi nello stesso giorno e nella stessa ora.
8) Uno funzionario non deve proporre ad uno stesso cliente la stessa proposta più volte al giorno.
9) La scadenza di una proposta deve essere successiva alla data di inserimento.

Dizionario dei dati

Entità	Descrizione	Attributi	Identificatori
Proposta	Modello astratto della proposta reale.	Codice Scadenza Descrizione	Codice
Proposta reale	Proposta sottoposta al cliente in un determinato momento.	Data Nota	Data CF [esterno] P.Iva [esterno] Codice [esterno]
Funzionario	Soggetto del minimondo che fornisce proposte al cliente.	Id	Id
Cliente	Soggetto del minimondo che valute le proposte del funzionario.	Nome, Cognome, CF, Registrazione, Nascita, Telefono, Mail, Fax,	CF
Azienda	Tipo di cliente avente un rappresentante.	Nome, Cognome, CF, P.IVA Registrazione, Nascita, Telefono, Mail, Fax,	P.IVA

Appuntamento	Incontro tra cliente e funzionario per discutere le proposte.	Nota, Data, Ora	Data, Ora, Nome [esterno] Indirizzo[esterno]
Sala	Luogo interno alla sede dove avviene l'appuntamento.	Nome	Nome, Indirizzo[esterno]
Sede	Luogo fisico dove avviene l'appuntamento.	Indirizzo	Indirizzo

4. Progettazione logica

Volume dei dati

Come da testo, “l’azienda di CRM in questione è di dimensione elevata”.
Per poter stimare il volume dei dati, si è preso come riferimento un insieme di aziende che offrono o includono un CRM o un suo derivato. La stima è aleatoria, poiché nei vari riferimenti riguardanti, ad esempio, il numero di funzionari, troviamo inclusi anche altri ruoli di non interesse.
Prendendo aziende come Vodafone, Wind, Tim troviamo un numero di dipendenti oscillanti tra 6536 e 55000. Altre aziende come Sky presentano un numero di 31000 dipendenti.
Poiché, come detto prima, tali dati non sono riferiti ad un determinato settore dell’azienda, e aggiungendo il fatto che, nel nostro caso, ci occupiamo solo della gestione del rapporto con i clienti, possiamo assumere un numero di funzionari pari a *20000*, in fede alla dimensione elevata proposta dal testo. Per stimare il numero di clienti possiamo fare riferimento al tipo di rapporto tra cliente e funzionario. Poiché un cliente si interfaccia con un solo funzionario, possiamo constatare che quest’ultimo possa gestire un numero discreto di clienti. Stimando che, un funzionario possa gestire un sottogruppo di clienti pari a 20 persone, otteniamo che il numero di clienti si attesti su 400000. Stimiamo il numero di aziende come il 40% della totalità dei clienti, ovvero circa 160000 aziende. Mediamente possiamo ipotizzare 5 tipi di proposte.
Ciascun funzionario proporrà, in un caso medio, ad ognuno dei suoi 20 clienti appartenenti al suo sottogruppo 3 proposte, otteniamo che il numero di proposte reali sia circa 1,2 milioni.
Poiché vengono effettuate circa 1,2 milioni di proposte, facenti parte di 5 tipologie diverse di proposte, la relazione “tipo” è acceduta circa 240000 volte; inoltre, ciascun funzionario effettuerà in media 60 proposte ai clienti del suo sottogruppo.
Se ipotizziamo che ciascun cliente, prima o poi, accetti una proposta reale, e la mantenga per almeno 2 anni otterremo 200.000 appuntamenti, e quindi 200.000 accessi alla relazione “associato”.
Stimiamo 40 sedi in Italia, ottenute ipotizzando che regioni con una popolazione abbondante (vedi Lombardia, Lazio etc) presentino più sedi rispetto ad altre regioni (e.g. meno di 3 milioni di abitanti) aventi una sola sede. Consideriamo che ogni sede abbia 30 sale, otteniamo un totale di 1200 sale. Mediamente in una sala avverranno circa 333 appuntamenti, dilazionati nell’anno.
Tutte le ipotesi appena elencate fanno riferimento ad un caso limite, nel quale ogni cliente accetta una proposta. Tale considerazione può essere assunta come valida considerando due estremi: clienti che preferiscono mantenere una proposta per più anni, e clienti che aggiornano una proposta più volte in un singolo anno.

Concetto nello schema	Tipo[1]	Volume atteso
Proposte	E	5
Proposta reale	E	1.200,000
Funzionario	E	20000
Cliente (generalizzazione)	E	400000
Azienda	E	160000
Cliente (escluso Azienda)	E	240000
Appuntamento	E	200000
Sala	E	1200
Sede	E	40
Tipo	R	240000
Propone	R	60
Riceve	R	3
Associato	R	200000
Svolge	R	333
Appartiene	R	30

Tavola delle operazioni

Si riporta la frequenza delle operazioni in "e/d" cioè "esecuzioni al giorno".

Cod.	Descrizione	Frequenza attesa
Op 1	Visualizzare il singolo cliente, le relative proposte accettate e le note	400.000*5 in 12 mesi → 5500 e/d
Op 2	Inserimento di una nuova proposta	5 volte in 12 mesi → 0,01 e/d.
Op 3	Inserimento nuovi clienti.	20 e/d
Op 4	Modifica di una nota presa durante una proposta reale.	1.200,000/2 in 12 mesi → 1645 e/d
Op 5	Visualizza clienti assegnati ad un funzionario	20*5 in 12 mesi → 0, 27 e/d
Op 6	Inserimento di un appuntamento ed eventuali note.	200000 in 12 mesi → 548 e/d
Op 7	Inserimento di nuovi funzionari da parte di un manager.	2 e/d
Op 8	Visualizzazione del calendario degli appuntamenti.	200.000 in 12 mesi → 555 e/d
Op 9	Comunicazione di una nuova proposta offerta al cliente.	400.000 in 12 mesi →1100 e/d

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

Costo delle operazioni

Si riporta il costo di un'operazione in "a/d" cioè "accessi al giorno".

Operazione 1:

In questa prima operazione abbiamo necessità di visualizzare in lettura le informazioni su un singolo cliente e le relative proposte accettate. Non sono presenti operazioni in scrittura. La stima considera che esistano al massimo 5 tipi di proposte, e si assume che un singolo cliente possa riceverle tutte. Si assume inoltre che una proposta rifiutata non venga nuovamente sottoposta al cliente.

Operazione 1			
Concetto	**Costrutto**	**Accessi**	**Tipo**
Cliente	E	1	L
Riceve	R	5	L
Proposta Reale	E	5	L
Tipo	R	5	L
Proposta	R	5	L

Costo = 5500 * (21L) = 116500 a/d

Operazione 2:

In questa operazione sono previste operazioni in scrittura sull'entità "proposta".

Operazione 2			
Concetto	**Costrutto**	**Accessi**	**Tipo**
Proposta	E	1	S

Costo = 0,01 * S = 0,02 a/d

Operazione 3:

In questa operazione sono previste operazioni in scrittura

Operazione 3			
Concetto	**Costrutto**	**Accessi**	**Tipo**
Cliente	E	1	S

Costo = 10 * S = 20 a/d

Operazione 4:

In questa operazione sono previste operazioni di lettura e scrittura.

Poiché ci interessa modificare una nota, inclusa nell'entità "proposta reale", dovremmo accedere in scrittura in quest'entità ed in lettura alle varie relazioni/entità mantenenti le chiavi esterne di "proposta reale".

Operazione 4			
Concetto	**Costrutto**	**Accessi**	**Tipo**
Proposta Reale	E	1	S
Tipo	R	1	L
Proposta	E	1	L
Riceve	R	1	L
Cliente	E	1	L

Costo = 1645 * (4L +S) = 9870 a/d

Operazione 5:

In questa operazione sono previste unicamente operazioni di lettura.

Un singolo funzionario contatterà in un anno tutti i suoi 20 clienti, poiché siamo interessati solo a conoscere la lista di clienti (e non quali proposte abbiano valutato), possiamo definire che un cliente sia assegnato ad un funzionario nel momento in cui esista almeno una proposta reale tra le due entità.

Operazione 5			
Concetto	**Costrutto**	**Accessi**	**Tipo**
Funzionario	E	1	L
Propone	R	20	L
Proposta	E	20	L
Tipo	R	20	L
Proposta Reale	E	20	L
Riceve	R	20	L
Cliente	E	20	L

Costo = 0,27* (121L) = 32,67 a/d

Operazione 6:

In questa operazione sono previste operazioni di lettura e scrittura.

In scrittura accediamo alla relazione “associato” ed all’entità “appuntamento”.

Poiché l’entità, per essere definita, necessita di specificare una sala ed una sede, dovremo accedere in lettura a queste. Lo stesso ragionamento viene fatto per “associato”, che è collegata con “proposta reale”.

Operazione 6			
Concetto	**Costrutto**	**Accessi**	**Tipo**
Proposta Reale	E	1	L
Tipo	R	1	L
Proposta	E	1	L
Riceve	R	1	L
Cliente	E	1	L
Associato	R	1	S
Appuntamento	R	1	S
Svolge	R	1	S
Sala	E	1	L
Appartiene	R	1	L
Sede	E	1	L

Costo = 548 * (8L +3S) = 7672 a/d

Operazione 7:

Questa operazione, eseguita da un manager, consiste in un accesso in scrittura in ‘Funzionario’.

Operazione 7			
Concetto	**Costrutto**	**Accessi**	**Tipo**
Funzionario	E	1	S

Costo = 2 * (S) = 4 a/d

Operazione 8:

Un funzionario accede in lettura in ‘Appuntamento’, per visualizzare la lista degli appuntamenti. Accede inoltre anche alle tabelle ‘Sala’ e ‘Sede’, elementi informativi facenti parte dell’appuntamento.

Operazione 8			
Concetto	**Costrutto**	**Accessi**	**Tipo**
Appuntamento	E	1	L
Svolge	R	1	L
Sala	E	1	L
Appartiene	R	1	L
Sede	E	1	L

Costo = 555 * (5L) = 2275 a/d

Operazione 9:

Per il completamento di questo task sono previste operazioni di lettura e scrittura. L’unico accesso in scrittura viene effettuato sulla relazione “riceve”, collegante “cliente” e “proposta reale”. Si accede anche all’entità “proposta” per prelevare il codice della proposta accettata.

Operazione 9			
Concetto	**Costrutto**	**Accessi**	**Tipo**
Cliente	E	1	L
Riceve	R	1	S
Proposta Reale	E	1	L
Tipo	R	1	L
Proposta	R	1	L

Costo = 1100 * (4L + S) = 6600 a/d

Ristrutturazione dello schema E-R

- Analizziamo l'unica generalizzazione presente: quella tra "cliente" e "Azienda".
  Nel testo non vengono fatte distinzioni tra le due entità, le quali condividono tutti gli attributi (ad eccezione di "Partita Iva"). La gestione di questo caso può presentare una importante criticità: Risulta difficile poter mantenere informazioni su clienti che, oltre ad una propria utenza, fanno le veci anche di un'azienda. Questo caso non è trascurabile, infatti è plausibile che l'azienda di CRM, dopo aver stipulato un contratto con un'azienda, possa avanzare delle proposte anche ai suoi addetti ai lavori. Poiché "cliente" è identificato dal codice fiscale, un cliente non potrebbe rappresentare sé stesso e una azienda.
  Non esistendo differenziazioni a livello di richieste tra "clienti" e "aziende", e considerando che queste ultime sono circa il 30% dei clienti totali, è possibile mantenere un'unica entità "cliente" con una nuova chiave primaria "Codice Cliente". Si aggiunge anche l'attributo Partita Iva. In questo caso l'attributo potrà presentare un valore NULL in un numero di casi limitato. L'attributo "Partita Iva", oltre che a specificare quest'ultima nel caso delle aziende, permette anche di differenziarle dai semplici clienti.

- L'entità "proposta reale" presenta un attributo multiplo "Nota" con cardinalità (1,N).
  Anche l'entità "appuntamento" presenta uno stesso attributo, con cardinalità (1,1).
  In primo luogo potremmo pensare di accorpare questi due concetti in un'unica entità debole "Nota", tuttavia essa è debole sia nei confronti della "proposta reale" che dell' "appuntamento". Questo comporterebbe quindi a creare l'entità "Nota" che dipenda da entrambi, quando in realtà sappiamo che è plausibile, durante proposte reali, annotare note sulla conversazione senza organizzare un appuntamento. Sostanzialmente questa soluzione non è ottimale per "note" associate ad una proposta, ma non ad un appuntamento.
  L'implementazione suggerita prevede l'introduzione di un'entità debole "Nota" rispetto alla "proposta reale", mentre per l'entità appuntamento dedicheremo un attributo.
  In quest'ultimo caso avremmo potuto anche definire un'entità separata "Nota appuntamento", ma essendo la nota definita come una "descrizione"(quindi un testo scritto) e non come un feedback basato su risposte precompilate, abbiamo casi di ridondanza notevolmente ridotti. (Immagiamo il sistema di feedback di Amazon, è possibile raggruppare le valutazioni per stelle, e quindi filtrare tutte le recensioni aventi un certo numero di stelle, cioè parametro precompilato, ma non posso filtrare le recensioni basandomi sul loro contenuto, poiché è possibile definire la valutazione in modi diversi).

Trasformazione di attributi e identificatori

Nello schema relazione presenta vari identificatori esterni, questo poiché molte relazioni presenti vengono individuate da più fattori. Le criticità maggiori sono su “Proposta reale” e su “Appuntamento”. Questo ampio uso di identificatori esterni si riversa sulla relazione Associato, la quale presenta numerosi vincoli di integrità referenziale ed è identificato da una chiave contenente molti attributi.
È quindi consigliato definire due nuove chiavi primarie per facilitare la traduzione dello schema relazione: una chiave IdPropostaReale ed una IdAppuntamento.

Traduzione di entità e associazioni

Si traduce il modello E-R in uno schema logico equivalente a partire del precedente modello E-R ristrutturato.
Si ottiene la relazione Propone dalla rispettiva associazione molti a molti.
Le associazioni Tipo, Riceve, Svolge, Appartiene ed Associato hanno almeno una cardinalità (1,1), vengono quindi utilizzate come identificatori esterni dalle entità che vi sono collegate con tale cardinalità.

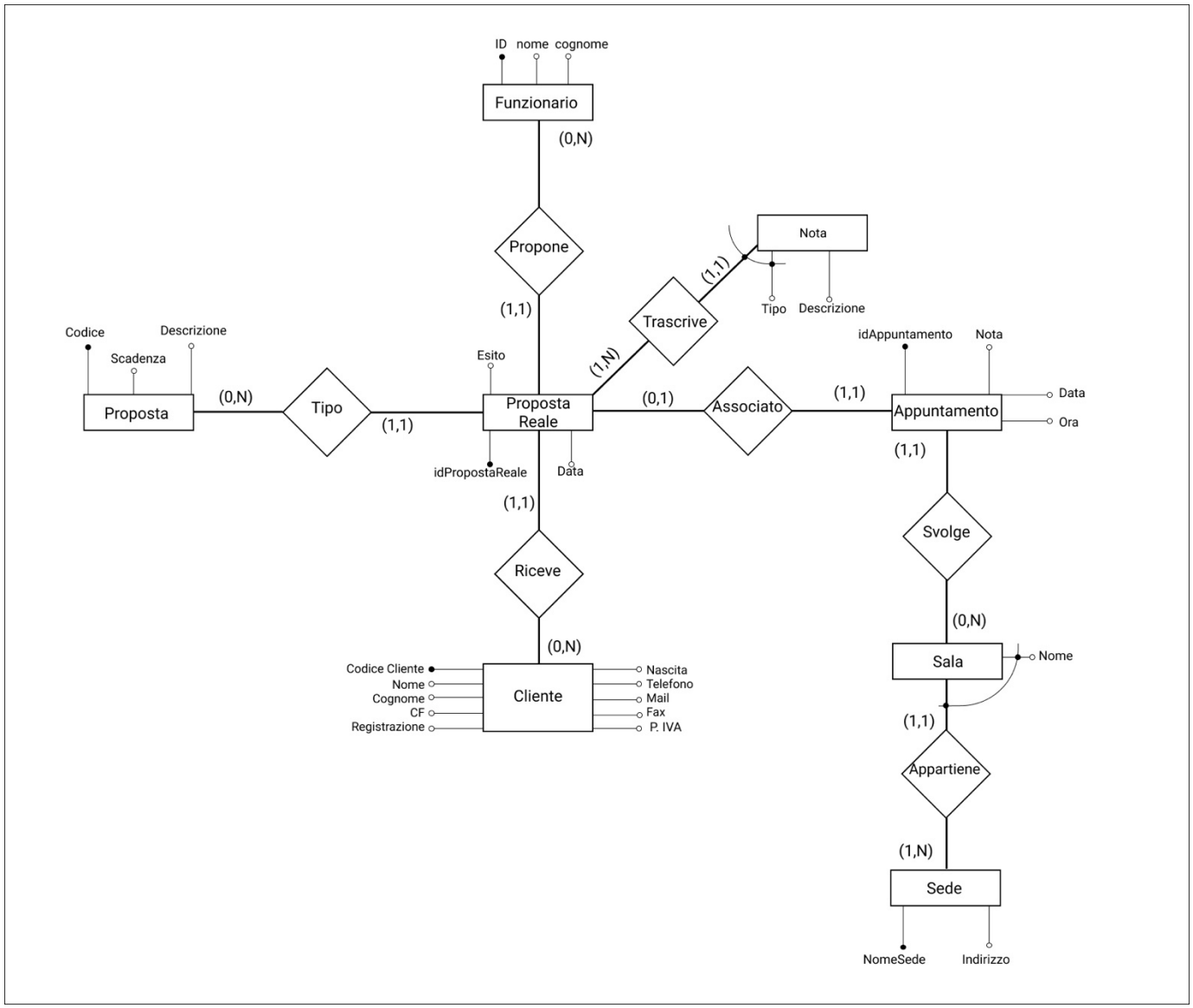
LEGENDA: ChiavePrimaria, *VincoloIntegritàReferenziale*

FUNZIONARIO (idFunzionario, Nome, Cognome)
PROPOSTA REALE (idPropostaReale, Data, Esito, *Cliente*, *Proposta*,*Funzionario*)
PROPOSTA (Codice, Descrizione, Scadenza)
CLIENTE (CodiceCliente, Nome, Cognome, PartitaIva, DataRegistrazione, DataNascita, Telefono, Mail, Fax)
NOTA (Tipo, *PropostaReale*, Descrizione)
APPUNTAMENTO (IdAppuntamento, Data, Ora, Nota, *Sala, Sede, PropostaReale*)
SALA (NomeSala, *Sede*)
SEDE (NomeSede, Indirizzo)

Definiamo inoltre i vincoli di integrità referenziale:

- PROPOSTA REALE (Cliente) ⊆ CLIENTE (CodiceCliente)
- PROPOSTA REALE (Proposta) ⊆ PROPOSTA (Codice)
- PROPOSTA REALE (Funzionario) ⊆ FUNZIONARIO (idFunzionario)
- NOTA (PropostaReale) ⊆ PROPOSTA REALE (idPropostaReale)
- APPUNTAMENTO (Sala) ⊆ SALA (NomeSala)
- APPUNTAMENTO (Sede) ⊆ SALA (NomeSede)
- APPUNTAMENTO (PropostaReale) ⊆ PROPOSTA REALE (idPropostaReale)
- SALA (NomeSede) ⊆ Sede (Nome)

Rappresentazione grafica



Normalizzazione del modello relazionale

Il modello relazione è in *prima forma normale* poiché:

- Ogni relazione presenta una chiave primaria.

- Gli attributi delle relazioni sono definiti su valori atomici, quindi ogni colonna risulta indivisibile.

Il modello relazionale è in *seconda forma normale* poiché non sono presenti dipendenze parziali. Questa proprietà è ovvia per le relazioni dove la chiave è composta da un solo attributo.

Il modello relazionale è in *terza forma normale* poiché non sono presenti dipendenze transitive.

La relazione *Sala* è composta esclusivamente dalla chiave primaria.
Per le altre relazioni è facile verificare che non esistono attributi non-chiave dipendenti funzionalmente da altri attributi non-chiave.

5. Progettazione fisica

Utenti e privilegi

Il sistema prevede tre tipologie di utente: Manager, addetti al settore commerciale e funzionari.

- I manager hanno la possibilità di creare nuove proposte e di segnalare che alcune proposte già presenti nel sistema sono terminate, e quindi non più disponibili.
  Un manager ha anche la possibilità di inserire nuovi funzionari.
  Di conseguenza, un Manager esegue formalmente l'operazione 6.
  I privilegi del manager sono accessi in scrittura e lettura su *Funzionario* e *Proposta*.
- Gli addetti al settore commerciale hanno l'unico ruolo di inserire clienti nella base dati.
  Ciò comporta un accesso in lettura e scrittura su *Cliente*, per realizzare l'operazione 7.
- Il funzionario si interfaccia con molte operazioni, e quindi con molte relazioni.
  I suoi permessi devono consentire di svolgere le operazioni 1, 2, 3, 4, 5.
  Per realizzare ciò è necessario un accesso in lettura e scrittura su Propone, Proposta Reale, Associato, Nota ed Appuntamento (quindi tutte le relazioni toccate per fare una proposta, prendere note e definire un appuntamento); in lettura su Tipo, Proposta, Riceve, Cliente, Svolge, Sala, Appartiene, Sede. (per visionare chi sia il cliente e dove svolgere l'appuntamento)
- Si prevede un quarto tipo di utente per effettuare il login. Tale utente ha esclusivamente accesso in lettura ad una tabella adibita a contenere le informazioni di login per le tipologie di utenti elencati.

Strutture di memorizzazione

Tabella <Proposta>		
Attributo	**Tipo di dato**	**Attributi**[2]
Codice	VARCHAR(15)	PK, NN, UN
Scadenza	DATE	NN
Descrizione	VARCHAR(150)	NN

Tabella <Proposta Reale>		
Attributo	**Tipo di dato**	**Attributi**
idPropostaReale	INT	PK, NN, UN
Data	DATE	NN
Esito	ENUM('Positivo', 'Negativo', 'Accettato', 'Altro')	
Cliente	INT	NN
Proposta	VARCHAR(15)	NN
Funzionario	INT	NN

Nell'attributo 'esito' definiamo un riscontro 'positivo' se il cliente vuole procedere con un appuntamento, 'negativo' altrimenti. Nel caso l'esito sia 'positivo' e si proceda con un appuntamento, l'esito potrà mutare in 'accettato' (il cliente ha finalizzato la proposta).

Tabella <Cliente>		
Attributo	**Tipo di dato**	**Attributi**
CodiceCliente	INT	PK, NN, UN, AI
Nome	VARCHAR(30)	NN
Cognome	VARCHAR(30)	NN
CodiceFiscale	VARCHAR(17)	NN
PartitaIva	VARCHAR(11)	
DataRegistrazione	DATE	NN
DataNascita	DATE	NN
Telefono	VARCHAR(11)	NN
Mail	VARCHAR(50)	NN
Fax	VARCHAR(11)	
Citta	VARCHAR(30)	NN
Indirizzo	VARCHAR(50)	NN

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

Tabella <Funzionario>		
Attributo	**Tipo di dato**	**Attributi**
Id	INT	PK, NN, UN, AI
Nome	VARCHAR(30)	NN
Cognome	VARCHAR(30)	NN

Tabella <Nota>		
Attributo	**Tipo di dato**	**Attributi**
Tipo	ENUM(‘Costo’, ‘Assistenza’, ‘Servizio’, ‘Altro’)	PK, NN
PropostaReale	INT	PK, NN
Descrizione	VARCHAR(150)	NN

Tabella <Appuntamento>		
Attributo	**Tipo di dato**	**Attributi**
idAppuntamento	VARCHAR(20)	PK, NN
Data	DATETIME	NN
Sala	VARCHAR(20)	NN
Sede	VARCHAR(20)	NN
PropostaReale	INT	NN

Tabella <Sala>		
Attributo	**Tipo di dato**	**Attributi**
NomeSala	VARCHAR(20)	PK, NN
Sede	VARCHAR(20)	NN

Tabella <Sede>		
Attributo	**Tipo di dato**	**Attributi**
NomeSede	VARCHAR(20)	PK, NN
Indirizzo	VARCHAR(45)	NN
Citta	VARCHAR(45)	NN

Tabella <Utenti>		
Attributo	**Tipo di dato**	**Attributi**
Username	VARCHAR(30)	PK, NN
Password	VARCHAR(32)	NN
Ruolo	ENUM('Manager', 'Funzionario’, ‘Addetto')	NN

Indici

Tutte le operazioni previste nel sistema sono eseguite tramite selezioni sulle chiavi primarie, di conseguenza si prevedono degli indici su tali attributi.
Si predispone un indice primario per ogni tabella, costruito sugli attributi della chiave primaria.
Si predispone inoltre un indice basato sugli attributi delle chiavi esterne per ogni tabella, per ottimizzare le operazioni di join.

Tabella <Funzionario>		
Nome indice:	**Colonna:**	**Tipo:**
Funzionario_primary_idx	Id	PR

Tabella <PropostaReale>		
Nome indice:	**Colonna:**	**Tipo:**
PropostaReale_primary_idx	idPropostaReale	PR
PropostaReale_cliente_idx	Cliente	IDX
PropostaReale_Proposta_idx	Proposta	IDX
PropostaReale_Funzionario_idx	Funzionario	IDX

Tabella <Proposta>		
Nome indice:	**Colonna:**	**Tipo:**
Proposta_primary_idx	Codice	PR

Tabella <Cliente>		
Nome indice:	**Colonna:**	**Tipo:**
Cliente_primary_idx	CodiceCliente	PR

Tabella <Nota>		
Nome indice:	**Colonna:**	**Tipo:**
Nota_primary_idx	Tipo, PropostaReale	PR
Nota_PropostaReale_idx	PropostaReale	IDX

Tabella <Appuntamento>		
Nome indice:	**Colonna:**	**Tipo:**
Appuntamento_primary_idx	idAppuntamento	PR
Appuntamento_sala_idx	Sala	IDX
Appuntamento_propostaReale_idx	PropostaReale	IDX

Tabella <Sala>		
Nome indice:	**Colonna:**	**Tipo:**
Sala_primary_idx	NomeSala	PR
Sala_sede_idx	Sede	IDX

Tabella <Sede >		
Nome indice:	**Colonna:**	**Tipo:**
Sede_primary_idx	NomeSede	PR

Trigger

Appuntamento

- Con questo trigger, di tipo 'Before Insert', si controlla che l'inserimento di nuovo appuntamento sia ultimato solo se il cliente interessato mostri un esito 'positivo' alla proposta di cui si andrà a discutere.

```
CREATE
DEFINER=`root`@`localhost`
TRIGGER `CRM-db`.`Appuntamento_PropostaPositiva`
BEFORE INSERT ON `CRM-db`.`Appuntamento`
FOR EACH ROW
BEGIN
Declare risultato varchar(15);

select Distinct(Esito) INTO risultato
from `CRM-db`.`PropostaReale` join `CRM-db`.`Appuntamento` on
new.PropostaReale_idPropostaReale = `PropostaReale`.`idPropostaReale`;

IF risultato <> 'Positivo' THEN SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Un
appuntamento è fissabile sono per proposte con esito Positivo!';
END IF;
END
```

- Con questo trigger si effettua un controlla sulla disponibilità di una sala, verificando che essa non sia già stata prenotata nella data richiesta.

```
CREATE
DEFINER=`root`@`localhost`
TRIGGER `CRM-db`.`Appuntamento_SalaPrenotata`
BEFORE INSERT ON `CRM-db`.`Appuntamento`
FOR EACH ROW
BEGIN
```

```
DECLARE num INT;

SET NUM = (SELECT count(*)
                    FROM `CRM-db`.`Appuntamento`
                    WHERE  Giorno = NEW.Giorno and
        Sala_NomeSala = NEW.Sala_NomeSala and
        Sala_Sede_NomeSede = NEW.Sala_Sede_NomeSede);

IF Num = 1 THEN SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'La sala è già
stata prenotata per un altro appuntamento!';
END IF;
END
```

Cliente

- Con questi due trigger si verifica il corretto formato del codice fiscale e della mail del cliente

```
CREATE
DEFINER=`root`@`localhost`
TRIGGER `CRM-db`.`ValidazioneCF`
BEFORE INSERT ON `CRM-db`.`Cliente`
FOR EACH ROW
BEGIN
if new.CodiceFiscale not regexp'^[A-Z]{6}[0-9]{2}[A-Z][0-9]{2}[A-Z][0-9]{3}[A-Z]$' then
signal sqlstate '45001' set message_text="Formato codice Fiscale errato";
END IF;
END
```

```
CREATE
DEFINER=`root`@`localhost`
TRIGGER `CRM-db`.`ValidazioneEmail`
BEFORE INSERT ON `CRM-db`.`Cliente`
FOR EACH ROW
BEGIN

IF NEW.Mail NOT LIKE '%_@__%.__%' THEN
signal sqlstate '45001' set message_text='Formato mail non valido';
END IF;
END
```

PropostaReale

- Con questo trigger verifichiamo che un determinato cliente sia associato ad un solo funzionario.

```
CREATE
DEFINER=`root`@`localhost`
TRIGGER `CRM-db`.`PropostaReale_Funzionario`
BEFORE INSERT ON `CRM-db`.`PropostaReale`
FOR EACH ROW
BEGIN
```

```
DECLARE num INT;
SET num = (SELECT count(Funzionario_idFunzionario)
                    FROM `CRM-db`.`PropostaReale`
        WHERE Cliente_CodiceCliente = NEW.Cliente_CodiceCliente);

IF num = 1 AND NEW.Funzionario_idFunzionario NOT IN
                                    (SELECT  Funzionario_idFunzionario
                                     FROM `CRM-db`.`PropostaReale`
                                     WHERE Cliente_CodiceCliente = NEW.Cliente_CodiceCliente)
THEN SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Un cliente è associato ad un solo
funzionario';
END IF;
END
```

- Il seguente trigger verifica che un cliente non riceva una proposta già valutata.

```
CREATE
DEFINER=`root`@`localhost`
TRIGGER `CRM-db`.`PropostaReale_PropostaDoppia`
BEFORE INSERT ON `CRM-db`.`PropostaReale`
FOR EACH ROW
BEGIN

IF NEW.Proposta_Codice  IN (SELECT Proposta_Codice
                                FROM `CRM-db`.`PropostaReale`
                                WHERE Cliente_CodiceCliente = NEW.Cliente_CodiceCliente)
THEN SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = ' Un cliente non può ricevere una
proposta uguale';
END IF;
END
```

- Questo trigger effettua un controllo sulla data di una proposta reale, evitando che un cliente riceva, in una stessa giornata, più proposte. Insieme al trigger precedente, un cliente non riceverà mai due proposte nello stesso giorno, e non riceverà mai la stessa proposta più volte in date diverse.

```
CREATE
DEFINER=`root`@`localhost`
TRIGGER `CRM-db`.`PropostaReale_PropostaUnica`
BEFORE INSERT ON `CRM-db`.`PropostaReale`
FOR EACH ROW
BEGIN
IF NEW.Giorno  IN (SELECT Giorno
                    FROM `CRM-db`.`PropostaReale`
                    WHERE Cliente_CodiceCliente = NEW.Cliente_CodiceCliente)
THEN SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = ' Un cliente non può ricevere una
determinata proposta più volte al giorno.';
END IF;
END
```

- Questo trigger evita che un cliente riceva, e quindi possa sottoscrivere, una proposta scaduta.

```
CREATE
DEFINER=`root`@`localhost`
TRIGGER `CRM-db`.`PropostaReale_Scaduta`
BEFORE INSERT ON `CRM-db`.`PropostaReale`
FOR EACH ROW
BEGIN

IF NEW.Proposta_Codice  IN (SELECT Codice
                                        FROM `CRM-db`.`Proposta`
                                    WHERE Codice = NEW.Proposta_Codice AND Scadenza < curdate())
THEN SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'La proposta è scaduta';
END IF;
END
```

Eventi

Non vengono predisposti eventi.

Viste

Non vengono predisposte viste.

Stored Procedures e transazioni

1. **Visualizzare il singolo cliente, proposte accettate e note.**

1.1) Definiti in input nome e cognome del cliente, si richiama la stored procedure "visualizzaClienteSingolo", mostrante i dati anagrafici del cliente.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE
`visualizzaClienteSingolo`(in var_nome varchar(30), in var_cognome varchar(30))
BEGIN
set transaction read only;
set transaction isolation level read committed;
start transaction;
select `CodiceCliente`,`Nome`,`Cognome`,`PartitaIVA`,`Telefono`
from `CRM-db`.`Cliente`
where `Nome` = var_Nome and `Cognome` = var_Cognome;
commit;
END
```

1.2) Chiamiamo "visualizzaClienteProposteAccettate" per eseguire l'omonima operazione.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE
`visualizzaClienteProposteAccettate`(in var_nome varchar(30), in var_cognome
varchar(30))
BEGIN
set transaction read only;
set transaction isolation level read committed;
start transaction;
   select `CodiceCliente`as IDUtente ,`Proposta_Codice` as
Proposta,`CodiceFiscale` as CF,`partitaIVA` as IVA, `Telefono`, `Mail`
   from
   `Cliente` join `PropostaReale` on `PropostaReale`.`Cliente_CodiceCliente` =
`Cliente`.`CodiceCliente`
   where `Cliente`.`Nome` = var_Nome and `Cliente`.`Cognome` = var_Cognome
and `Esito` = 'Accettato';
commit;
END
```

1.3) Concludiamo visualizzando le note del cliente.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `listaNoteCliente`(in
var_nome varchar(30), in var_cognome varchar(30))
BEGIN

set transaction read only;
set transaction isolation level read committed;
start transaction;

select `Proposta_Codice` as Proposta,`Cliente_CodiceCliente` as
CodiceCliente,`Tipo`,`Descrizione`
from `Cliente` join `PropostaReale` on `PropostaReale`.`Cliente_CodiceCliente` =
`Cliente`.`CodiceCliente`
join `Nota` on `Nota`.`PropostaReale_idPropostaReale` =
`PropostaReale`.`idPropostaReale`
where `Cliente`.`Nome` = var_Nome and `Cliente`.`Cognome` = var_Cognome;

commit;
END
```

2. Inserimento di una Proposta

2.1) Tale operazione è eseguita unicamente da un utente di tipo "Amministratore", il quale aggiunge una descrizione ed una data di scadenza per la nuova proposta aggiunta. La data di scadenza deve essere postuma alla data di inserimento della proposta nel sistema.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `inserisciProposta`(in
var_Codice VARCHAR(15), in var_Scadenza DATE, in var_Descrizione
VARCHAR(150))
BEGIN

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

set transaction read write;
set transaction isolation level read committed;
start transaction;

If var_Scadenza >= curdate() THEN
INSERT INTO `Proposta`(`Codice`,`Scadenza`, `Descrizione`) VALUES
(var_Codice,var_Scadenza, var_Descrizione);
ELSE SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'La data di scadenza
della proposta è precedente alla data odierna.';
END IF;
COMMIT;
END
```

3. Inserimento nuovi clienti

3.1) Operazione eseguita dall'utente 'addetto', inserisce i dati di un cliente. Vengono eseguiti dei controlli sull'età del cliente e sulla validità dell'email e del codice fiscale.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `inserisciCliente`(in
var_Nome VARCHAR(30), in var_Cognome VARCHAR(30),in var_CodiceFiscale
VARCHAR(17), in var_PartitaIva VARCHAR(11), in var_DataNascita DATE, in
var_Telefono VARCHAR(11), in var_Mail VARCHAR(50), in var_Fax
VARCHAR(11), in var_Citta VARCHAR(30), in var_Indirizzo VARCHAR(50))
BEGIN
DECLARE Difference INT;
DECLARE normaluser INT;
DECLARE ivauser    INT;
DECLARE exit handler for sqlexception
begin
     rollback;
      resignal;
end;

set transaction read write;
set transaction isolation level read committed;
start transaction;

SELECT count(`CodiceFiscale`) INTO normaluser
FROM `CRM-db`.`Cliente`
WHERE `CodiceFiscale` = var_CodiceFiscale AND `PartitaIva` = '';

IF normaluser = 1 and var_PartitaIva = '' THEN SIGNAL SQLSTATE '45000' SET
MESSAGE_TEXT = 'Il cliente senza partita IVA è già presente.';
END IF;
```

```
SELECT count(`CodiceFiscale`) INTO ivauser
FROM `CRM-db`.`Cliente`
WHERE `CodiceFiscale` = var_CodiceFiscale AND  `PartitaIva`<> '';

IF ivauser = 1 and  var_PartitaIva <> '' THEN SIGNAL SQLSTATE '45000' SET
MESSAGE_TEXT = 'Il cliente con partita IVA è già presente.';
END IF;

SELECT timestampdiff(YEAR,var_DataNascita, curdate()) INTO Difference;
If Difference < '18' THEN SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Il
cliente deve essere maggiorenne';
ELSE INSERT INTO `Cliente`(`Nome`, `Cognome`,`CodiceFiscale`,`PartitaIva`,
`DataRegistrazione`,`DataNascita`, `Telefono`,`Mail`, `Fax`,`Citta`,`Indirizzo`)
VALUES (var_Nome, var_Cognome, var_CodiceFiscale, var_PartitaIva, curdate(),
var_DataNascita, var_Telefono, var_Mail, var_Fax, var_Citta,var_Indirizzo);

END IF;

COMMIT;

END
```

4. Gestione Nota

4.1) Si aggiunge una nota riferita ad una proposta, ottenuta durante una proposta reale.
Si definisce una descrizione riferita ad un parametro dell'offerta (Servizio, Costo etc…)
Se il parametro ha già una descrizione, lo si aggiorna, altrimenti lo si aggiunge.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `gestioneNota`(in
var_PropostaReale INT, in var_Tipo ENUM('Costo', 'Assistenza', 'Servizio', 'Altro'),
in var_Descrizione VARCHAR(150))
BEGIN
declare controllo VARCHAR(15);
begin
        rollback;
end;
set transaction read write;
set transaction isolation level read committed;
start transaction;

select `Tipo` INTO controllo
from `Nota`
where var_PropostaReale = `PropostaReale_idPropostaReale`AND var_Tipo =
`Tipo`;
if controllo is NULL
THEN
INSERT INTO `Nota`(`Tipo`, `Descrizione`, `PropostaReale_idPropostaReale`)
VALUES (var_Tipo, var_Descrizione,var_PropostaReale);

ELSE
UPDATE `Nota` SET `Tipo`= var_Tipo, `Descrizione`= var_Descrizione
WHERE `Nota`.`PropostaReale_idPropostaReale` = var_PropostaReale AND
`Nota`.`Tipo` = var_Tipo;

END IF;
COMMIT;
END
```

4.2) A livello puramente di usabilità si introduce anche la stored Procedure “lista Comunicazioni”, la quale stampa tutte le proposte reali per cui è possibile aggiungere una nota.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `listaComunicazioni`()
BEGIN

set transaction read only;
set transaction isolation level read committed;
start transaction;

SELECT idPropostaReale, Proposta_Codice, Nome, Cognome
FROM `CRM-db`.PropostaReale join `CRM-db`.Cliente
on  `Cliente`.CodiceCliente = `PropostaReale`.Cliente_CodiceCliente;
COMMIT;
END
```

5. Visualizza Clienti associati ad un funzionario

5.1) Definito il codice identificativo del funzionario, si ottiene la lista dei suoi clienti.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE
`visualizzaClientiFunzionario`(in var_Funzionario INT)
BEGIN
set transaction read only;
set transaction isolation level read committed;
start transaction;

   select `Nome`, `Cognome`, `CodiceCliente`,`CodiceFiscale`,`Proposta_Codice`
   from `Cliente` join `PropostaReale` on `PropostaReale`.`Cliente_CodiceCLiente` =
`Cliente`.`CodiceCliente`
   where `PropostaReale`.`Funzionario_idFunzionario` = var_Funzionario;
commit;
END
```

6. Inserimento di un appuntamento ed eventuali note.

6.1) L'appuntamento viene definito inserendo in input data, orario, sala, sede, nota e proposta Reale associata all'appuntamento. Si verifica che la data sia postuma alla data odierna.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE
`inserisciAppuntamento`(in var_Data DATETIME, in var_Sala_NomeSala
VARCHAR(20), in var_Sala_Sede_NomeSede VARCHAR(20), in
var_PropostaReale_idPropostaReale INT, in var_Nota VARCHAR(150))
BEGIN

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

set transaction read write;
set transaction isolation level serializable;
start transaction;

If var_Data >= curdate() THEN
INSERT INTO `Appuntamento`(`Giorno`,`Sala_NomeSala`,
`Sala_Sede_NomeSede`, `PropostaReale_idPropostaReale`,`Nota`) VALUES
(var_Data, var_Sala_NomeSala, var_Sala_Sede_NomeSede,
var_PropostaReale_idPropostaReale,var_Nota);
ELSE SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'La data
appuntamento deve essere postuma alla data odierna.';
END IF;
COMMIT;
END
```

6.2) Per un'esperienza migliore, viene aggiunta una routine per visualizzare le proposte per cui è possibile definire un appuntamento.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `visualizzaClientePropostePendenti`()
BEGIN

set transaction read only;
set transaction isolation level read committed;
start transaction;

 select
`CodiceCliente`,`Cliente`.`Nome`,`Cliente`.`Cognome`,`idPropostaReale`,`Funzionario`.`Nome`,`F
unzionario`.`Cognome`,`Funzionario`.`idFunzionario`
   from
   `Cliente` join `PropostaReale` on `PropostaReale`.`Cliente_CodiceCLiente` =
`Cliente`.`CodiceCliente`
   join  `Funzionario` on  `Funzionario`.`idFunzionario` = `Funzionario_idFunzionario`
   where  `Esito` = 'Positivo'
AND  `PropostaReale`.`idPropostaReale` NOT IN ( SELECT PropostaReale_idPropostaReale
                                                FROM  `CRM-db`.Appuntamento);
commit;
END
```

6.3) Per un'esperienza migliore, viene aggiunta una routine per visualizzare un calendario degli appuntamenti già presenti.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `calendario`()
BEGIN
set transaction read only;
set transaction isolation level read committed;
start transaction;
select date_format(`Appuntamento`.`Giorno`,'%d-%m-%Y %k:%i') as Giorno,Sala_NomeSala
as Sala, Sala_Sede_NomeSede as Sede
from `CRM-db`.`Appuntamento`;
```

```
COMMIT;
END
```

6.4) Per un'esperienza migliore, viene aggiunta una routine per visualizzare sale e sedi.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `listaSedi`()
BEGIN
set transaction read only;
set transaction isolation level read committed;
start transaction;

SELECT nomeSala as Sala,NomeSede as Sede, Indirizzo, Citta
FROM `CRM-db`.Sala join `CRM-db`.Sede on NomeSede = Sede_NomeSede;
COMMIT;
END
```

7) Inserimento di nuovi funzionari da parte del manager.

7.1) Si tratta di una semplice insert.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `inserisciFunzionario`(IN
var_nome varchar(30), IN var_cognome VARCHAR(30))
BEGIN
 insert into Funzionario(`Nome`,`Cognome`) VALUES (var_nome, var_cognome);
END
```

8) Visualizzazione del calendario degli appuntamenti.

8.1) Per differenziarsi dalla stored procedure precedente, in questo caso è possibile vedere gli appuntamenti fissati per un dato giorno.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE
`visualizzaAppuntamento`(in var_Giorno DATETIME)
BEGIN
set transaction read only;
set transaction isolation level read committed;
start transaction;

  select `Sala_NomeSala`as Sala,`Sala_Sede_NomeSede`as
Sede,date_format(`Appuntamento`.`Giorno`,'%d-%m-%Y %k:%i') as
Giorno,`Nome`,`Cognome`, `Proposta_Codice`, `Appuntamento`. `Nota`
  from `Appuntamento` join `PropostaReale` on PropostaReale_idPropostaReale =
idPropostaReale
  join `Cliente` on Cliente_CodiceCliente = CodiceCliente
  where DATE(`Appuntamento`.`Giorno`) = DATE(var_Giorno);
commit;
END
```

9) Comunica proposta reale al cliente.

9.1) Si inseriscono informazioni relative ad una proposta reale effettuata ad un cliente.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `comunicaProposta`( in
var_Esito ENUM('Positivo','Negativo','Accettato','Altro'),in var_PropostaCodice
VARCHAR(15), in var_Cliente_CodiceCliente INT, in var_Funzionario_idFunzionario
INT, out idProposta INT)
BEGIN
DECLARE exit handler for sqlexception
begin
    rollback;
    resignal;
end;
set transaction read write;
set transaction isolation level serializable;
start transaction;
If var_PropostaCodice IN (SELECT Codice
                          FROM `Proposta`)

THEN INSERT INTO `PropostaReale`(`Giorno`, `Esito`,`Proposta_Codice`,
`Cliente_CodiceCliente`, `Funzionario_idFunzionario`) VALUES (curdate(),
var_Esito, var_PropostaCodice, var_Cliente_CodiceCliente,
var_Funzionario_idFunzionario);
SET idProposta = last_insert_id();
ELSE SIGNAL SQLSTATE `45000` SET MESSAGE TEXT = ‘codice Proposta
inesistente’;
END IF;
COMMIT;
END
```

9.2) Per migliorare l'esperienza di utilizzo, viene fornita una di tutti i clienti, per visualizzare il cliente per il quale si vuole inoltrare una proposta.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `listaClienti`()
BEGIN
set transaction read only;
set transaction isolation level read committed;
start transaction;
SELECT distinct(codiceCliente), Nome, Cognome, CodiceFiscale, PartitaIVA,
IFNULL(Funzionario_idFunzionario,'') as Funzionario
FROM `CRM-db`.Cliente left join `CRM-db `.PropostaReale on CodiceCliente = Cliente_CodiceCliente;
COMMIT;
END
```

10) Visualizzazione nota cliente

10.1) È possibile visionare per una determinata proposta, tutte le note associate.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `visualizzaNota`(in
var_Codice varchar(15))
BEGIN
set transaction read only;
set transaction isolation level read committed;
start transaction;

 select `CodiceCliente`, `Nome`, `Cognome`,`CodiceFiscale`,`Tipo`,`Descrizione`
   from `Cliente` join `PropostaReale` on `PropostaReale`.`Cliente_CodiceCLiente`
= `Cliente`.`CodiceCliente`
   join `Nota` on  `Nota`.`PropostaReale_idPropostaReale` =
`PropostaReale`.`idPropostaReale`
   where `Proposta_Codice` =  var_Codice;
commit;
END
```

11) Stipulazione di un contratto.

11.1) E' possibile, per appuntamenti precedentemente definiti, concludere una proposta (e quindi stipulare un contratto), oppure declinare l'offerta.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `esitoAppuntamento`(in
var_idAppuntamento VARCHAR(20), in var_Esito
ENUM('Positivo','Accettato','Rifiutato','Altro'))
BEGIN
DECLARE id INT;
DECLARE codice INT;
DECLARE exit handler for sqlexception
begin
     rollback;
   resignal;
end;

set transaction read write;
set transaction isolation level read committed;
start transaction;

SELECT count(idAppuntamento) INTO codice
FROM `CRM-db`.`Appuntamento`
WHERE  idAppuntamento = var_idAppuntamento;

IF codice = 0 THEN SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Non
esiste un appuntamento con tale codice.';
ELSE

IF var_Esito ='Accettato' or 'Negativo' THEN
SELECT idPropostaReale into id
FROM `CRM-db`.`PropostaReale` join `CRM-db`.`Appuntamento` on
`PropostaReale`.`idPropostaReale` =
`Appuntamento`.`PropostaReale_idPropostaReale`
```

```
    WHERE `idAppuntamento` = var_idAppuntamento;

    UPDATE `CRM-db`.`PropostaReale` SET `Esito` = var_Esito
    WHERE `PropostaReale`.`idPropostaReale` = id;
    ELSE SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Esito appuntamento
    può essere solo "Accettato" o "Negativo"';

    END IF;
    END IF;

    COMMIT;
    END
```

11.2) Per una migliore esperienza d'utilizzo, si mostrano prima di definire l'esito, tutti gli appuntamenti fissati, per scegliere quello da concludere.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `listaCodiciAppuntamenti`()
BEGIN

set transaction read only;
set transaction isolation level read committed;
start transaction;

SELECT idAppuntamento, idPropostaReale, `Cliente`.`Nome`, `Cliente`.`Cognome`
FROM `CRM-db`.PropostaReale join `CRM-db`.Appuntamento on idPropostaReale =
PropostaReale_idPropostaReale
join `CRM-db`.Cliente on CodiceCliente = Cliente_CodiceCliente
WHERE `PropostaReale`.`Esito` <> "Accettato";

COMMIT;
END
```

12) Registrazione

12.1) attualmente è presente un utente per ogni tipologia:

Ruolo	**Username**	**Password**
Funzionario	Funzionario	Funzionario
Manager	Manager	Manager
Addetto	Addetto	Addetto

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `accountRegistrazione`(in
var_Username VARCHAR(30),in var_Password VARCHAR(30),in var_Ruolo
ENUM('Manager', 'Funzionario', 'Addetto'))
BEGIN
INSERT INTO `CRM-db`.`Utenti` (`Username`, `Password`, `Ruolo`)
VALUES (var_Username, md5(var_Password), var_Ruolo);
END
```

13) Login

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `login`(in var_Username
VARCHAR(30),in var_Password VARCHAR(30),out var_Ruolo INT)
BEGIN
 declare var_user_role ENUM('Funzionario', 'Addetto', 'Manager');

 select `Ruolo` from `Utenti`
  where `Username` = var_Username
       and `Password` = md5(var_Password)
       into var_user_role;

 if var_user_role = 'Funzionario' then
  set var_Ruolo = 1;
 elseif var_user_role = 'Manager' then
  set var_Ruolo = 2;
```

```
    elseif var_user_role = 'Addetto' then
     set var_Ruolo = 3;
    else
     set var_Ruolo = 0;
    end if;

   END
```

Appendice: Implementazione

Codice SQL per instanziare il database

Di seguito si riporta il codice SQL per instanziare lo schema e le tabelle della base di dati.

```
CREATE DATABASE  IF NOT EXISTS `CRM-db` /*!40100 DEFAULT CHARACTER SET
utf8mb4 COLLATE utf8mb4_0900_ai_ci */ /*!80016 DEFAULT ENCRYPTION='N' */;
USE `CRM-db`;
-- MySQL dump 10.13  Distrib 8.0.26, for Linux (x86_64)
--
-- Host: localhost    Database: CRM-db
-- ------------------------------------------------------
-- Server version    8.0.26-0ubuntu0.20.04.2

/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
/*!40101 SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
/*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;
/*!50503 SET NAMES utf8 */;
/*!40103 SET @OLD_TIME_ZONE=@@TIME_ZONE */;
/*!40103 SET TIME_ZONE='+00:00' */;
/*!40014 SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0 */;
/*!40014 SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS,
FOREIGN_KEY_CHECKS=0 */;
/*!40101 SET @OLD_SQL_MODE=@@SQL_MODE,
SQL_MODE='NO_AUTO_VALUE_ON_ZERO' */;
/*!40111 SET @OLD_SQL_NOTES=@@SQL_NOTES, SQL_NOTES=0 */;

--
-- Table structure for table `Appuntamento`
--

DROP TABLE IF EXISTS `Appuntamento`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!50503 SET character_set_client = utf8mb4 */;
CREATE TABLE `Appuntamento` (
  `idAppuntamento` varchar(20) GENERATED ALWAYS AS
(concat(`PropostaReale_idPropostaReale`,`Sala_NomeSala`)) STORED NOT NULL,
  `Giorno` datetime NOT NULL,
  `Sala_NomeSala` varchar(20) NOT NULL,
  `Sala_Sede_NomeSede` varchar(20) NOT NULL,
  `PropostaReale_idPropostaReale` int NOT NULL,
  `Nota` varchar(150) DEFAULT NULL,
  PRIMARY KEY (`idAppuntamento`),
  KEY `Appuntamento_sala_idx` (`Sala_NomeSala`,`Sala_Sede_NomeSede`),
  KEY `Appuntamento_propostaReale_idx` (`PropostaReale_idPropostaReale`),
  CONSTRAINT `fk_Appuntamento_PropostaReale1` FOREIGN KEY
(`PropostaReale_idPropostaReale`) REFERENCES `PropostaReale` (`idPropostaReale`),
```

```
  CONSTRAINT `fk_Appuntamento_Sala1` FOREIGN KEY (`Sala_NomeSala`,
`Sala_Sede_NomeSede`) REFERENCES `Sala` (`NomeSala`, `Sede_NomeSede`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
/*!40101 SET character_set_client = @saved_cs_client */;

--
-- Dumping data for table `Appuntamento`
--


--
-- Table structure for table `Cliente`
--

DROP TABLE IF EXISTS `Cliente`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!50503 SET character_set_client = utf8mb4 */;
CREATE TABLE `Cliente` (
  `CodiceCliente` int unsigned NOT NULL AUTO_INCREMENT,
  `Nome` varchar(30) NOT NULL,
  `Cognome` varchar(30) NOT NULL,
  `CodiceFiscale` varchar(17) NOT NULL,
  `PartitaIva` varchar(11) DEFAULT NULL,
  `DataRegistrazione` date NOT NULL,
  `DataNascita` date NOT NULL,
  `Telefono` varchar(11) NOT NULL,
  `Mail` varchar(50) NOT NULL,
  `Fax` varchar(11) DEFAULT NULL,
  `Citta` varchar(30) NOT NULL,
  `Indirizzo` varchar(50) NOT NULL,
  PRIMARY KEY (`CodiceCliente`)
) ENGINE=InnoDB AUTO_INCREMENT=24 DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_0900_ai_ci;
/*!40101 SET character_set_client = @saved_cs_client */;


--
-- Table structure for table `Funzionario`
--

DROP TABLE IF EXISTS `Funzionario`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!50503 SET character_set_client = utf8mb4 */;
CREATE TABLE `Funzionario` (
  `idFunzionario` int unsigned NOT NULL AUTO_INCREMENT,
  `Nome` varchar(30) NOT NULL,
  `Cognome` varchar(30) NOT NULL,
  PRIMARY KEY (`idFunzionario`)
) ENGINE=InnoDB AUTO_INCREMENT=8 DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_0900_ai_ci;
/*!40101 SET character_set_client = @saved_cs_client */;
```

```
--
-- Table structure for table `Nota`
--

DROP TABLE IF EXISTS `Nota`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!50503 SET character_set_client = utf8mb4 */;
CREATE TABLE `Nota` (
  `Tipo` enum('Costo','Assistenza','Servizio','Altro') NOT NULL,
  `PropostaReale_idPropostaReale` int NOT NULL,
  `Descrizione` varchar(150) NOT NULL,
  PRIMARY KEY (`Tipo`,`PropostaReale_idPropostaReale`),
  KEY `Nota_PropostaReale_idx` (`PropostaReale_idPropostaReale`),
  CONSTRAINT `fk_Nota_PropostaReale1` FOREIGN KEY
(`PropostaReale_idPropostaReale`) REFERENCES `PropostaReale` (`idPropostaReale`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
/*!40101 SET character_set_client = @saved_cs_client */;
--
-- Table structure for table `Proposta`
--
DROP TABLE IF EXISTS `Proposta`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!50503 SET character_set_client = utf8mb4 */;
CREATE TABLE `Proposta` (
  `Codice` varchar(15) NOT NULL,
  `Scadenza` date NOT NULL,
  `Descrizione` varchar(150) NOT NULL,
  PRIMARY KEY (`Codice`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
/*!40101 SET character_set_client = @saved_cs_client */;

--
-- Table structure for table `PropostaReale`
--
DROP TABLE IF EXISTS `PropostaReale`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!50503 SET character_set_client = utf8mb4 */;
CREATE TABLE `PropostaReale` (
  `idPropostaReale` int NOT NULL AUTO_INCREMENT,
  `Giorno` date NOT NULL,
  `Esito` enum('Positivo','Negativo','Accettato','Altro') DEFAULT NULL,
  `Cliente_CodiceCliente` int unsigned NOT NULL,
  `Proposta_Codice` varchar(15) NOT NULL,
  `Funzionario_idFunzionario` int unsigned NOT NULL,
  PRIMARY KEY (`idPropostaReale`),
  KEY `PropostaReale_Cliente_idx` (`Cliente_CodiceCliente`),
  KEY `fk_PropostaReale_Proposta1_idx` (`Proposta_Codice`),
  KEY `fk_PropostaReale_Funzionario1_idx` (`Funzionario_idFunzionario`),
```

```
  CONSTRAINT `fk_PropostaReale_Cliente1` FOREIGN KEY (`Cliente_CodiceCliente`)
REFERENCES `Cliente` (`CodiceCliente`),
  CONSTRAINT `fk_PropostaReale_Funzionario1` FOREIGN KEY
(`Funzionario_idFunzionario`) REFERENCES `Funzionario` (`idFunzionario`),
  CONSTRAINT `fk_PropostaReale_Proposta1` FOREIGN KEY (`Proposta_Codice`)
REFERENCES `Proposta` (`Codice`)
) ENGINE=InnoDB AUTO_INCREMENT=9 DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_0900_ai_ci;
/*!40101 SET character_set_client = @saved_cs_client */;

--
-- Table structure for table `Sala`
--
DROP TABLE IF EXISTS `Sala`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!50503 SET character_set_client = utf8mb4 */;
CREATE TABLE `Sala` (
  `NomeSala` varchar(20) NOT NULL,
  `Sede_NomeSede` varchar(20) NOT NULL,
  PRIMARY KEY (`NomeSala`,`Sede_NomeSede`),
  KEY `Sala_sede_idx` (`Sede_NomeSede`),
  CONSTRAINT `fk_Sala_Sede1` FOREIGN KEY (`Sede_NomeSede`) REFERENCES `Sede`
(`NomeSede`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
/*!40101 SET character_set_client = @saved_cs_client */;

--
-- Table structure for table `Sede`
--
DROP TABLE IF EXISTS `Sede`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!50503 SET character_set_client = utf8mb4 */;
CREATE TABLE `Sede` (
  `NomeSede` varchar(20) NOT NULL,
  `Indirizzo` varchar(45) NOT NULL,
  `Citta` varchar(45) NOT NULL,
  PRIMARY KEY (`NomeSede`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
/*!40101 SET character_set_client = @saved_cs_client */;
--
-- Table structure for table `Utenti`
--
DROP TABLE IF EXISTS `Utenti`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!50503 SET character_set_client = utf8mb4 */;
CREATE TABLE `Utenti` (
  `Username` varchar(30) NOT NULL,
  `Password` varchar(32) NOT NULL,
  `Ruolo` enum('Manager','Funzionario','Addetto') NOT NULL,
  PRIMARY KEY (`Username`)
```

```
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
/*!40101 SET character_set_client = @saved_cs_client */;
--
-- Dumping events for database 'CRM-db'
--
--
-- Dumping routines for database 'CRM-db'
--

DROP USER IF EXISTS login;
CREATE USER 'login' IDENTIFIED BY 'login';
GRANT EXECUTE ON procedure `CRM-db`.`login` TO 'login';

DROP USER IF EXISTS funzionario;
CREATE USER 'funzionario' IDENTIFIED BY 'funzionario';
GRANT EXECUTE ON procedure `CRM-db`.`calendario` TO 'funzionario';
GRANT EXECUTE ON procedure `CRM-db`.`comunicaProposta` TO 'funzionario';
GRANT EXECUTE ON procedure `CRM-db`.`esitoAppuntamento` TO 'funzionario';
GRANT EXECUTE ON procedure `CRM-db`.`gestioneNota` TO 'funzionario';
GRANT EXECUTE ON procedure `CRM-db`.`inserisciAppuntamento` TO 'funzionario';
GRANT EXECUTE ON procedure `CRM-db`.`listaClienti` TO 'funzionario';
GRANT EXECUTE ON procedure `CRM-db`.`listaCodiciAppuntamenti` TO 'funzionario';
GRANT EXECUTE ON procedure `CRM-db`.`listaComunicazioni` TO 'funzionario';
GRANT EXECUTE ON procedure `CRM-db`.`listaNoteCliente` TO 'funzionario';
GRANT EXECUTE ON procedure `CRM-db`.`listaSedi` TO 'funzionario';
GRANT EXECUTE ON procedure `CRM-db`.`ProposteAccettateSingolo` TO 'funzionario';
GRANT EXECUTE ON procedure `CRM-db`.`visualizzaAppuntamento` TO 'funzionario';
GRANT EXECUTE ON procedure `CRM-db`.`visualizzaClienteSingolo` TO 'funzionario';
GRANT EXECUTE ON procedure `CRM-db`.`visualizzaClientePropostePendenti` TO
'funzionario';
GRANT EXECUTE ON procedure `CRM-db`.`visualizzaClientiFunzionario` TO 'funzionario';
GRANT EXECUTE ON procedure `CRM-db`.`visualizzaNota` TO 'funzionario';

DROP USER IF EXISTS manager;
CREATE USER 'manager' IDENTIFIED BY 'manager';
GRANT EXECUTE ON procedure `CRM-db`.`inserisciProposta` TO 'manager';
GRANT EXECUTE ON procedure `CRM-db`.`inserisciFunzionario` TO 'manager';

DROP USER IF EXISTS addetto;
CREATE USER 'addetto' IDENTIFIED BY 'addetto';
GRANT EXECUTE ON procedure `CRM-db`.`inserisciCliente` TO 'addetto';

/*!50003 SET sql_mode              = @saved_sql_mode */ ;
/*!50003 SET character_set_client  = @saved_cs_client */ ;
/*!50003 SET character_set_results = @saved_cs_results */ ;
/*!50003 SET collation_connection  = @saved_col_connection */ ;
/*!40103 SET TIME_ZONE=@OLD_TIME_ZONE */;

/*!40101 SET SQL_MODE=@OLD_SQL_MODE */;
/*!40014 SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS */;
```

```
/*!40014 SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS */;
/*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;
/*!40101 SET CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS */;
/*!40101 SET COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION */;
/*!40111 SET SQL_NOTES=@OLD_SQL_NOTES */;
```

Codice del Front-End

Per la compilazione ed esecuzione del client è necessario:

- Installare il mysql server mediante “sudo apt install mysql-server”.
- Se non presente, installare la libreria <mysql.h> mediante “apt-get install libmysqlclient-dev”

Per eseguire il client è necessario poi istanziare la base dati:

- In MySQLWorkbench, recarsi in “file” >> “Open MySQL Script..”, selezionare “CRM-db.sql”, ed eseguirlo premendo l’icona del fulmine 

Sono presenti già alcune tuple all’interno del database.

Main.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <mysql.h>
#include "defines.h"

typedef enum {
        FUNZIONARIO = 1,
        MANAGER,
        ADDETTO,
        FAILED_LOGIN
} role_t;


struct configuration conf;

static MYSQL *conn;


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

        MYSQL_BIND param[3]; // Used both for input and output
        int role = 0;

        if(!setup_prepared_stmt(&login_procedure, "call login(?, ?, ?)", conn)) {
                print_stmt_error(login_procedure, "Unable to initialize login statement\n");
                goto err2;
        }

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

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

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

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

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

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

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

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

        if(mysql_stmt_bind_result(login_procedure, param)) {
                print_stmt_error(login_procedure, "Could not retrieve output parameter");
                goto err;
        }

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

        mysql_stmt_close(login_procedure);
        return role;

err:
        mysql_stmt_close(login_procedure);
err2:
        return FAILED_LOGIN;
}

int main(void) {
        role_t role;

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

        conn = mysql_init (NULL);
        if (conn == NULL) {
                fprintf (stderr, "mysql_init() failed (probably out of memory)\n");
                exit(EXIT_FAILURE);
        }
```

```
	//test

	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, "Failed to connect to database: Error: %s\n",
	  mysql_error(conn));
		fprintf (stderr, "mysql_real_connect() failed\n");
		mysql_close (conn);
		exit(EXIT_FAILURE);
	}

	printf("\033[2J\033[H");
	printf("**************************************************\n");
	printf("***            >>  WELCOME TO CRM-db  <<            ***\n");
	printf("**************************************************\n");

	printf("\n");
	printf("Username: ");
	getInput(128, conf.username, false);
	printf("Password: ");
	getInput(128, conf.password, true);

	role = attempt_login(conn, conf.username, conf.password);

	switch(role) {
		case FUNZIONARIO:
			run_funzionario(conn);
			break;

		case ADDETTO:
			run_addetto(conn);
			break;

		case MANAGER:
			run_manager(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("Bye!\n");

	mysql_close (conn);
	return 0;
}
```

Addetto.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "defines.h"

static void inserisciCliente(MYSQL *conn){

        MYSQL_STMT *prepared_stmt;
        MYSQL_BIND param[12];

        char nome[30];
        char cognome[30];
        char codiceFiscale[17];
        char partitaIva[11];


        char giorno[3];
        char mese[3];
        char anno[5];

        char telefono[11];
        char mail[50];
    char fax[11];
    char citta[30];
        char indirizzo[50];

        MYSQL_TIME nascita;
        memset(&nascita,0, sizeof(nascita));

        printf("\n>> Inserimento nuovo utente << \n");
        printf("\n* indica un campo obbligatorio.\nPer gli campi è possibile premere 'invio' e proseguire. \n");
        // Get the required information

        printf("\n >>Dati personali<<\n\n");
        printf("\n* Nome: ");
        getInput(30, nome, false);
        printf("* Cognome: ");
        getInput(30, cognome, false);
        printf("* Codice Fiscale: ");
        getInput(17, codiceFiscale, false);
        printf(" partita Iva: ");
        getInput(11, partitaIva, false);

        printf("\n >>Data di Nascita<<\n\n");
        printf("* Giorno: ");
        getInput(3, giorno, false);
        printf("* Mese (numerico): ");
        getInput(3, mese, false);
        printf("* Anno: ");
        getInput(5, anno, false);
        printf("\n");

        printf("\n >>Contatti<<\n\n");
        printf("* Mail: ");
        getInput(50, mail, false);
```

```
            printf("* Telefono: ");
            getInput(11, telefono, false);
            printf(" Fax: ");
            getInput(11, fax, false);

            printf("\n >>Residenza<<\n\n");
            printf("* Citta: ");
            getInput(30, citta, false);
            printf("* Indirizzo:");
            getInput(50, indirizzo, false);

            nascita.year = atoi(anno);
            nascita.month = atoi(mese);
            nascita.day = atoi(giorno);


            // Prepare stored procedure call
            if(!setup_prepared_stmt(&prepared_stmt, "call inserisciCliente(?, ?, ?, ?, ?, ?, ?, ?, ?, ?)", conn)) {
                        finish_with_stmt_error(conn, prepared_stmt, "Errore nella preparazione della stored procedure
'inserisci Cliente'.\n", false);
            }

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


            param[0].buffer_type = MYSQL_TYPE_VAR_STRING;
            param[0].buffer = nome;
            param[0].buffer_length = strlen(nome);

            param[1].buffer_type = MYSQL_TYPE_VAR_STRING;
            param[1].buffer = cognome;
            param[1].buffer_length = strlen(cognome);

            param[2].buffer_type = MYSQL_TYPE_VAR_STRING;
            param[2].buffer = codiceFiscale;
            param[2].buffer_length = strlen(codiceFiscale);

            param[3].buffer_type = MYSQL_TYPE_VAR_STRING;
            param[3].buffer = partitaIva;
            param[3].buffer_length = strlen(partitaIva);

            param[4].buffer_type = MYSQL_TYPE_DATE;
            param[4].buffer = &nascita;
            param[4].buffer_length = sizeof(nascita);

            //new insert
            param[5].buffer_type = MYSQL_TYPE_VAR_STRING;
            param[5].buffer = telefono;
            param[5].buffer_length = strlen(telefono);

            param[6].buffer_type = MYSQL_TYPE_VAR_STRING;
            param[6].buffer = mail;
            param[6].buffer_length = strlen(mail);

            //new insert
```

```
		param[7].buffer_type = MYSQL_TYPE_VAR_STRING;
		param[7].buffer = fax;
		param[7].buffer_length = strlen(fax);


		param[8].buffer_type = MYSQL_TYPE_VAR_STRING;
		param[8].buffer = citta;
		param[8].buffer_length = strlen(citta);

		param[9].buffer_type = MYSQL_TYPE_VAR_STRING;
		param[9].buffer = indirizzo;
		param[9].buffer_length = strlen(indirizzo);

		if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
			finish_with_stmt_error(conn, prepared_stmt, "Errore di binding param per l'utente addetto.\n",
true);
		}

		// Run procedure
		if (mysql_stmt_execute(prepared_stmt) != 0) {
			print_stmt_error (prepared_stmt, "Errore nell'inserimento del nuovo cliente.");
		} else{
			printf("\nNuovo cliente inserito con successo.\nPremi 'invio' per tornare al menu.");
		}

		mysql_stmt_close(prepared_stmt);

}



void run_addetto(MYSQL *conn)
{
		char options[2] = {'1','0'};
		char op;

		printf("Eseguo come addetto...\n");

		if(!parse_config("users/addetto.json", &conf)) {
			fprintf(stderr, "Errore nel parsing per l'utente 'addetto'\n");
			exit(EXIT_FAILURE);
		}

		if(mysql_change_user(conn, conf.db_username, conf.db_password, conf.database)) {
			fprintf(stderr, "mysql_change_user() failed\n");
			exit(EXIT_FAILURE);
		}

		while(true) {
			printf("\033[2J\033[H");
			printf(">> Lista operazioni disponibili <<\n\n");
			printf("1) Registra un nuovo cliente\n");
			printf("0) Esci\n");

			op = multiChoice("Seleziona operazione:", options, 2);

			switch(op) {
```

```
                                case '1':
                                        inserisciCliente(conn);
                                        break;

                                case '0':
                                        return;

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

                        getchar();
                }
}
```

Manager.c

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

#include "defines.h"

//Con questa funzione creo una nuova proposta
static void creaProposta(MYSQL *conn) {
        MYSQL_STMT *prepared_stmt;
        MYSQL_BIND param[3];

        char codice[15];
        char descrizione[150];

        char giorno[3];
        char mese[3];
        char anno[5];

        MYSQL_TIME scadenza;
        memset(&scadenza,0, sizeof(scadenza));


        printf("\n>> Inserimento nuova proposta <<\n\n");
        printf("\nNome Proposta: ");
        getInput(10, codice, false);
        printf("Descrizione: ");
        getInput(150, descrizione, false);


        printf("\n >> Data di scadenza <<\n\n");
        printf("* Giorno: ");
        getInput(3, giorno, false);
        printf("* Mese (numerico): ");
        getInput(3, mese, false);
```

```
		printf("* Anno: ");
		getInput(5, anno, false);
		printf("\n");

		scadenza.year = atoi(anno);
		scadenza.month = atoi(mese);
		scadenza.day = atoi(giorno);


		if(!setup_prepared_stmt(&prepared_stmt, "call inserisciProposta(?, ?, ?)", conn)) {
			finish_with_stmt_error(conn, prepared_stmt, "Errore nella preparazione della stored procedure 'Crea Proposta'.\n", false);
		}

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

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

		param[1].buffer_type = MYSQL_TYPE_DATE;
		param[1].buffer = &scadenza;
		param[1].buffer_length = sizeof(scadenza);


		param[2].buffer_type = MYSQL_TYPE_VAR_STRING;
		param[2].buffer = descrizione;
		param[2].buffer_length = strlen(descrizione);




		if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
			finish_with_stmt_error(conn, prepared_stmt, "Errore di binding param per 'CreaProposta'\n", true);
		}

		// Run procedure
		if (mysql_stmt_execute(prepared_stmt) != 0) {
			print_stmt_error(prepared_stmt, "Errore nella creazione di una nuova proposta.Premi 'invio' per tornare al menu.\n");
		}else{
			printf("\nNuova proposta aggiunta correttamente!\nPremi 'invio' per tornare al menu.");
		}

		mysql_stmt_close(prepared_stmt);

}

static void aggiungiFunzionario(MYSQL *conn)
{
		MYSQL_STMT *prepared_stmt;
		MYSQL_BIND param[5];

		// Input for the registration routine
```

```
	char nome[30];
	char cognome[30];


	// Get the required information

	printf("\n >> Aggiungi funzionario <<\n\n");
	printf("Nome: ");
	getInput(30, nome, false);
	printf("Cognome: ");
	getInput(30, cognome, false);

	// Prepare stored procedure call
	if(!setup_prepared_stmt(&prepared_stmt, "call inserisciFunzionario(?, ?)", conn)) {
		finish_with_stmt_error(conn, prepared_stmt, "Errore nella preparazione della stored procedure
'aggiungiFunzionario'\n", false);
	}

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


	param[0].buffer_type = MYSQL_TYPE_VAR_STRING;
	param[0].buffer = nome;
	param[0].buffer_length = strlen(nome);

	param[1].buffer_type = MYSQL_TYPE_VAR_STRING;
	param[1].buffer = cognome;
	param[1].buffer_length = strlen(cognome);


	if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
		finish_with_stmt_error(conn, prepared_stmt, "Errore di binding param per
'aggiungiFunzionario'\n", true);
	}

	// Run procedure
	if (mysql_stmt_execute(prepared_stmt) != 0) {
		print_stmt_error (prepared_stmt, "Errore nell'aggiunta di un nuovo funzionario\n");
	} else {
		printf("\nNuovo funzionario aggiunto.\nPremi 'invio' per tornare al menu.");
	}

	mysql_stmt_close(prepared_stmt);
}


void run_manager(MYSQL *conn)
{
	char options[3] = {'1','2', '0'};
	char op;

	printf("Eseguo come 'Manager'...\n");

	if(!parse_config("users/manager.json", &conf)) {
		fprintf(stderr, "Errore nel parsing per l'utente 'manager'\n");
		exit(EXIT_FAILURE);
```

```
        }

        if(mysql_change_user(conn, conf.db_username, conf.db_password, conf.database)) {
                fprintf(stderr, "mysql_change_user() failed\n");
                exit(EXIT_FAILURE);
        }

        while(true) {
                printf("\033[2J\033[H");
                printf(">> Lista operazioni disponibili <<\n\n");
                printf("1) Crea una nuova proposta.\n");
                printf("2) Aggiungi funzionario. \n");
                printf("0) Esci\n");

                op = multiChoice("Seleziona operazione:", options, 3);

                switch(op) {
                        case '1':
                                creaProposta(conn);
                                break;

                        case '2':
                                aggiungiFunzionario(conn);
                                break;

                        case '0':
                                return;

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

                getchar();
        }
}
```

Funzionario.C

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

#include "defines.h"

static void visualizzaClientiFunzionario(MYSQL *conn){
		MYSQL_STMT *prepared_stmt;
		MYSQL_BIND param[1];

		char idFunzionario [15] ;

		int idf;
		printf("\n >> Visualizza clienti associati ad un funzionario <<\n\n");
		printf("\nInserire matricola funzionario da visionare: ");
		getInput(17, idFunzionario, false);

		idf = atoi(idFunzionario);


		if(!setup_prepared_stmt(&prepared_stmt, "call visualizzaClientiFunzionario(?)", conn)) {
			finish_with_stmt_error(conn, prepared_stmt, "Errore nella preparazione della stored procedure
'VisualizzaClientiFunzionario'.\n", false);
		}

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

		param[0].buffer_type = MYSQL_TYPE_LONG;
		param[0].buffer = &idf;
		param[0].buffer_length = sizeof(idf);


		if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
			finish_with_stmt_error(conn, prepared_stmt, "Errore di binding param per
'VisualizzaClientiFunzionario'\n", true);
		}

		// Run procedure
		if (mysql_stmt_execute(prepared_stmt) != 0) {
			print_stmt_error(prepared_stmt, "Errore nella visualizzazione dei clienti.\n");
		}

		// Dump the result set
		dump_result_set(conn, prepared_stmt, "\nLista dei clienti.");
		mysql_stmt_next_result(prepared_stmt);
		mysql_stmt_close(prepared_stmt);



}
```

```
static void visualizzaClienteSingolo(MYSQL *conn){

		MYSQL_STMT *prepared_stmt;
		MYSQL_BIND param[2];
		char nome[30];
		char cognome[30];

		printf("\n >> Dati cliente <<\n\n");
		printf("\nInserire nome: ");
		getInput(30, nome, false);

		printf("\nInserire cognome: ");
		getInput(30, cognome, false);


			if(!setup_prepared_stmt(&prepared_stmt, "call visualizzaClienteSingolo(?,?)", conn)) {
			finish_with_stmt_error(conn, prepared_stmt, "Errore nella preparazione della stored procedure
'visualizzaClienteSingolo'\n", false);
		}

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

		param[0].buffer_type = MYSQL_TYPE_VAR_STRING;
		param[0].buffer = nome;
		param[0].buffer_length = strlen(nome);

		param[1].buffer_type = MYSQL_TYPE_VAR_STRING;
		param[1].buffer = cognome;
		param[1].buffer_length = strlen(cognome);


		if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
			finish_with_stmt_error(conn, prepared_stmt, "Errore di binding param per
'visualizzaClienteSingolo'\n", true);
		}


		// Run procedure
		if (mysql_stmt_execute(prepared_stmt) != 0) {
			print_stmt_error(prepared_stmt, "Errore nell'esecuzione di 'visualizzaClienteSingolo'\n");
		}

		// Dump the result set
		dump_result_set(conn, prepared_stmt, "\nDati Personali");
		mysql_stmt_next_result(prepared_stmt);
		mysql_stmt_close(prepared_stmt);


		/////// PROPOSTE ACCETTATE


		if(!setup_prepared_stmt(&prepared_stmt, "call proposteAccettateSingolo(?,?)", conn)) {
			finish_with_stmt_error(conn, prepared_stmt, "Errore inizializzazione proposte\n", false);
		}
```

```
	if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
		finish_with_stmt_error(conn, prepared_stmt, "errore binding proposte accettate.\n", true);
	}


	// Run procedure
	if (mysql_stmt_execute(prepared_stmt) != 0) {
		print_stmt_error(prepared_stmt, "Errore nell'esecuzione di 'proposteAccettate'\n");
	}

	// Dump the result set
	dump_result_set(conn, prepared_stmt, "\nProposte Accettate\n");
	mysql_stmt_next_result(prepared_stmt);


	printf("\n");


	///////NOTE CLIENTE

		if(!setup_prepared_stmt(&prepared_stmt, "call listaNoteCliente(?,?)", conn)) {
		finish_with_stmt_error(conn, prepared_stmt, "Errore inizializzazione note\n", false);
	}
	if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
		finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters for company list\n",
true);
	}


	// Run procedure
	if (mysql_stmt_execute(prepared_stmt) != 0) {
		print_stmt_error(prepared_stmt, "Errore nell'esecuzione di 'listaNoteCliente'\n");
	}

	// Dump the result set
	dump_result_set(conn, prepared_stmt, "\nNote rilasciate dal cliente\n");
	mysql_stmt_next_result(prepared_stmt);


	printf("\n");




}

static void visualizzaNota(MYSQL *conn){

	MYSQL_STMT *prepared_stmt;
	MYSQL_BIND param[1];
	char codice[15];

	printf("\n >> Visualizza note associate ad una proposta <<\n\n");
	printf("\nNome proposta: ");
	getInput(15, codice, false);
```

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

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



		if(!setup_prepared_stmt(&prepared_stmt, "call visualizzaNota(?)", conn)) {
			finish_with_stmt_error(conn, prepared_stmt, "Errore nella preparazione della stored procedure
'visualizza Nota'\n", false);
		}

		if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
			finish_with_stmt_error(conn, prepared_stmt, "Errore di binding param per 'Visualizza Nota'\n",
true);
		}

		// Run procedure
		if (mysql_stmt_execute(prepared_stmt) != 0) {
			print_stmt_error(prepared_stmt, "Errore nell'esecuzione di 'Visualizza Nota'\n");
		}

		// Dump the result set
		dump_result_set(conn, prepared_stmt, "\nLista delle note\n");
		mysql_stmt_next_result(prepared_stmt);
		mysql_stmt_close(prepared_stmt);



}

static void visualizzaAppuntamento(MYSQL *conn){

		MYSQL_STMT *prepared_stmt;
		MYSQL_BIND param[1];
		char anno[5];
		char mese[3];
		char giorno[3];

		MYSQL_TIME data;
		memset(&data,0, sizeof(data));

		printf("\n >> Visione appuntamenti << \n\n");
		printf("Inserire giorno: ");
		getInput(3, giorno, false);
		printf("Inserire mese: ");
		getInput(3, mese, false);
		printf("Inserire anno: ");
		getInput(5, anno, false);

		data.day=atoi(giorno);
		data.month=atoi(mese);
		data.year=atoi(anno);
```

```
		if(!setup_prepared_stmt(&prepared_stmt, "call visualizzaAppuntamento(?)", conn)) {
			finish_with_stmt_error(conn, prepared_stmt, "Errore nella preparazione della stored procedure 'visualizzaAppuntamento'\n", false);
		}


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

		param[0].buffer_type = MYSQL_TYPE_DATETIME;
		param[0].buffer =&data;
		param[0].buffer_length = sizeof(data);



		if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
			finish_with_stmt_error(conn, prepared_stmt, "Errore di binding param per 'VisualizzaAppuntamento' \n", true);
		}

		// Run procedure
		if (mysql_stmt_execute(prepared_stmt) != 0) {
			print_stmt_error(prepared_stmt, "Errore nell'esecuzione di 'VisualizzaAppuntamento'\n");
		}

		// Dump the result set
		dump_result_set(conn, prepared_stmt, "\nLista degli appuntamenti\n");
		mysql_stmt_next_result(prepared_stmt);
		mysql_stmt_close(prepared_stmt);


}

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

		char propostareale[20];
		char tipo[12];
		char descrizione[150];

		printf("\n >> Gestione delle note << \n\n");


		if(!setup_prepared_stmt(&prepared_stmt, "call listaComunicazioni()", conn)) {
			finish_with_stmt_error(conn, prepared_stmt, "Errore inizializzazione lista comunicazioni\n", false);
		}


		// Run procedure
		if (mysql_stmt_execute(prepared_stmt) != 0) {
			print_stmt_error(prepared_stmt, "Errore nell'esecuzione di 'listaComunicazioni'\n");
		}

		// Dump the result set
		dump_result_set(conn, prepared_stmt, "\nLista codici\n");
```

```
	mysql_stmt_next_result(prepared_stmt);


	printf("\n");




	printf("\nProposta Reale associata: ");
	getInput(20, propostareale, false);
	printf("Definisci il tipo ('Servizio','Costo','Assistenza','Altro'): ");
	getInput(12, tipo, false);
	printf("Inserisci la nota: ");
	getInput(150, descrizione, false);



	if(!setup_prepared_stmt(&prepared_stmt, "call gestioneNota(?, ?, ?)", conn)) {
		finish_with_stmt_error(conn, prepared_stmt, "Errore nella preparazione della stored procedure
'gestioneNota'\n", false);
	}
	memset(param, 0, sizeof(param));

	param[0].buffer_type = MYSQL_TYPE_VAR_STRING;
	param[0].buffer = propostareale;
	param[0].buffer_length = strlen(propostareale);

	param[1].buffer_type = MYSQL_TYPE_VAR_STRING;
	param[1].buffer = tipo;
	param[1].buffer_length = strlen(tipo);

	param[2].buffer_type = MYSQL_TYPE_VAR_STRING;
	param[2].buffer = descrizione;
	param[2].buffer_length = strlen(descrizione);


	if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
		finish_with_stmt_error(conn, prepared_stmt, "Errore di binding param per 'gestioneNota'\n",
true);
	}

	// Run procedure
	if (mysql_stmt_execute(prepared_stmt) != 0) {
		print_stmt_error(prepared_stmt, "Errore nella gestione della nota\n");
	}else{
		printf("\nNota inserita correttamente.\n");
	}


	mysql_stmt_close(prepared_stmt);
}


static void inserisciAppuntamento(MYSQL *conn){
	printf("\033[2J\033[H");
	MYSQL_STMT *prepared_stmt;
```

```
		MYSQL_BIND param[5];

		char anno[5];
		char mese[3];
		char giorno[3];
		char ora[3];
		char minuti[3];

		char sala[20];
		char sede[20];
		char propostareale[20];
		char nota[150];

		printf("\n >> Creazione appuntamento << \n");


		//LISTA PROPOSTE POSITIVE
		if(!setup_prepared_stmt(&prepared_stmt, "call visualizzaClientePropostePendenti()", conn)) {
		finish_with_stmt_error(conn, prepared_stmt, "Errore nella preparazione della stored procedure
'visualizzaClientiPropostePendenti'\n", false);
		}



		if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
				finish_with_stmt_error(conn, prepared_stmt, "Errore di binding param per
'visualizzaClientiPropostePendenti'\n", true);
		}


		// Run procedure
		if (mysql_stmt_execute(prepared_stmt) != 0) {
				print_stmt_error(prepared_stmt, "Errore nell'esecuzione di
'visualizzaClientiPropostePendenti'\n");
		}

		// Dump the result set
		dump_result_set(conn, prepared_stmt, "\nLista dei clienti con esito 'Positivo'.");
		mysql_stmt_next_result(prepared_stmt);






		MYSQL_TIME data,orario;
		memset(&data,0, sizeof(data));
		memset(&orario,0, sizeof(orario));




		if(!setup_prepared_stmt(&prepared_stmt, "call calendario()", conn)) {
				finish_with_stmt_error(conn, prepared_stmt, "Errore inizializzazione calendario\n", false);
		}
```

```
// Run procedure
if (mysql_stmt_execute(prepared_stmt) != 0) {
        print_stmt_error(prepared_stmt, "Errore nell'esecuzione di 'VisualizzaAppuntamento'\n");
}

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


printf("\n");

//LISTA SEDI

if(!setup_prepared_stmt(&prepared_stmt, "call listaSedi()", conn)) {
        finish_with_stmt_error(conn, prepared_stmt, "Errore inizializzazione sedi\n", false);
}


// Run procedure
if (mysql_stmt_execute(prepared_stmt) != 0) {
        print_stmt_error(prepared_stmt, "Errore nell'esecuzione di 'VisualizzaAppuntamento'\n");
}

// Dump the result set
dump_result_set(conn, prepared_stmt, "\nLista delle sedi.");
mysql_stmt_next_result(prepared_stmt);


printf("\n >> Inserimento appuntamento << \n\n");


printf("Sala: ");
getInput(20, sala, false);
printf("Sede: ");
getInput(20, sede, false);
printf("Proposta Reale associata: ");
getInput(10, propostareale, false);

printf("\n Selezione data\n");

printf("Giorno: ");
getInput(3, giorno, false);
printf("Mese: ");
getInput(3, mese, false);
printf("Anno: ");
getInput(5, anno, false);
printf("Ora: ");
getInput(3, ora, false);
printf("Minuti: ");
getInput(3, minuti, false);

printf("Nota descrittiva: ");
getInput(150, nota, false);
```

```
		data.day=atoi(giorno);
		data.month=atoi(mese);
		data.year=atoi(anno);
		data.hour = atoi(ora);
		data.minute = atoi(minuti);




		if(!setup_prepared_stmt(&prepared_stmt, "call inserisciAppuntamento(?, ?, ?, ?, ?)", conn)) {
			finish_with_stmt_error(conn, prepared_stmt, "Errore nella preparazione della stored procedure
'inserisciAppuntamento'\n", false);
		}

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

		param[0].buffer_type = MYSQL_TYPE_DATETIME;
		param[0].buffer = &data;
		param[0].buffer_length = sizeof(data);


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

		param[2].buffer_type = MYSQL_TYPE_VAR_STRING;
		param[2].buffer =sede;
		param[2].buffer_length = strlen(sede);

		param[3].buffer_type = MYSQL_TYPE_VAR_STRING;
		param[3].buffer = propostareale;
		param[3].buffer_length = strlen(propostareale);

		param[4].buffer_type = MYSQL_TYPE_VAR_STRING;
		param[4].buffer = nota;
		param[4].buffer_length = strlen(nota);

		if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
			finish_with_stmt_error(conn, prepared_stmt, "Errore di binding param per
'inserisciAppuntamento'\n", true);
		}

		// Run procedure
		if (mysql_stmt_execute(prepared_stmt) != 0) {
			print_stmt_error(prepared_stmt, "Errore nell'esecuzione di 'inserisciAppuntamento'\n");
		}else{
			printf("Appuntamento registrato.\n");
		}


		mysql_stmt_close(prepared_stmt);

}
```

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

		char esito[20];
		char codiceproposta[15];
		char cliente[5];
		char funzionario[5];
		int  idcomunicazione = 0;

		printf("\n >> Comunicazione proposta ad un cliente << \n\n");


		if(!setup_prepared_stmt(&prepared_stmt, "call listaClienti()", conn)) {
			finish_with_stmt_error(conn, prepared_stmt, "Errore inizializzazione listaClienti\n", false);
		}


		// Run procedure
		if (mysql_stmt_execute(prepared_stmt) != 0) {
			print_stmt_error(prepared_stmt, "Errore nell'esecuzione di 'listaClienti'\n");
		}

		// Dump the result set
		dump_result_set(conn, prepared_stmt, "\nLista Clienti\n");
		mysql_stmt_next_result(prepared_stmt);

		printf("\n\nCodice Cliente: ");
		getInput(5, cliente, false);
		printf("Codice Funzionario: ");
		getInput(5, funzionario, false);
		printf("Codice della proposta: ");
		getInput(15, codiceproposta, false);
		printf("Esito ('Positivo','Negativo','Accettato','Altro'): ");
		getInput(20, esito, false);







		if(!setup_prepared_stmt(&prepared_stmt, "call comunicaProposta(?, ?, ?, ?,?)", conn)) {
			finish_with_stmt_error(conn, prepared_stmt, "Errore nella preparazione della stored procedure
'inserisciAppuntamento'\n", false);
		}

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

		param[0].buffer_type = MYSQL_TYPE_VAR_STRING;
		param[0].buffer = esito;
		param[0].buffer_length = strlen(esito);


		param[1].buffer_type = MYSQL_TYPE_VAR_STRING;
		param[1].buffer = codiceproposta;
```

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

        param[2].buffer_type = MYSQL_TYPE_VAR_STRING;
        param[2].buffer = cliente;
        param[2].buffer_length = strlen(cliente);

        param[3].buffer_type = MYSQL_TYPE_VAR_STRING;
        param[3].buffer = funzionario;
        param[3].buffer_length = strlen(funzionario);

        //OUT
        param[4].buffer_type = MYSQL_TYPE_LONG;
        param[4].buffer = &idcomunicazione;
        param[4].buffer_length = sizeof(idcomunicazione);


        if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
                finish_with_stmt_error(conn, prepared_stmt, "Errore di binding param per
'comunicaProposta'\n", true);
        }

        // Run procedure
        if (mysql_stmt_execute(prepared_stmt) != 0) {
                print_stmt_error(prepared_stmt, "Errore nell'esecuzione di 'comunicaProposta'\n");
        }else{
                printf("Proposta comunicata.\n");
        }

        memset(param, 0, sizeof(param));
        param[0].buffer_type = MYSQL_TYPE_LONG; // OUT
        param[0].buffer = &idcomunicazione;
        param[0].buffer_length = sizeof(idcomunicazione);

        if(mysql_stmt_bind_result(prepared_stmt, param)) {
           print_stmt_error(prepared_stmt, "Errore nel risultato in uscita.");
          }

        // Retrieve output parameter
        if(mysql_stmt_fetch(prepared_stmt)) {
            print_stmt_error(prepared_stmt, "Errore mysql_stmt_fetch");

         } else {
                 printf("\nId univoco della comunicazione: %d",idcomunicazione);
                }


        mysql_stmt_close(prepared_stmt);


}

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

        char esito[20];
        char appuntamento[20];
```

```
		printf("\n >> Esito Appuntamento << \n\n");

		//LISTA CODICI

		if(!setup_prepared_stmt(&prepared_stmt, "call listaCodiciAppuntamenti()", conn)) {
			finish_with_stmt_error(conn, prepared_stmt, "Errore inizializzazione lista codici\n", false);
		}


		// Run procedure
		if (mysql_stmt_execute(prepared_stmt) != 0) {
			print_stmt_error(prepared_stmt, "Errore nell'esecuzione di 'listaCodiciAppuntamenti'\n");
		}

		// Dump the result set
		dump_result_set(conn, prepared_stmt, "\nLista codici\n");
		mysql_stmt_next_result(prepared_stmt);


		printf("\n");



		printf("Codice dell'appuntamento: ");
		getInput(20, appuntamento, false);
		printf("Esito ('Accettato' o 'Negativo'): ");
		getInput(20, esito, false);

		if(!setup_prepared_stmt(&prepared_stmt, "call esitoAppuntamento(?, ?)", conn)) {
			finish_with_stmt_error(conn, prepared_stmt, "Errore nella preparazione della stored procedure
'esitoAppuntamento'\n", false);
		}

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

		param[0].buffer_type = MYSQL_TYPE_VAR_STRING;
		param[0].buffer = appuntamento;
		param[0].buffer_length = strlen(appuntamento);


		param[1].buffer_type = MYSQL_TYPE_VAR_STRING;
		param[1].buffer = esito;
		param[1].buffer_length = strlen(esito);

		if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
			finish_with_stmt_error(conn, prepared_stmt, "Errore di binding param per
'esitoAppuntamento'\n", true);
		}

		// Run procedure
		if (mysql_stmt_execute(prepared_stmt) != 0) {
			print_stmt_error(prepared_stmt, "Errore nell'esecuzione di 'esitoAppuntamento'\n");
		}else{
			printf("Esito appuntamento registrato.\n");
		}
```

```
        mysql_stmt_close(prepared_stmt);


}




void run_funzionario(MYSQL *conn)
{
        char options[9] = {'1','2', '3', '4', '5', '6','7','8','0'};
        char op;

        printf("Eseguo come funzionario...\n");

        if(!parse_config("users/funzionario.json", &conf)) {
                fprintf(stderr, "Errore nel parsing per l'utente 'funzionario'\n");
                exit(EXIT_FAILURE);
        }

        if(mysql_change_user(conn, conf.db_username, conf.db_password, conf.database)) {
                fprintf(stderr, "mysql_change_user() failed\n");
                exit(EXIT_FAILURE);
        }

        while(true) {
                printf("\033[2J\033[H");
                printf(">> Lista operazioni disponibili <<\n\n");

                printf("1) Lista clienti gestiti da un funzionario.\n");
                printf("2) Visualizza singolo cliente, note e proposte accettate.\n");
                printf("3) Visualizza le note di una proposta.\n");
                printf("4) Visualizza appuntamenti fissati.\n");
                printf("5) Gestisci le note dei clienti.\n");
                printf("6) Inserisci un appuntamento.\n");
                printf("7) Comunica una proposta ad un cliente.\n");
                printf("8) Aggiorna l'esito di un appuntamento.\n");


                printf("0) esci\n\n");

                op = multiChoice("Seleziona operazione:", options,9 );


                switch(op) {
                        case '1':
                                visualizzaClientiFunzionario(conn);
                                break;

                        case '2':
                                visualizzaClienteSingolo(conn);
                                break;

                        case '3':
```

```
                                visualizzaNota(conn);
                                break;

                        case '4':
                                visualizzaAppuntamento(conn);
                                break;

                        case '5':
                                gestioneNota(conn);
                                break;

                        case '6':
                                inserisciAppuntamento(conn);
                                break;

                        case '7':
                                comunicaPropostaReale(conn);
                                break;

                        case '8':
                                esitoAppuntamento(conn);
                                break;


                        case '0':
                                return;

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

                getchar();
        }
}
```

Inout.c

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

#include "defines.h"

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

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

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

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

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

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

```
                }
        }

        // Acquisisce da tastiera al più lung - 1 caratteri
        for(i = 0; i < lung; i++) {
                (void) fread(&c, sizeof(char), 1, stdin);
                if(c == '\n') {
                        stringa[i] = '\0';
                        break;
                } else
                        stringa[i] = c;

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

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

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

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

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

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

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

        return stringa;
}
```

```
// Per la gestione dei segnali
static void handler(int s) {
		signo = s;
}


bool yesOrNo(char *domanda, char yes, char no, bool predef, bool insensitive)
{

		// I caratteri 'yes' e 'no' devono essere minuscoli
		yes = tolower(yes);
		no = tolower(no);

		// Decide quale delle due lettere mostrare come predefinite
		char s, n;
		if(predef) {
			s = toupper(yes);
			n = no;
		} else {
			s = yes;
			n = toupper(no);
		}

		// Richiesta della risposta
		while(true) {
			// Mostra la domanda
			printf("%s [%c/%c]: ", domanda, s, n);

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

			// Controlla quale risposta è stata data
			if(c == '\0') { // getInput() non può restituire '\n'!
				return predef;
			} else if(c == yes) {
				return true;
			} else if(c == no) {
				return false;
			} else if(c == toupper(yes)) {
				if(predef || insensitive) return true;
			} else if(c == toupper(yes)) {
				if(!predef || insensitive) return false;
			}
		}
}

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

		// Genera la stringa delle possibilità
		char *possib = malloc(2 * num * sizeof(char));
		int i, j = 0;
		for(i = 0; i < num; i++) {
			possib[j++] = choices[i];
			possib[j++] = '/';
		}
```

```
		possib[j-1] = '\0'; // Per eliminare l'ultima '/'

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

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

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

Parse.c

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

#include "defines.h"

#define BUFF_SIZE 4096

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

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

enum jsmnerr {
		/* Not enough tokens were provided */
		JSMN_ERROR_NOMEM = -1,
		/* Invalid character inside JSON string */
		JSMN_ERROR_INVAL = -2,
```

```
		/* The string is not a full JSON packet, more bytes expected */
		JSMN_ERROR_PART = -3
};

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

/**
 * JSON parser. Contains an array of token blocks available. Also stores
 * the string being parsed now and current position in that string
 */
typedef struct {
		unsigned int pos; /* offset in the JSON string */
		unsigned int toknext; /* next token to allocate */
		int toksuper; /* superior token node, e.g parent object or array */
} jsmn_parser;

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

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

```
/**
 * Fills next available token with JSON primitive.
 */
static int jsmn_parse_primitive(jsmn_parser *parser, const char *js,
		size_t len, jsmntok_t *tokens, size_t num_tokens) {
	jsmntok_t *token;
	int start;

	start = parser->pos;

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

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

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

```
	int start = parser->pos;

	parser->pos++;

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

		/* Quote: end of string */
		if (c == '\"') {
			if (tokens == NULL) {
				return 0;
			}
			token = jsmn_alloc_token(parser, tokens, num_tokens);
			if (token == NULL) {
				parser->pos = start;
				return JSMN_ERROR_NOMEM;
			}
			jsmn_fill_token(token, JSMN_STRING, start+1, parser->pos);
#ifdef JSMN_PARENT_LINKS
			token->parent = parser->toksuper;
#endif
			return 0;
		}

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

```
                                        }
                                }
                        }
                        parser->pos = start;
                        return JSMN_ERROR_PART;
}

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

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

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

```
                                        }
                                        if (token->parent == -1) {
                                                if(token->type != type || parser->toksuper == -1) {
                                                        return JSMN_ERROR_INVAL;
                                                }
                                                break;
                                        }
                                        token = &tokens[token->parent];
                                }
#else
                                for (i = parser->toknext - 1; i >= 0; i--) {
                                        token = &tokens[i];
                                        if (token->start != -1 && token->end == -1) {
                                                if (token->type != type) {
                                                        return JSMN_ERROR_INVAL;
                                                }
                                                parser->toksuper = -1;
                                                token->end = parser->pos + 1;
                                                break;
                                        }
                                }
                                /* Error if unmatched closing bracket */
                                if (i == -1) return JSMN_ERROR_INVAL;
                                for (; i >= 0; i--) {
                                        token = &tokens[i];
                                        if (token->start != -1 && token->end == -1) {
                                                parser->toksuper = i;
                                                break;
                                        }
                                }
#endif
                                break;
                        case '\"':
                                r = jsmn_parse_string(parser, js, len, tokens, num_tokens);
                                if (r < 0) return r;
                                count++;
                                if (parser->toksuper != -1 && tokens != NULL)
                                        tokens[parser->toksuper].size++;
                                break;
                        case '\t' : case '\r' : case '\n' : case ' ':
                                break;
                        case ':':
                                parser->toksuper = parser->toknext - 1;
                                break;
                        case ',':
                                if (tokens != NULL && parser->toksuper != -1 &&
                                                tokens[parser->toksuper].type != JSMN_ARRAY
&&
                                                tokens[parser->toksuper].type !=
JSMN_OBJECT) {
#ifdef JSMN_PARENT_LINKS
                                        parser->toksuper = tokens[parser->toksuper].parent;
#else
                                        for (i = parser->toknext - 1; i >= 0; i--) {
                                                if (tokens[i].type == JSMN_ARRAY ||
tokens[i].type == JSMN_OBJECT) {
```

```
                                                                        if (tokens[i].start != -1 &&
tokens[i].end == -1) {
                                                                                parser->toksuper = i;
                                                                                break;
                                                                        }
                                                                }
                                                        }
#endif
                                                }
                                                break;
#ifdef JSMN_STRICT
                                /* In strict mode primitives are: numbers and booleans */
                                case '-': case '0': case '1' : case '2': case '3' : case '4':
                                case '5': case '6': case '7' : case '8': case '9':
                                case 't': case 'f': case 'n' :
                                        /* And they must not be keys of the object */
                                        if (tokens != NULL && parser->toksuper != -1) {
                                                jsmntok_t *t = &tokens[parser->toksuper];
                                                if (t->type == JSMN_OBJECT ||
                                                                (t->type == JSMN_STRING && t-
>size != 0)) {
                                                        return JSMN_ERROR_INVAL;
                                                }
                                        }
#else
                                /* In non-strict mode every unquoted value is a primitive */
                                default:
#endif
                                        r = jsmn_parse_primitive(parser, js, len, tokens, num_tokens);
                                        if (r < 0) return r;
                                        count++;
                                        if (parser->toksuper != -1 && tokens != NULL)
                                                tokens[parser->toksuper].size++;
                                        break;

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

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

                return count;
}

/**
 * Creates a new parser based over a given  buffer with an array of tokens
```

```
 * available.
 */
static void jsmn_init(jsmn_parser *parser) {
	parser->pos = 0;
	parser->toknext = 0;
	parser->toksuper = -1;
}

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

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

	fseek(f, 0, SEEK_END);
	size_t fsize = ftell(f);
	fseek(f, 0, SEEK_SET);  //same as rewind(f);

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

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

	config[fsize] = 0;
	return fsize;
}

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

	load_file(path);

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

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

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

Utils.c

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

#include "defines.h"

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

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

bool setup_prepared_stmt(MYSQL_STMT **stmt, char *statement, MYSQL *conn)
{
	bool update_length = true;

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

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

	mysql_stmt_attr_set(*stmt, STMT_ATTR_UPDATE_MAX_LENGTH, &update_length);

	return true;
}

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

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

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

	mysql_field_seek(res_set, 0);
	putchar('+');
```

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

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

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

		mysql_field_seek (res_set, 0);

		for (i = 0; i < mysql_num_fields (res_set); i++) {
			field = mysql_fetch_field (res_set);
			col_len = strlen(field->name);

			if (col_len < field->max_length)
				col_len = field->max_length;
			if (col_len < 4 && !IS_NOT_NULL(field->flags))
				col_len = 4; /* 4 = length of the word "NULL" */
			field->max_length = col_len; /* reset column info */
		}

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

		print_dashes(res_set);
}

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

		/* Prefetch the whole result set. This in conjunction with
		 * STMT_ATTR_UPDATE_MAX_LENGTH set in `setup_prepared_stmt`
		 * updates the result set metadata which are fetched in this
```

```
 * function, to allow to compute the actual max length of
 * the columns.
 */
if (mysql_stmt_store_result(stmt)) {
        fprintf(stderr, " mysql_stmt_execute(), 1 failed\n");
        fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
        exit(0);
}

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

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

        if((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_DATETIME:
                        case MYSQL_TYPE_TIME:
                                attr_size = sizeof(MYSQL_TIME);
                                break;
                        case MYSQL_TYPE_FLOAT:
                                attr_size = sizeof(float);
                                break;
                        case MYSQL_TYPE_DOUBLE:
                                attr_size = sizeof(double);
                                break;
                        case MYSQL_TYPE_TINY:
                                attr_size = sizeof(signed char);
                                break;
                        case MYSQL_TYPE_SHORT:
                        case MYSQL_TYPE_YEAR:
                                attr_size = sizeof(short int);
                                break;
                        case MYSQL_TYPE_LONG:
                        case MYSQL_TYPE_INT24:
```

```
                                        attr_size = sizeof(int);
                                        break;
                                case MYSQL_TYPE_LONGLONG:
                                        attr_size = sizeof(int);
                                        break;
                                default:
                                        attr_size = fields[i].max_length;
                                        break;
                        }

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

                        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_VAR_STRING:
                                        case MYSQL_TYPE_DATETIME:
                                                printf(" %-*s |", (int)fields[i].max_length,
(char*)rs_bind[i].buffer);

                                                break;

                                        case MYSQL_TYPE_DATE:
                                        case MYSQL_TYPE_TIMESTAMP:
                                                date = (MYSQL_TIME *)rs_bind[i].buffer;
                                                printf(" %d-%02d-%02d |", date->year, date-
>month, date->day);

                                                break;

                                        case MYSQL_TYPE_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);
        }
}
```