|  |  |
| --- | --- |
|  |  |

**Basi di Dati**

**Progetto A.A. 2021/2022**

TITOLO DEL PROGETTO

Matricola

Nome e Cognome

**Indice**

[1. Descrizione del Minimondo 3](#_Toc606296459)

[2. Analisi dei Requisiti 5](#_Toc1289394997)

[3. Progettazione concettuale 9](#_Toc2081466291)

[4. Progettazione logica 10](#_Toc2147004904)

[5. Progettazione fisica 12](#_Toc518560220)

[Appendice: Implementazione 14](#_Toc403811585)

Tutto il testo su sfondo grigio, all’interno di questo template, deve essere eliminato prima della consegna. Viene utilizzato per fornire informazioni sulla corretta compilazione del report di progetto.

Non modificare il formato del documento:

- Carattere: Times New Roman, 12pt

- Dimensione pagina: A4

- Margini: superiore/inferiore 2,5cm, sinistro/destro: 1,9cm

L’assegnazione della tesina può essere effettuata online, visitando il sito <https://www.pellegrini.tk/progetti/> ed inserendo i propri dati. Per qualsiasi problema, contattare il docente via email all’indirizzo [a.pellegrini@ing.uniroma2.it](mailto:a.pellegrini@ing.uniroma2.it)

# Descrizione del Minimondo

|  |  |
| --- | --- |
|  | Inserire all’interno di questo riquadro la specifica così come è stata fornita. Riportare nella colonna a sinistra la numerazione delle righe. Questi numeri dovranno essere utilizzati per riferirsi al testo nelle sezioni successive.  1.Un sistema di Customer Relationship Management (o gestione delle relazioni con i clienti) 2.è un sistema informativo che verte sulla fidelizzazione del cliente. Si vuole realizzare un 3.sistema CRM per un’azienda marketing-oriented che intende realizzare relazioni durevoli 4.di breve e lungo periodo con i propri clienti, massimizzando quindi il valore degli stessi.  5.La base di dati del sistema informativo dell’azienda di CRM deve poter memorizzare le 6.informazioni su tutti i clienti di interesse dell’azienda, caratterizzati da nome, cognome, 7.codice fiscale, data di nascita ed un insieme di contatti, sia in forma di indirizzi che di 8.recapiti telefonici, email, fax. Alcuni dei clienti sono società che ricevono servizi dalla 9.società di CRM. Di questi interessa anche mantenere il numero di partita IVA. Di tutti i 10.clienti interessa sapere qual è la data di registrazione nel sistema di CRM.  11.L’azienda di CRM in questione è di dimensione elevata ed ha a disposizione vari 12.funzionari che interagiscono con i clienti. A ciascun utente aziendale del sistema viene 13.assegnato un sottoinsieme di clienti da gestire.  14.Su base periodica, gli operatori dell’azienda di CRM contattano i clienti mediante uno 15.dei recapiti forniti. In questa fase operativa, l’utente deve inserire una nota testuale in cui 16.viene riportato un breve resoconto dell’interazione con il cliente, annotando anche 17.possibili risposte affermative alle proposte commerciali. Una risposta positiva di 18.accettazione di una proposta commerciale può essere associata ad un appuntamento in 19.sede. L’azienda ha più sedi, ciascuna caratterizzata da un indirizzo. In ciascuna sede sono 20.presenti una o più sale riunione, in cui è possibile ricevere i clienti. Non è possibile 21.assegnare una stessa sede, nello stesso giorno ed alla stessa ora, a più di un cliente. Agli 22.appuntamenti partecipano i clienti e gli operatori dell’azienda.  23.L’azienda ha anche un gruppo di manager che definisce quali sono le proposte 24.commerciali che l’azienda offre. Ogni proposta è identificata da un codice alfanumerico 25.definito internamente dall’azienda. I manager hanno la possibilità di creare nuove 26.proposte e di segnalare che alcune proposte già presenti nel sistema sono terminate, 27.ovverosia che non possono più essere fornite ai clienti.  28.Infine, l’azienda ha un settore commerciale i cui membri reclutano nuovi clienti e li 29.inseriscono all’interno del sistema.  30.In generale, il sistema informativo deve fornire le seguenti possibilità.  31.\* Visualizzare il singolo cliente, eventualmente con i dati dell’azienda e del referente 32.aziendale, con tutti i dettagli e le caratteristiche, l’elenco delle note cliente e l’elenco dei 33.servizi di consulenza acquistati.  34.\* Possibilità di visualizzare l’elenco clienti a cui un utente è assegnato.  35.\* Gestione delle note cliente: ogni volta che un cliente viene contattato deve essere 36.possibile registrare/modificare/cancellare una o più note relative alla conversazione 37.avvenuta e dell’utente che l’ha registrata.  38.\* Gestione delle opportunità: per ogni cliente deve essere possibile inserire una nuova 39.opportunità, cioè una proposta commerciale.  40.\* Gestione degli appuntamenti: deve essere possibile inserire un appuntamento con una 41.nota descrittiva, una data/ora e un cliente a cui è riferito.  42.\* Visualizzazione dell’agenda degli appuntamenti per un utente.  43.\* Possibilità di inserire nuovi servizi di consulenza (riservata ai manager).  44.\* Possibilità di inserire nuovi clienti (riservata al settore commerciale).  45.\* Possibilità di inserire nuovi utenti dell’applicativo web (riservata ai manager). |

# Analisi dei Requisiti

Lo scopo di questa sezione è raffinare la specifica fornita, andando ad effettuare un’operazione preliminare di disambiguazione.

## Identificazione dei termini ambigui e correzioni possibili

Compilare la seguente tabella, facendo riferimento alla specifica del minimondo di riferimento precedentemente indicata. Individuare i termini ambigui nella specifica (indicando la linea in cui essi compaiono), indicare il nuovo termine che si intende adottare nella specifica, ed indicare il motivo del cambiamento che si propone.

|  |  |  |  |
| --- | --- | --- | --- |
| **Linea** | **Termine** | **Nuovo termine** | **Motivo correzione** |
| 12 | Utente aziendale | Funzionario | E’ utilizzato come sinonimo di funzionario |
| 14 | Operatore | Funzionario | E’ utilizzato come sinonimo di funzionario |
| 15 | Utente | Funzionario | E’ utilizzato come sinonimo di funzionario |
| 28 | Membro | Funzionario | E’ utilizzato come sinonimo di funzionario |
| 35 | Note cliente | Note testuali | E’ utilizzato come sinonimo di note testuali |
| 17 | Risposta positiva di accettazione | Risposta affermativa | E’ utilizzato come sinonimo di risposta affermativa |
| 32 | Dettagli e caratteristiche | Informazioni | E’ utilizzato come sinonimo di informazioni |
| 41 | Nota descrittiva | Nota testuale | E’ utilizzato come sinonimo di nota testuale |
| 43 | Servizi di consulenza | Proposte commerciali | E’ utilizzato come sinonimo di proposte commerciali |

### 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, data di registrazione nel sistema ed un insieme di contatti (indirizzo, recapito telefonico, email, fax). Alcuni dei clienti sono società che ricevono servizi dalla società di CRM e di questi interessa anche mantenere il numero di partita IVA.  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 dell’azienda di CRM 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, ognuna delle quali 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 dell’azienda.  L’azienda ha anche un gruppo di manager che definisce quali sono le proposte commerciali che l’azienda offre, ognuna 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 funzionari 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 il funzionario associato, con tutte le informazioni, l’elenco delle note testuali e l’elenco delle proposte commerciali acquistate.  \* Possibilità di visualizzare l’elenco clienti a cui un funzionario è assegnato.  \* Gestione delle note testuali: 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 opportunità, cioè una proposta commerciale.  \* Gestione degli appuntamenti: deve essere possibile inserire un appuntamento con una nota testuale, una data/ora e un cliente a cui è riferito.  \* Visualizzazione dell’agenda degli appuntamenti per un funzionario.  \* Possibilità di inserire nuove proposte commerciali (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

Realizzare un dizionario dei termini, compilando la tabella qui sotto, a partire dalle specifiche precedentemente disambiguate

|  |  |  |  |
| --- | --- | --- | --- |
| **Termine** | **Descrizione** | **Sinonimi** | **Collegamenti** |
| Funzionario | Impiegato dell’azienda addetto alla gestione dei clienti | Utente, Membro, Operatore, Utente aziendale | Nota testuale, Cliente, Manager, Proposta commerciale, Appuntamento |
| Cliente | Fruitore dei servizi dell’azienda |  | Funzionario, Nota testuale, Settore commerciale, Proposta commerciale, Appuntamento |
| Nota testuale | Resoconto della chiamata tra funzionario e cliente | Nota descrittiva, Nota cliente | Funzionario, Cliente |
| Manager | Gestori delle proposte commerciali |  | Funzionario, Proposta commerciale |
| Settore commerciale | Incaricato di inserire nuovi client nel sistema |  | Cliente |
| Proposta commerciale | Servizio avanzato da manager per i clienti | Servizio di consulenza | Manager, Cliente, Funzionario |
| Appuntamento | Incontro tra funzionario e cliente |  | Cliente, Funzionario |

## Raggruppamento dei requisiti in insiemi omogenei

Per ciascun elemento “più importante” della specifica (riportata anche nel glossario precedente), estrapolare dalla specifica disambiguata le frasi ad esso associate. Compilare una tabella separata per ciascun elemento individuato.

|  |
| --- |
| **Frasi relative a Funzionario** |
| A ciascun funzionario del sistema viene assegnato un sottoinsieme di clienti da gestire.  Su base periodica, i funzionari dell’azienda di CRM 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.  Visualizzazione dell’agenda degli appuntamenti per un funzionario.  Possibilità di visualizzare l’elenco clienti a cui un funzionario è assegnato. |
| **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, data di registrazione nel sistema ed un insieme di contatti (indirizzo, recapito telefonico, e-mail, fax). Alcuni dei clienti sono società che ricevono servizi dalla società di CRM e di questi interessa anche mantenere il numero di partita IVA.  Visualizzare il singolo cliente, eventualmente con i dati dell’azienda e del referente aziendale, con tutte le informazioni, l’elenco delle note testuali e l’elenco delle proposte commerciali acquistate. |
| **Frasi relative a Nota testuale** |
| 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.  Gestione delle note testuali: 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. |
| **Frasi relative a Manager** |
| 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.  Possibilità di inserire nuove proposte commerciali (riservata ai manager).  Possibilità di inserire nuovi funzionari dell’applicativo web (riservata ai manager). |
| **Frasi relative a Settore commerciale** |
| Infine, l’azienda ha un settore commerciale i cui funzionari reclutano nuovi clienti e li inseriscono all’interno del sistema.  Possibilità di inserire nuovi clienti (riservata al settore commerciale). |
| **Frasi relative a Proposta commerciale** |
| Una risposta affermativa di una proposta commerciale può essere associata ad un appuntamento in sede.  L’azienda ha anche un gruppo di manager che definisce quali sono le proposte commerciali che l’azienda offre. Ogni proposta è identificata da un codice alfanumerico definito internamente dall’azienda. I manager hanno la possibilità di creare nuove proposte e di segnalare che alcune proposte già presenti nel sistema sono terminate, ovverosia che non possono più essere fornite ai clienti. |
| **Frasi relative a Appuntamento** |
| 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 dell’azienda.  Gestione degli appuntamenti: deve essere possibile inserire un appuntamento con una nota testuale, una data/ora e un cliente a cui è riferito. |

# Progettazione concettuale

## Costruzione dello schema E-R

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

### Il primo passo è identificare la prima entità, Cliente, con i relativi attributi e la sua specializzazione Società. Dopodiché si passa a costruire l’entità Funzionario, il quale è in stretta relazione con Cliente. Una delle relazioni che le legano si basa sulla partecipazione ad un Appuntamento, che in questo caso viene reificato e diventa un’entità a tutti gli effetti. La stessa è caratterizzata dalla Sede, in cui ha luogo, e questa, invece di lasciarla come semplice attributo, ha senso renderla un’entità a sé stante. Inoltre, l’entità Appuntamento dovrebbe avere come attributo la Proposta Commerciale per il quale è stato organizzato, ma anche in questo la reificazione dell’attributo è la soluzione migliore, sia per le sue due specializzazioni che per le relazioni che ha con Cliente. Inoltre, è collegata anche all’entità Manager poiché sono proprio i manager dell’azienda ad avanzare le proposte commerciali e, successivamente, a farle terminare. Si costruiscono poi le entità Contatto, che ha relazioni con Cliente e Funzionario, e Nota, dipendente da Contatto, che è in relazione con Proposta Commerciale.

### Integrazione finale

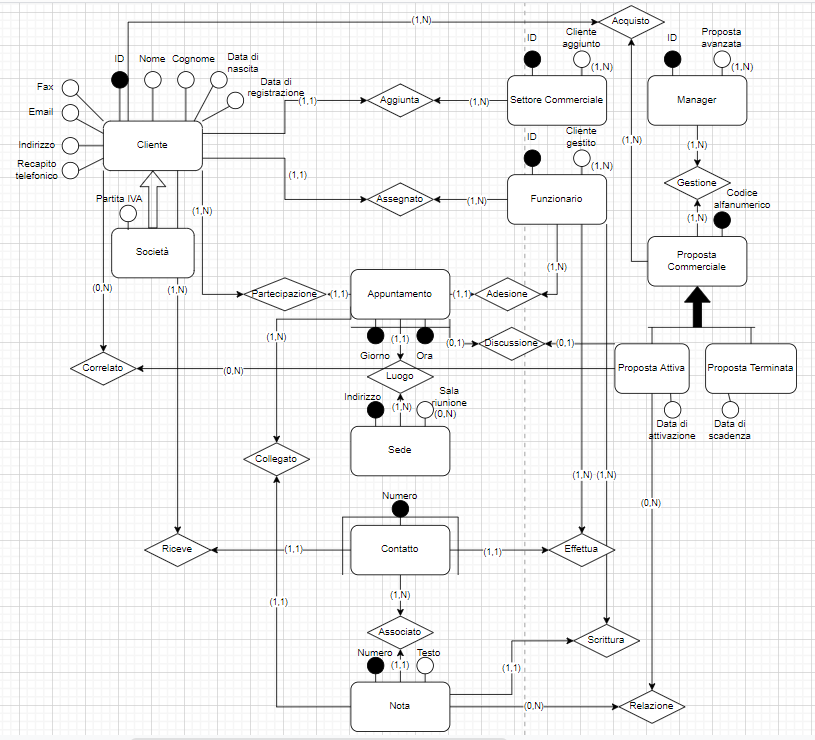
Nell’integrazione finale delle varie parti dello schema E-R è possibile che si evidenzino dei conflitti sui nomi utilizzati e dei conflitti strutturali. Prima di riportare lo schema E-R finale, descrivere quali passi sono stati adottati per risolvere tali conflitti.

Uno tra i conflitti strutturali più evidenti è stato l’identificatore dell’entità Funzionario; nel primo schema lo avevo posto come entità debole con identificatore esterno verso Cliente, mentre nel secondo come entità forte con un suo identificatore.

Un altro conflitto strutturale si è presentato per l’identificatore dell’entità Nota; anche in questo caso nel primo schema si aveva un’entità debole verso Cliente e Funzionario, e nel secondo come entità forte.

Un conflitto sui nomi c’è stato tra la relazione che collega Funzionario e Cliente e quella che lega Manager a Proposta Commerciale, poiché in entrambe una delle due entità gestiva l’altra.

Un altro conflitto sui nomi è occorso per le relazioni che legano Appuntamento con Funzionario e Cliente; anche in questo caso avevano lo stesso nome poiché entrambe le entità partecipavano all’entità Appuntamento.



## Regole aziendali

Laddove la specifica non sia catturata in maniera completa dallo schema E-R, corredare lo stesso in questo paragrafo con l’insieme delle regole aziendali necessarie a completare la progettazione concettuale.

L’instaurazione di un contatto deve avvenire tramite uno dei recapiti salvati.

L’aggiunta di un nuovo cliente deve essere a carico del settore commerciale.

## Dizionario dei dati

Completare la progettazione concettuale riportando nella tabella seguente il dizionario dei dati

|  |  |  |  |
| --- | --- | --- | --- |
| **Entità** | **Descrizione** | **Attributi** | **Identificatori** |
| Cliente | Fruitore dei servizi offerti dall’azienda | Codice Fiscale, Nome, Cognome, Data di nascita, Data di registrazione, Fax, E-mail, Indirizzo, Recapito Telefonico | Codice Fiscale |
| Funzionario | Dipendente dell’azienda | ID, Cliente gestito | ID |
| Società | Sottoinsieme di Cliente | Codice Fiscale, Partita IVA, … (tutti gli attributi di Cliente) | Codice Fiscale |
| Appuntamento | Incontro tra Cliente e Funzionario | Giorno, Ora, Sede | Giorno, Ora, Sede (Identificatore esterno) |
| Sede | Luogo dell’appuntamento | Indirizzo, Sala riunione | Indirizzo |
| Contatto | Conversazione tra Cliente e Funzionario | Cliente, Funzionario, Numero | Cliente (Identificatore esterno), Funzionario (Identificatore esterno), Numero |
| Nota | Riassunto scritto dal Funzionario di un Contatto | Numero | Numero, Testo |
| Proposta Commerciale | Servizio offerto dall’azienda ai Clienti | Codice Alfanumerico | Codice Alfanumerico |
| Settore Commerciale | Dipendente dell’azienda, dedito all’inserimento nel sistema di nuovi Clienti | ID, Cliente aggiunto | ID |
| Manager | Dipendente dell’azienda, dedito alla gestione delle Proposte Commerciali | ID, Proposta avanzata | ID |
| Proposta Attiva | Storicizzazione di Proposta Commerciale, offerta attualmente attiva | Codice Alfanumerico, Data di attivazione | Codice Alfanumerico |
| Proposta Terminata | Storicizzazione di Proposta Commerciale, offerta terminata | Codice Alfanumerico, Data di scadenza | Codice Alfanumerico |

# Progettazione logica

## Volume dei dati

Questa sezione serve ad illustrare qual è il carico che la base di dati dovrà sopportare. A tal fine, è necessario prevedere un volume di dati attesi. Compilare la tabella sottostante, per ciascun concetto identificato nello schema E-R. I volumi devono essere stimati dallo studente in maniera ragionevole rispetto all’operatività presunta dell’applicativo.

|  |  |  |
| --- | --- | --- |
| **Concetto nello schema** | **Tipo[[1]](#footnote-1)** | **Volume atteso** |
| Cliente | E | 100.000 |
| Società | E | 1.000 |
| Appuntamento | E | 100.000 |
| Funzionario | E | 10.000 |
| Proposta Commerciale | E | 10.000 |
| Sede | E | 10 |
| Contatto | E | 1.000.000 |
| Nota | E | 1.000.000 |
| Manager | E | 100 |
| Settore Commerciale | E | 1.000 |
| Proposta Attiva | E | 1.000 |
| Proposta Terminata | E | 9.000 |
| Assegnato | R | 100.000 |
| Aggiunta | R | 100.000 |
| Partecipazione | R | 1.000.000 |
| Adesione | R | 1.000.000 |
| Collegato | R | 100.000 |
| Associato | R | 1.000.000 |
| Discussione | R | 10.000 |
| Riceve | R | 1.000.000 |
| Effettua | R | 1.000.000 |
| Scrittura | R | 1.000.000 |
| Relazione | R | 10.000 |
| Gestione | R | 10.000 |
| Luogo | R | 1.000.000 |
| Acquisto | R | 10.000 |
| Correlato | R | 100.000 |

## Tavola delle operazioni

Rappresentare nella tabella sottostante tutte le operazioni sulla base di dati che devono essere supportate dall’applicazione, con la frequenza attesa. Le operazioni da supportare devono essere desunte dalle specifiche raccolte.

|  |  |  |
| --- | --- | --- |
| **Cod.** | **Descrizione** | **Frequenza attesa** |
| OP1 | Aggiunta nuovo cliente | 100/g |
| OP2 | Aggiunta proposta commerciale | 1/g |
| OP3 | Aggiunta nota | 10/g |
| OP4 | Aggiunta appuntamento | 10/g |
| OP5 | Aggiunta opportunità | 10/g |
| OP6 | Aggiunta funzionario | 1/g |
| OP7 | Aggiunta contatto | 10/g |
| OP8 | Revisione utenti assegnati ad un funzionario | 100/g |
| OP9 | Visualizzazione consulenze attive | 10/g |
| OP10 | Revisione cliente | 10/g |
| OP11 | Revisione appuntamenti di un funzionario | 10/g |
| OP12 | Invalidazione consulenza | 1/g |
| OP13 | Modifica nota | 1/g |
| OP14 | Eliminazione nota | 1/g |

## Costo delle operazioni

In riferimento a tutte le operazioni precedentemente indicate, calcolarne il costo supponendo, per questa fase del progetto, che il costo in scrittura di un dato sia doppio rispetto a quello in lettura.

OP1: Aggiunta nuovo cliente

Si fa 1 accesso in scrittura all’entità Cliente, 1 accesso in scrittura all’entità Funzionario, 1 accesso in scrittura all’entità Settore Commerciale, 1 accesso in scrittura alla relazione Assegnato, 1 accesso in scrittura alla relazione Aggiunta, 1 accesso in lettura all’entità Funzionario, 1 accesso in lettura all’entità Settore Commerciale.

OP2: Aggiunta proposta commerciale

Si fa 1 accesso in scrittura all’entità Proposta Commerciale, 1 accesso in scrittura all’entità Manager, 1 accesso in scrittura all’entità Proposta Attiva, 1 accesso in scrittura alla relazione Gestione, 1 accesso in lettura all’entità Manager.

OP3: Aggiunta nota

Si fa 1 accesso in scrittura all’entità Nota, 1 accesso in scrittura alla relazione Collegato, 1 accesso in scrittura alla relazione Associato, 1 accesso in scrittura alla relazione Scrittura, 1 accesso in scrittura alla relazione Relazione, 1 accesso in lettura all’entità Proposta Attiva, 1 accesso in lettura all’entità Contatto, 1 accesso in lettura all’entità Funzionario, 1 accesso in lettura all’entità Appuntamento.

OP4: Aggiunta appuntamento

Si fa 1 accesso in scrittura all’entità Appuntamento, 1 accesso in scrittura alla relazione Partecipazione, 1 accesso in scrittura alla relazione Adesione, 1 accesso in lettura all’entità Funzionario, 1 accesso in lettura all’entità Cliente.

OP5: Aggiunta opportunità

Si fa 1 accesso in lettura all’entità Cliente, 1 accesso in lettura all’entità Proposta Attiva, 1 accesso in scrittura all’entità Correlato.

OP6: Aggiunta funzionario

Si fa 1 accesso in scrittura all’entità Funzionario.

OP7: Aggiunta contatto

Si fa 1 accesso in lettura all’entità Cliente, 1 accesso in lettura all’entità Funzionario, 1 accesso in scrittura alla relazione Riceve, 1 accesso in scrittura alla relazione Effettua, 1 accesso in scrittura Contatto.

OP8: Revisione utenti assegnati ad un funzionario

Si fa 1 accesso in lettura all’entità Funzionario.

OP9: Visualizzazione consulenze attive

Si fa 1 accesso in lettura all’entità Manager, 100 accessi in lettura all’entità Proposta Commerciale, 100 accessi in lettura all’entità Proposta Attiva, 100 accessi in lettura alla relazione Gestione.

OP10: Revisione cliente

Si fa 1 accesso in lettura all’entità Cliente, 1 accesso in lettura all’entità Funzionario, 1 accesso in lettura all’entità Contatto, 1 accesso in lettura all’entità Appuntamento, 1 accesso in lettura all’entità Sede, 1 accesso in lettura all’entità Proposta Commerciale, 1 accesso in lettura all’entità Proposta Attiva, 1 accesso in lettura all’entità Nota, 1 accesso in lettura alla relazione Assegnato, 1 accesso in lettura alla relazione Riceve, 1 accesso in lettura alla relazione Partecipazione, 1 accesso in lettura alla relazione Discussione, 1 accesso in lettura alla relazione Associato, 1 accesso in lettura alla relazione Acquisto.

OP11: Revisione appuntamenti di un funzionario

Si fa 1 accesso in lettura all’entità Funzionario, 10 accessi in lettura all’entità Appuntamento, 10 accessi in lettura alla relazione Adesione, 10 accessi in lettura alla relazione Luogo.

OP12: Invalidazione consulenza

Si fa 1 accesso in scrittura all’entità Proposta Attiva, 1 accesso in scrittura all’entità Proposta Terminata, 1 accesso in lettura all’entità Proposta Commerciale.

OP13: Modifica nota

Si fa 1 accesso in scrittura all’entità Nota, 1 accesso in lettura all’entità Nota.

OP14: Eliminazione nota

Si fa 1 accesso in scrittura all’entità Nota, 1 accesso in scrittura alla relazione Associato, 1 accesso in scrittura alla relazione Scrittura, 1 accesso in scrittura alla relazione Relazione, 1 accesso in lettura all’entità Contatto, 1 accesso in lettura all’entità Funzionario, 1 accesso in lettura all’entità Proposta Attiva.

## Ristrutturazione dello schema E-R

Descrivere (laddove necessario fornendo anche degli schemi) quali passi vengono adottati per ristrutturare lo schema E-R, ad esempio in termini di:

Analisi delle ridondanze

Eliminazione delle generalizzazioni

Scelta degli identificatori primari

Si noti che in questa fase è possibile fare riferimento al costo delle operazioni precedentemente realizzato per guidare le scelte. Ad esempio, un leggero spreco di memoria legato alla non rimozione di ridondanze può essere facilmente giustificato da un guadagno in termini di prestazioni.

1) Analisi delle ridondanze

Sebbene possa sembrare che aggiungendo un attributo NumProposteAttive nell’entità Proposta Commerciale si risparmino parecchi accessi, in realtà non è così. Ciò accadrebbe se non si necessitasse anche di visualizzare le singole proposte, cosa che mantiene invariati gli accessi per l’operazione 8. Inoltre questo causerebbe un aumento, anche se piccolo, degli accessi per l’operazione 11; infatti senza attributo il numero di accessi è 1\*(2+2+1) = 5, mentre in presenza dell’attributo si avrebbero 1\*(2+2+1+2) = 7, 2 in più dovuti alla scrittura dell’attributo NumProposteAttive. In tutto ciò si avrebbe anche uno spreco di memoria dovuto alla memorizzazione dell’attributo, cosa che fa decadere totalmente l’idea di inserirlo.

2) Eliminazione delle generalizzazioni

Nello Schema E-R ci sono in totale due generalizzazioni, una che ha come padre Cliente e l’altra Proposta Commerciale.

La prima, poiché si tratta di una generalizzazione parziale e l’entità padre ha un volume di dati molto maggiore rispetto alla sua specializzazione, si può essere tradurre in un accorpamento dell’entità figlia nell’entità padre.

La seconda, trattandosi di una storicizzazione e data la considerevole quantità di operazioni svolte con le entità in quanto Proposta, si può tradurre, come prima, in un accorpamento delle entità figlie nell’entità padre.

3) Scelta degli identificatori primari

Nell’entità Contatto, invece di adoperare due identificatori esterni a cui si aggiunge il numero dei contatti effettuati in precedenza tra il cliente e il funzionario, si può ricorrere all’uso di un identificatore interno univoco ID.

## Trasformazione di attributi e identificatori

Qualora siano presenti, in questa fase della progettazione, attributi ripetuti o identificatori esterni, descrivere quali trasformazioni vengono realizzate sul modello per facilitare la traduzione nello schema relazionale.

Rimane un identificatore esterno da trasformare.

L’identificatore esterno di Appuntamento diventa un suo attributo che costituirà il suo identificatore.

Nell’entità Sede si ha un attributo multivalore che può essere trasformato in un’entità a sé stante connessa all’entità originaria tramite una relazione uno a molti.

Nelle entità Funzionario, Settore Commerciale e Manager si hanno rispettivamente gli attributi multivalore Cliente gestito, Cliente aggiunto e Proposta avanzata che vengono eliminati perché nel reificarli per renderli entità a sé stanti ci si accorge che sono una ripetizione di entità già presenti.

## Traduzione di entità e associazioni

Riportare in questa sezione la traduzione di entità ed associazioni nello schema relazionale.

Fornire una rappresentazione grafica del modello relazionale completo.

Cliente (CF, Nome, Cognome, DataDiNascita, DataDiRegistrazione, Fax, E-mail, Indirizzo, RecapitoTelefonico, IDSettoreCommerciale, IDFunzionario, PartitaIVA)

Funzionario (ID)

Settore Commerciale (ID)

Appuntamento (Giorno, Ora, IndirizzoSede, AdesioneFunzionario, PartecipazioneCliente)

Acquisto (Cliente, CodicePropostaCommerciale)

Correlato (Cliente, CodicePropostaCommerciale)

Sede (Indirizzo, SalaRiunione)

Sala Riunione (Numero)

Contatto (ID, ClienteContattato, FunzionarioContattante)

Nota (Numero, ContattoAssociato, AppuntamentoCollegato, Funzionario)

Relazione (Nota, CodicePropostaCommerciale)

Proposta Commerciale (CodiceAlfanumerico, DataDiAttivazione, DataDiScadenza, Validità)

Manager (ID)

Gestione (Manager, CodicePropostaCommerciale)

VIR

IDSettoreCommerciale → SettoreCommerciale

IDFunzionario → Funzionario

IndirizzoSede → Sede

AdesioneFunzionario → Funzionario

PartecipazioneCliente → Cliente

Cliente → Cliente

CodicePropostaCommerciale → PropostaCommerciale

Cliente → Cliente

CodicePropostaCommerciale → PropostaCommerciale

SalaRiunione → SalaRiunione

ClienteContattato → Cliente

FunzionarioContattante → Funzionario

ContattoAssociato → Contatto

AppuntamentoCollegato → Appuntamento

Funzionario → Funzionario

Nota → Nota

CodicePropostaCommerciale → PropostaCommerciale

Manager → Manager

CodicePropostaCommerciale → PropostaCommerciale

## Normalizzazione del modello relazionale

Effettuare la normalizzazione del modello relazionale precedentemente descritto (in forma grafica) andando a mostrare le forme 1NF, 2NF, 3NF.

* 1NF

La normalizzazione in prima forma non varia il modello relazionale precedente perché tutti gli attributi sono definiti su domini con valori atomici e ogni attributo contiene un singolo valore:

Cliente (CF, Nome, Cognome, DataDiNascita, DataDiRegistrazione, Fax, E-mail, Indirizzo, RecapitoTelefonico, IDSettoreCommerciale, IDFunzionario, PartitaIVA)

Funzionario (ID)

Settore Commerciale (ID)

Appuntamento (Giorno, Ora, IndirizzoSede, AdesioneFunzionario, PartecipazioneCliente)

Acquisto (Cliente, CodicePropostaCommerciale)

Correlato (Cliente, CodicePropostaCommerciale)

Sede (Indirizzo, SalaRiunione)

Sala Riunione (Numero)

Contatto (ID, ClienteContattato, FunzionarioContattante)

Nota (Numero, ContattoAssociato, AppuntamentoCollegato, Funzionario)

Relazione (Nota, CodicePropostaCommerciale)

Proposta Commerciale (CodiceAlfanumerico, DataDiAttivazione, DataDiScadenza, Validità)

Manager (ID)

Gestione (Manager, CodicePropostaCommerciale)

* 2NF

Anche la normalizzazione in seconda forma non fa variare il modello relazionale perché ogni attributo non-chiave dipende funzionalmente dall’intera chiave:

Cliente (CF, Nome, Cognome, DataDiNascita, DataDiRegistrazione, Fax, E-mail, Indirizzo, RecapitoTelefonico, IDSettoreCommerciale, IDFunzionario, PartitaIVA)

Funzionario (ID)

Settore Commerciale (ID)

Appuntamento (Giorno, Ora, IndirizzoSede, AdesioneFunzionario, PartecipazioneCliente)

Acquisto (Cliente, CodicePropostaCommerciale)

Correlato (Cliente, CodicePropostaCommerciale)

Sede (Indirizzo, SalaRiunione)

Sala Riunione (Numero)

Contatto (ID, ClienteContattato, FunzionarioContattante)

Nota (Numero, ContattoAssociato, AppuntamentoCollegato, Funzionario)

Relazione (Nota, Codice Proposta commerciale)

Proposta Commerciale (CodiceAlfanumerico, DataDiAttivazione, DataDiScadenza, Validità)

Manager (ID)

Gestione (Manager, CodicePropostaCommerciale)

* 3NF

Invece, nel caso di normalizzazione in terza forma si hanno varie modifiche da apportare a varie tabelle, come nel caso di Appuntamento che ha AdesioneFunzionario dipendente da PartecipazioneCliente, stessa cosa vale per Contatto nel quale FunzionarioContattante dipende da ClienteContattato, infine per Nota si hanno ContattoAssociato e AppuntamentoCollegato che dipendono da Funzionario:

Cliente (CF, Nome, Cognome, DataDiNascita, DataDiRegistrazione, Fax, E-mail, Indirizzo, RecapitoTelefonico, IDSettoreCommerciale, IDFunzionario, PartitaIVA)

Funzionario (ID)

Settore Commerciale (ID)

Appuntamento (Giorno, Ora, IndirizzoSede, PartecipazioneCliente)

FunzionarioAssociato (PartecipazioneCliente, AdesioneFunzionario)

Acquisto (Cliente, CodicePropostaCommerciale)

Correlato (Cliente, CodicePropostaCommerciale)

Sede (Indirizzo, SalaRiunione)

Sala Riunione (Numero)

Contatto (ID, Cliente contattato)

FunzionarioContatto (ClienteContattato, FunzionarioContattante)

Nota (Numero, Funzionario)

FunzionarioNota (Funzionario, ContattoAssociato, AppuntamentoCollegato)

Relazione (Nota, CodicePropostaCommerciale)

Proposta Commerciale (CodiceAlfanumerico, DataDiAttivazione, DataDiScadenza, Validità)

Manager (ID)

Gestione (Manager, CodicePropostaCommerciale)

# Progettazione fisica

## Utenti e privilegi

Descrivere, all’interno dell’applicazione, quali utenti sono stati previsti con quali privilegi di accesso su quali tabelle, giustificando le scelte progettuali.

## Nell’applicazione sono previsti 4 differenti utenti: l’utente Login che serve per collegarsi al database e identificare l’utente appena connesso; l’utente Utente che corrisponde al funzionario generico dell’azienda e può accedere sia in scrittura che in lettura a molte tra le tabelle presenti, ma non può accedere alle tabelle Dipendente e Gestione né in lettura né in scrittura perché non deve essere a conoscenza né dei manager dell’azienda né delle proposte da loro avanzate, né alle tabelle PropostaCommerciale e Cliente se non in lettura, poiché non deve essere in grado di modificare campi di competenza dei colleghi; l’utente Manager che può accedere alle tabelle Proposta Commerciale, Gestione e Dipendente in scrittura e lettura per le sue operazioni; l’utente Commerciale che può accedere solo in scrittura alla tabella Cliente per l’aggiunta di nuovi clienti.

## Strutture di memorizzazione

Compilare la tabella seguente indicando quali tipi di dato vengono utilizzati per memorizzare le informazioni di interesse nelle tabelle, per ciascuna tabella.

|  |  |  |
| --- | --- | --- |
| **Acquisto** | | |
| **Colonna** | **Tipo di dato** | **Attributi[[2]](#footnote-2)** |
| **Cliente** | VARCHAR(128) | PK, NN |
| **CodiceProposta** | VARCHAR(128) | PK, NN |

|  |  |  |
| --- | --- | --- |
| **Appuntamento** | | |
| **Colonna** | **Tipo di dato** | **Attributi[[3]](#footnote-3)** |
| **Giorno** | DATE | NN |
| **Ora** | TIME | NN |
| **IndirizzoSede** | VARCHAR(128) | NN |
| **AdesioneFunzionario** | INT | NN |
| **PartecipazioneCliente** | VARCHAR(128) | NN |
| **ID** | INT | PK, NN |

|  |  |  |
| --- | --- | --- |
| **Cliente** | | |
| **Colonna** | **Tipo di dato** | **Attributi[[4]](#footnote-4)** |
| **CodiceFiscale** | VARCHAR(128) | PK, NN |
| **Nome** | VARCHAR(45) | NN |
| **Cognome** | VARCHAR(45) | NN |
| **DataDiNascita** | DATE | NN |
| **DataDiRegistrazione** | DATE | NN |
| **Fax** | INT |  |
| **Email** | VARCHAR(128) |  |
| **Indirizzo** | VARCHAR(128) |  |
| **RecapitoTelefonico** | INT |  |
| **IDComm** | INT |  |
| **IDFunz** | INT |  |
| **PartitaIVA** | VARCHAR(45) |  |

|  |  |  |
| --- | --- | --- |
| **Contatto** | | |
| **Colonna** | **Tipo di dato** | **Attributi[[5]](#footnote-5)** |
| **ID** | INT | PK, NN, AI |
| **ClienteContattato** | VARCHAR(128) | NN |
| **FunzionarioContattante** | INT | NN |

|  |  |  |
| --- | --- | --- |
| **Correlato** | | |
| **Colonna** | **Tipo di dato** | **Attributi[[6]](#footnote-6)** |
| **Cliente** | VARCHAR(128) | PK, NN |
| **CodiceProposta** | VARCHAR(128) | PK, NN |

|  |  |  |
| --- | --- | --- |
| **Dipendente** | | |
| **Colonna** | **Tipo di dato** | **Attributi[[7]](#footnote-7)** |
| **ID** | INT | PK, NN, AI |
| **Username** | VARCHAR(45) | NN, UQ |
| **Password** | VARCHAR(45) | NN |
| **Ruolo** | ENUM(‘utente’,’manager’,’commerciale’) | NN |

|  |  |  |
| --- | --- | --- |
| **Gestione** | | |
| **Colonna** | **Tipo di dato** | **Attributi[[8]](#footnote-8)** |
| **Manager** | INT | PK, NN |
| **CodiceProposta** | VARCHAR(128) | PK, NN, UQ |

|  |  |  |
| --- | --- | --- |
| **Nota** | | |
| **Colonna** | **Tipo di dato** | **Attributi[[9]](#footnote-9)** |
| **Numero** | INT | PK, NN |
| **Funzionario** | INT | NN |
| **Testo** | VARCHAR(4096) | NN |
| **ContattoAssociato** | INT |  |
| **AppuntamentoID** | INT |  |

|  |  |  |
| --- | --- | --- |
| **PropostaCommerciale** | | |
| **Colonna** | **Tipo di dato** | **Attributi[[10]](#footnote-10)** |
| **CodiceAlfanumerico** | VARCHAR(128) | PK, NN |
| **DataDiAttivazione** | DATE | NN |
| **DataDiScadenza** | DATE | NN |
| **Validità** | ENUM(‘Attiva’, ‘Terminata’) | NN |

|  |  |  |
| --- | --- | --- |
| **Relazione** | | |
| **Colonna** | **Tipo di dato** | **Attributi[[11]](#footnote-11)** |
| **Nota** | INT | PK, NN |
| **CodiceProposta** | VARCHAR(128) | PK, NN |

|  |  |  |
| --- | --- | --- |
| **Sede** | | |
| **Colonna** | **Tipo di dato** | **Attributi[[12]](#footnote-12)** |
| **Indirizzo** | VARCHAR(128) | PK, NN |
| **SalaRiunione** | VARCHAR(45) | NN |

## Indici

Compilare la seguente tabella, per ciascuna tabella del database in cui sono presenti degli indici. Descrivere le motivazioni che hanno portato alla creazione di un indice.

|  |  |
| --- | --- |
| **Acquisto** | |
| **Indice <nome>** | **Tipo[[13]](#footnote-13):** |
| Primary | PR |
| fk\_Proposta\_idx | IDX |
| fk\_Cliente\_idx | IDX |

|  |  |
| --- | --- |
| **Appuntamento** | |
| **Indice <nome>** | **Tipo[[14]](#footnote-14):** |
| Primary | PR |
| Unique | UQ |
| fk\_Appuntamento\_idx | IDX |
| fk\_ID\_Cliente\_idx | IDX |
| fk\_Sede\_idx | IDX |

|  |  |
| --- | --- |
| **Cliente** | |
| **Indice <nome>** | **Tipo[[15]](#footnote-15):** |
| Primary | PR |
| fk\_ID\_Comm\_idx | IDX |
| fk\_ID\_Funz\_idx | IDX |

|  |  |
| --- | --- |
| **Contatto** | |
| **Indice <nome>** | **Tipo[[16]](#footnote-16):** |
| Primary | PR |
| fk\_Cliente\_idx | IDX |
| fk\_Dipendente\_idx | IDX |

|  |  |
| --- | --- |
| **Correlato** | |
| **Indice <nome>** | **Tipo[[17]](#footnote-17):** |
| Primary | PR |
| fk\_Proposta\_idx | IDX |
| fk\_Cliente\_idx | IDX |

|  |  |
| --- | --- |
| **Dipendente** | |
| **Indice <nome>** | **Tipo[[18]](#footnote-18):** |
| Primary | PR |
| Unique | UQ |

|  |  |
| --- | --- |
| **Gestione** | |
| **Indice <nome>** | **Tipo[[19]](#footnote-19):** |
| Primary | PR |
| fk\_Proposta\_idx | IDX |
| fk\_Manager\_idx | IDX |

|  |  |
| --- | --- |
| **Nota** | |
| **Indice <nome>** | **Tipo[[20]](#footnote-20):** |
| Primary | PR |
| fk\_Dipendente\_idx | IDX |
| fk\_Contatto\_idx | IDX |
| fk\_Appuntamento\_idx | IDX |

|  |  |
| --- | --- |
| **PropostaCommerciale** | |
| **Indice <nome>** | **Tipo[[21]](#footnote-21):** |
| Primary | PR |

|  |  |
| --- | --- |
| **Relazione** | |
| **Indice <nome>** | **Tipo[[22]](#footnote-22):** |
| Primary | PR |
| fk\_Proposta\_idx | IDX |
| fk\_Nota\_idx | IDX |

|  |  |
| --- | --- |
| **Sede** | |
| **Indice <nome>** | **Tipo[[23]](#footnote-23):** |
| Primary | PR |

## Trigger

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

## È stato introdotto un BeforeInsert nella tabella Correlato, poiché si ha una fk riferita alla tabella Proposta Commerciale, la quale comprende sia Proposte Attive che Proposte Terminate, che dovrebbe riferirsi solo alle proposte attualmente attive. Per fare in modo che un Utente non inserisca, volontariamente o involontariamente, il codice di una proposta già terminata, si ricorre al seguente trigger:

CREATE

DEFINER=`root`@`localhost`

TRIGGER `azienda`.`Correlato\_BEFORE\_INSERT`

BEFORE INSERT ON `azienda`.`Correlato`

FOR EACH ROW

BEGIN

DECLARE temp VARCHAR(128);

SET temp = NEW.CodiceProposta;

if (select count(\*) from proposte\_terminate where `CodiceAlfanumerico` = temp) > 0 then

SET NEW.CodiceProposta = null;

end if;

END

Si hanno inoltre 2 BeforeInsert e 2 AfterInsert nelle tabelle Appuntamento e Contatto con lo stesso scopo, cioè far sì che si abbia un auto-incremento di 2 prima dell’inserimento di una qualsiasi riga e poi lo si riporti a 1 appena si finisce, il tutto per avere identificativi dispari per Contatto e identificativi pari per Appuntamento e quindi poterli distinguere all’interno di una stored procedure:

CREATE DEFINER=`root`@`localhost` TRIGGER `Appuntamento\_BEFORE\_INSERT` BEFORE INSERT ON `Appuntamento` FOR EACH ROW BEGIN

SET @@auto\_increment\_increment=2;

SET @@auto\_increment\_offset=2;

END

CREATE DEFINER=`root`@`localhost` TRIGGER `Appuntamento\_AFTER\_INSERT` AFTER INSERT ON `Appuntamento` FOR EACH ROW BEGIN

SET @@auto\_increment\_increment=1;

SET @@auto\_increment\_offset=1;

END

CREATE DEFINER=`root`@`localhost` TRIGGER `Contatto\_BEFORE\_INSERT` BEFORE INSERT ON `Contatto` FOR EACH ROW BEGIN

SET @@auto\_increment\_increment=2;

SET @@auto\_increment\_offset=2;

END

CREATE DEFINER=`root`@`localhost` TRIGGER `Contatto\_AFTER\_INSERT` AFTER INSERT ON `Contatto` FOR EACH ROW BEGIN

SET @@auto\_increment\_increment=1;

SET @@auto\_increment\_offset=1;

END

Infine si ha un BeforeInsert nella tabella Relazione che serve a evitare preventivamente che venga immesso un codice di una proposta già terminata:

CREATE DEFINER=`root`@`localhost` TRIGGER `Relazione\_BEFORE\_INSERT` BEFORE INSERT ON `Relazione` FOR EACH ROW BEGIN

if (select count(\*) from proposte\_terminate where `CodiceAlfanumerico` = new.CodiceProposta) > 0 then

signal sqlstate '45000';

end if;

END

## Eventi

Descrivere quali eventi sono stati implementati, mostrando il codice SQL per la loro instanziazione. Si descriva anche se gli eventi sono istanziati soltanto in fase di configurazione del sistema, o se alcuni eventi specifici vengono istanziati in maniera effimera durante l’esecuzione di alcune procedure.

## Viste

Mostrare e commentare il codice SQL necessario a creare tutte le viste necessarie per l’implementazione dell’applicazione.

## Si ha una sola vista all’interno dell’intera applicazione per facilitare la creazione del trigger esposto in una sezione precedente.

CREATE

ALGORITHM = UNDEFINED

DEFINER = `root`@`localhost`

SQL SECURITY DEFINER

VIEW `proposte\_terminate` AS

SELECT

`PropostaCommerciale`.`CodiceAlfanumerico` AS `CodiceAlfanumerico`

FROM

`PropostaCommerciale`

WHERE

(`PropostaCommerciale`.Validità = 'Terminata')

La vista seleziona tutte le proposte all’interno della tabella PropostaCommerciale che sono terminate tramite l’uso di una select e dell’attributo `Validità`, aggiunto all’entità nel momento in cui è stata eliminata la generalizzazione e si sono accorpate le due entità figlie nell’entità padre per differenziare se ci si riferisse all’una o all’altra.

## Stored Procedures e transazioni

Mostrare e commentare le stored procedure che sono state realizzate per implementare la logica applicativa delle operazioni sui dati, evidenziando quando (e perché) sono state realizzate operazioni transazionali complesse.

* La prima stored procedure serve per fornire Giorno, Data e Sede di tutti gli appuntamenti di un Funzionario che lo richiede:

CREATE DEFINER=`root`@`localhost` PROCEDURE `agenda`(in ID INT)

BEGIN

select Appuntamento.Giorno,Appuntamento.Ora,Appuntamento.IndirizzoSede from Appuntamento where Appuntamento.AdesioneFunzionario = ID;

END

* La seconda serve per eliminare una nota precedentemente immessa nel sistema e di cui ora non si necessita più:

CREATE DEFINER=`root`@`localhost` PROCEDURE `eliminare\_nota`(in ID INT)

BEGIN

delete from Nota where Nota.Numero = ID;

END

* La terza viene utilizzata per inserire all’interno del sistema un appuntamento e farsi restituire l’identificativo dello stesso:

CREATE DEFINER=`root`@`localhost` PROCEDURE `inserire\_appuntamento`(in giorno DATE, in ora TIME, in sede varchar(128), in func INT, in cliente varchar(128), out codice\_appuntamento INT)

BEGIN

declare exit handler for sqlexception

begin

rollback;

resignal;

end;

set transaction isolation level repeatable read;

start transaction;

insert into `Appuntamento` (`Giorno`, `Ora`, `IndirizzoSede`, `AdesioneFunzionario`, `PartecipazioneCliente`)

values (giorno, ora, sede, func, cliente);

commit;

set codice\_appuntamento = last\_insert\_id();

END

* La quarta si usa per inserire un cliente nuovo nel sistema e si differenzia in base alla presenza della partita IVA:

CREATE DEFINER=`root`@`localhost` PROCEDURE `inserire\_cliente`(in cod\_fisc varchar(128), in nam varchar(45), in surname varchar(45), in birthday DATE, in registration\_date DATE, in fax INT, in email varchar(128), in address varchar(128), in phone INT, in comm INT, in func INT, in iva varchar(45))

BEGIN

if(iva != '') then

insert into `Cliente` (`CodiceFiscale`, `Nome`, `Cognome`, `DataDiNascita`, `DataDiRegistrazione`, `Fax`, `Email`, `Indirizzo`, `RecapitoTelefonico`, `IDComm`, `IDFunz`, `PartitaIVA`)

values (cod\_fisc, nam, surname, birthday, registration\_date, fax, email, address, phone, comm, func, iva);

else

insert into `Cliente` (`CodiceFiscale`, `Nome`, `Cognome`, `DataDiNascita`, `DataDiRegistrazione`, `Fax`, `Email`, `Indirizzo`, `RecapitoTelefonico`, `IDComm`, `IDFunz`, `PartitaIVA`)

values (cod\_fisc, nam, surname, birthday, registration\_date, fax, email, address, phone, comm, func, null);

end if;

END

* La quinta è finalizzata all’inserimento da parte di un manager di una nuova proposta commerciale, ma si usa una transazione poiché bisogna inserire più elementi in varie tabelle e se l’inserimento avvenisse solo all’interno di un’unica tabella, si avrebbe inconsistenza:

CREATE DEFINER=`root`@`localhost` PROCEDURE `inserire\_consulenza`(in var\_username varchar(45), in cod varchar(128), in dat DATE)

BEGIN

declare tempID INT;

declare exit handler for sqlexception

begin

rollback;

resignal;

end;

set transaction isolation level read committed;

start transaction;

select `ID` from `Dipendente` where `Username` = var\_username and `Ruolo` = 'manager' into tempID;

insert into `PropostaCommerciale` (`CodiceAlfanumerico`, `DataDiAttivazione`, `Validità`)

values (cod, dat,'Attiva');

insert into `Gestione` (`Manager`, `CodiceProposta`)

values (tempID, cod);

commit;

END

* La sesta serve per inserire un contatto:

CREATE DEFINER=`root`@`localhost` PROCEDURE `inserire\_contatto`(out codice\_contatto INT, in cliente varchar(128), in funzionario INT)

BEGIN

insert into `Contatto` (`ClienteContattato`, `FunzionarioContattante`)

values (cliente, funzionario);

set codice\_contatto = last\_insert\_id();

END

* La settima serve per inserire una nota, andando a distinguere se la nota si riferisce ad un contatto o ad un appuntamento:

CREATE DEFINER=`root`@`localhost` PROCEDURE `inserire\_nota`(in id INT, in testo varchar(4096), in func INT, in input INT, in cod varchar(128))

BEGIN

declare exit handler for sqlexception

begin

rollback;

resignal;

end;

set transaction isolation level repeatable read;

start transaction;

if ((select count(\*) from Contatto where Contatto.ID = input) > 0) then insert into Nota (`Numero`, `Funzionario`, `Testo`, `ContattoAssociato`, `AppuntamentoID`) values (id, func, testo, input, null);

else insert into Nota (`Numero`, `Funzionario`, `Testo`, `ContattoAssociato`, `AppuntamentoID`) values (id, func, testo, null, input);

end if;

insert into Relazione (`Nota`, `CodiceProposta`) values(id, cod);

commit;

END

* L’ottava va ad offrire un’opportunità ad un cliente correlando con una consulenza precedentemente inserita nel sistema:

CREATE DEFINER=`root`@`localhost` PROCEDURE `inserire\_opportunità`(in CodFis varchar(128), in CodPr varchar(128))

BEGIN

insert into `Correlato` (`Cliente`, `CodiceProposta`)

values (CodFis, CodPr);

END

* La nona inserisce un nuovo dipendente dell’azienda all’interno del sistema restituendone l’identificativo:

CREATE DEFINER=`root`@`localhost` PROCEDURE `inserire\_utente`(in username varchar(45), in pass varchar(45), in rol varchar(45), out codice INT)

BEGIN

declare exit handler for sqlexception

begin

rollback;

resignal;

end;

set transaction isolation level repeatable read;

start transaction;

insert into `Dipendente` (`Username`, `Password`, `Ruolo`)

values (username, pass, rol);

commit;

set codice = last\_insert\_id();

END

* La decima permette di far terminare una consulenza attualmente attiva:

CREATE DEFINER=`root`@`localhost` PROCEDURE `invalidare\_consulenza`(in cod varchar(128))

BEGIN

update PropostaCommerciale set `Validità` = 'Terminata', `DataDiScadenza` = CURDATE(), `DataDiAttivazione` = NULL where `CodiceAlfanumerico` = cod;

END

END

* L’undicesima viene usata per riconoscere il ruolo dell’utente che ha appena eseguito l’accesso al sistema:

CREATE DEFINER=`root`@`localhost` PROCEDURE `login`(in var\_username varchar(45), in var\_pass varchar(45), out var\_role INT)

BEGIN

declare var\_user\_role ENUM('utente', 'manager', 'commerciale');

select `Ruolo` from `Dipendente`

where `Username` = var\_username

and `Password` = var\_pass

into var\_user\_role;

-- See the corresponding enum in the client

if var\_user\_role = 'utente' then

set var\_role = 1;

elseif var\_user\_role = 'manager' then

set var\_role = 2;

elseif var\_user\_role = 'commerciale' then

set var\_role = 3;

else

set var\_role = 4;

end if;

END

* La dodicesima permette di modificare il testo di una nota precedentemente inserita:

CREATE DEFINER=`root`@`localhost` PROCEDURE `modificare\_nota`(in testo varchar(4096), in id INT)

BEGIN

update Nota set `Testo` = testo where `Numero` = id;

END

* La tredicesima, quattordicesima e quindicesima servono per ottenere informazioni varie su un determinato cliente, rispettivamente le proposte commerciali da lui acquistate, le sue informazioni personali e le note correlate:

CREATE DEFINER=`root`@`localhost` PROCEDURE `visualizzare\_cliente\_acquisti`(in cod\_fisc varchar(128))

BEGIN

select CodiceProposta from Acquisto where `Cliente` = cod\_fisc;

END

CREATE DEFINER=`root`@`localhost` PROCEDURE `visualizzare\_cliente\_dati`(in cod\_fisc varchar(128))

BEGIN

select \* from Cliente where `CodiceFiscale` = cod\_fisc;

END

CREATE DEFINER=`root`@`localhost` PROCEDURE `visualizzare\_cliente\_note`(in cod\_fisc varchar(128))

BEGIN

select n.Numero

from Nota as n left join Contatto as c on n.ContattoAssociato = c.ID

left join Appuntamento as a on n.AppuntamentoID = a.ID

where c.ClienteContattato = cod\_fisc or a.PartecipazioneCliente = cod\_fisc;

END

* La sedicesima permette di visualizzare i clienti assegnati ad un determinato funzionario:

CREATE DEFINER=`root`@`localhost` PROCEDURE `visualizzare\_clienti\_assegnati`(in ID INT)

BEGIN

select `CodiceFiscale` from Cliente where Cliente.IDFunz = ID;

END

* La diciassettesima serve per ottenere le proposte attualmente ancora attive avanzate da un manager:

CREATE DEFINER=`root`@`localhost` PROCEDURE `visualizzare\_consulenze\_attive`(in cod INT)

BEGIN

select `CodiceProposta` from Gestione join PropostaCommerciale on `CodiceProposta` = `CodiceAlfanumerico` where `Manager` = cod and `Validità` = 'Attiva';

END

# Appendice: Implementazione

## Codice SQL per istanziare il database

Riportare il codice SQL necessario ad istanziare lo schema del DB. Le stored procedure, le viste, i trigger, gli eventi e tutto quello che è stato già inserito all’interno della relazione di progetto nelle sezioni precedenti non deve essere inserito in questa appendice.

Sì, avete letto bene: **riportare il codice SQL**. Frasi del tipo “il codice è nel file allegato” non rispondono alla richiesta di riportare il codice SQL.

## CREATE TABLE IF NOT EXISTS `azienda`.`Acquisto` (

`Cliente` VARCHAR(128) NOT NULL,

`CodiceProposta` VARCHAR(128) NOT NULL,

PRIMARY KEY (`Cliente`, `CodiceProposta`),

INDEX `fk\_Cliente\_idx` (`Cliente` ASC) VISIBLE,

INDEX `fk\_PropostaCommerciale\_idx` (`CodiceProposta` ASC) VISIBLE,

CONSTRAINT `fk\_Cliente`

FOREIGN KEY (`Cliente`)

REFERENCES `azienda`.`Cliente` (`CodiceFiscale`),

CONSTRAINT `fk\_PropostaCommerciale`

FOREIGN KEY (`CodiceProposta`)

REFERENCES `azienda`.`PropostaCommerciale` (`CodiceAlfanumerico`)

ON DELETE RESTRICT

ON UPDATE RESTRICT)

ENGINE = InnoDB

DEFAULT CHARACTER SET = utf8mb4

COLLATE = utf8mb4\_0900\_ai\_ci;

CREATE TABLE IF NOT EXISTS `azienda`.`Appuntamento` (

`Giorno` DATE NOT NULL,

`Ora` TIME NOT NULL,

`IndirizzoSede` VARCHAR(128) NOT NULL,

`AdesioneFunzionario` INT NOT NULL,

`PartecipazioneCliente` VARCHAR(128) NOT NULL,

`ID` INT NOT NULL AUTO\_INCREMENT,

PRIMARY KEY (`ID`),

UNIQUE INDEX `UNIQUE` (`Giorno` ASC, `Ora` ASC, `IndirizzoSede` ASC) VISIBLE,

INDEX `fk\_Appuntamento\_idx` (`AdesioneFunzionario` ASC) VISIBLE,

INDEX `fk\_ID\_Cliente\_idx` (`PartecipazioneCliente` ASC) VISIBLE,

INDEX `fk\_Sede\_idx` (`IndirizzoSede` ASC) VISIBLE,

CONSTRAINT `fk\_ID\_Cliente`

FOREIGN KEY (`PartecipazioneCliente`)

REFERENCES `azienda`.`Cliente` (`CodiceFiscale`)

ON DELETE RESTRICT

ON UPDATE CASCADE,

CONSTRAINT `fk\_ID\_Dipendente`

FOREIGN KEY (`AdesioneFunzionario`)

REFERENCES `azienda`.`Dipendente` (`ID`),

CONSTRAINT `fk\_Sede`

FOREIGN KEY (`IndirizzoSede`)

REFERENCES `azienda`.`Sede` (`Indirizzo`))

ENGINE = InnoDB

AUTO\_INCREMENT = 0

DEFAULT CHARACTER SET = utf8mb4

COLLATE = utf8mb4\_0900\_ai\_ci;

CREATE TABLE IF NOT EXISTS `azienda`.`Cliente` (

`CodiceFiscale` VARCHAR(128) NOT NULL,

`Nome` VARCHAR(45) NOT NULL,

`Cognome` VARCHAR(45) NOT NULL,

`DataDiNascita` DATE NOT NULL,

`DataDiRegistrazione` DATE NOT NULL,

`Fax` INT NULL DEFAULT NULL,

`Email` VARCHAR(128) NULL DEFAULT NULL,

`Indirizzo` VARCHAR(128) NULL DEFAULT NULL,

`RecapitoTelefonico` INT NULL DEFAULT NULL,

`IDComm` INT NULL DEFAULT NULL,

`IDFunz` INT NULL DEFAULT NULL,

`PartitaIVA` VARCHAR(45) NULL DEFAULT NULL,

PRIMARY KEY (`CodiceFiscale`),

INDEX `fk\_ID\_Comm\_idx` (`IDComm` ASC) VISIBLE,

INDEX `fk\_ID\_Funz\_idx` (`IDFunz` ASC) VISIBLE,

CONSTRAINT `fk\_Cliente\_1`

FOREIGN KEY (`IDComm`)

REFERENCES `azienda`.`Dipendente` (`ID`)

ON DELETE SET NULL

ON UPDATE CASCADE,

CONSTRAINT `fk\_Cliente\_2`

FOREIGN KEY (`IDFunz`)

REFERENCES `azienda`.`Dipendente` (`ID`)

ON DELETE SET NULL

ON UPDATE CASCADE)

ENGINE = InnoDB

DEFAULT CHARACTER SET = utf8mb4

COLLATE = utf8mb4\_0900\_ai\_ci;

CREATE TABLE IF NOT EXISTS `azienda`.`Contatto` (

`ID` INT NOT NULL AUTO\_INCREMENT,

`ClienteContattato` VARCHAR(128) NOT NULL,

`FunzionarioContattante` INT NOT NULL,

PRIMARY KEY (`ID`),

INDEX `fk\_Dipendente\_idx` (`FunzionarioContattante` ASC) VISIBLE,

INDEX `fk\_Cliente\_idx` (`ClienteContattato` ASC) VISIBLE,

CONSTRAINT `fk\_Cliente\_idx`

FOREIGN KEY (`ClienteContattato`)

REFERENCES `azienda`.`Cliente` (`CodiceFiscale`),

CONSTRAINT `fk\_Funzionario\_idx`

FOREIGN KEY (`FunzionarioContattante`)

REFERENCES `azienda`.`Dipendente` (`ID`))

ENGINE = InnoDB

AUTO\_INCREMENT = 1

DEFAULT CHARACTER SET = utf8mb4

COLLATE = utf8mb4\_0900\_ai\_ci;

CREATE TABLE IF NOT EXISTS `azienda`.`Correlato` (

`Cliente` VARCHAR(128) NOT NULL,

`CodiceProposta` VARCHAR(128) NOT NULL,

PRIMARY KEY (`Cliente`, `CodiceProposta`),

INDEX `fk\_Cliente\_idx` (`Cliente` ASC) VISIBLE,

INDEX `fk\_PropostaCliente\_idx` (`CodiceProposta` ASC) VISIBLE,

CONSTRAINT `fk\_Client\_idx`

FOREIGN KEY (`Cliente`)

REFERENCES `azienda`.`Cliente` (`CodiceFiscale`),

CONSTRAINT `fk\_PropostaCliente`

FOREIGN KEY (`CodiceProposta`)

REFERENCES `azienda`.`PropostaCommerciale` (`CodiceAlfanumerico`))

ENGINE = InnoDB

DEFAULT CHARACTER SET = utf8mb4

COLLATE = utf8mb4\_0900\_ai\_ci;

CREATE TABLE IF NOT EXISTS `azienda`.`Dipendente` (

`ID` INT NOT NULL AUTO\_INCREMENT,

`Username` VARCHAR(45) NOT NULL,

`Password` VARCHAR(45) NOT NULL,

`Ruolo` ENUM('utente', 'manager', 'commerciale') NOT NULL,

PRIMARY KEY (`ID`),

UNIQUE INDEX `Username\_UNIQUE` (`Username` ASC) VISIBLE)

ENGINE = InnoDB

AUTO\_INCREMENT = 0

DEFAULT CHARACTER SET = utf8mb4

COLLATE = utf8mb4\_0900\_ai\_ci;

CREATE TABLE IF NOT EXISTS `azienda`.`Gestione` (

`Manager` INT NOT NULL,

`CodiceProposta` VARCHAR(128) NOT NULL,

PRIMARY KEY (`Manager`, `CodiceProposta`),

UNIQUE INDEX `CodiceProposta\_UNIQUE` (`CodiceProposta` ASC) VISIBLE,

CONSTRAINT `fk\_Manager`

FOREIGN KEY (`Manager`)

REFERENCES `azienda`.`Dipendente` (`ID`),

CONSTRAINT `fk\_Proposta`

FOREIGN KEY (`CodiceProposta`)

REFERENCES `azienda`.`PropostaCommerciale` (`CodiceAlfanumerico`))

ENGINE = InnoDB

DEFAULT CHARACTER SET = utf8mb4

COLLATE = utf8mb4\_0900\_ai\_ci;

CREATE TABLE IF NOT EXISTS `azienda`.`Nota` (

`Numero` INT NOT NULL,

`Funzionario` INT NOT NULL,

`Testo` VARCHAR(4096) NOT NULL,

`ContattoAssociato` INT NULL DEFAULT NULL,

`AppuntamentoID` INT NULL DEFAULT NULL,

PRIMARY KEY (`Numero`),

INDEX `fk\_Dipendente\_idx` (`Funzionario` ASC) VISIBLE,

INDEX `fk\_Contatto\_idx` (`ContattoAssociato` ASC) VISIBLE,

INDEX `fk\_Appuntamento\_idx` (`AppuntamentoID` ASC) VISIBLE,

CONSTRAINT `fk\_Nota\_1`

FOREIGN KEY (`Funzionario`)

REFERENCES `azienda`.`Dipendente` (`ID`),

CONSTRAINT `fk\_Nota\_2`

FOREIGN KEY (`ContattoAssociato`)

REFERENCES `azienda`.`Contatto` (`ID`),

CONSTRAINT `fk\_Nota\_3`

FOREIGN KEY (`AppuntamentoID`)

REFERENCES `azienda`.`Appuntamento` (`ID`))

ENGINE = InnoDB

DEFAULT CHARACTER SET = utf8mb4

COLLATE = utf8mb4\_0900\_ai\_ci;

CREATE TABLE IF NOT EXISTS `azienda`.`PropostaCommerciale` (

`CodiceAlfanumerico` VARCHAR(128) NOT NULL,

`DataDiAttivazione` DATE NULL DEFAULT NULL,

`DataDiScadenza` DATE NULL DEFAULT NULL,

`Validità` ENUM('Attiva', 'Terminata') NOT NULL,

PRIMARY KEY (`CodiceAlfanumerico`))

ENGINE = InnoDB

DEFAULT CHARACTER SET = utf8mb4

COLLATE = utf8mb4\_0900\_ai\_ci;

CREATE TABLE IF NOT EXISTS `azienda`.`Relazione` (

`Nota` INT NOT NULL,

`CodiceProposta` VARCHAR(128) NOT NULL,

PRIMARY KEY (`Nota`, `CodiceProposta`),

INDEX `fk\_Nota\_idx` (`Nota` ASC) VISIBLE,

INDEX `fk\_Relazione\_2\_idx` (`CodiceProposta` ASC) VISIBLE,

CONSTRAINT `fk\_Relazione\_1`

FOREIGN KEY (`Nota`)

REFERENCES `azienda`.`Nota` (`Numero`),

CONSTRAINT `fk\_Relazione\_2`

FOREIGN KEY (`CodiceProposta`)

REFERENCES `azienda`.`PropostaCommerciale` (`CodiceAlfanumerico`))

ENGINE = InnoDB

DEFAULT CHARACTER SET = utf8mb4

COLLATE = utf8mb4\_0900\_ai\_ci;

CREATE TABLE IF NOT EXISTS `azienda`.`Sede` (

`Indirizzo` VARCHAR(128) NOT NULL,

`SalaRiunione` VARCHAR(45) NOT NULL,

PRIMARY KEY (`Indirizzo`))

ENGINE = InnoDB

DEFAULT CHARACTER SET = utf8mb4

COLLATE = utf8mb4\_0900\_ai\_ci;

## Codice del Front-End

Riportare (correttamente formattato) il codice C del thin client realizzato per interagire con la base di dati.

Sì, avete letto bene: **riportare il codice C**. Frasi del tipo “il codice è nei file allegati” non rispondono alla richiesta di riportare il codice C.

commerciale.c

#include <stdio.h>

#include "db.h"

#include "commerciale.h"

#include "io.h"

#include "validation.h"

int get\_comm\_action(void)

{

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

char op;

clear\_screen();

puts("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

puts("\* COMMERCIAL'S DASHBOARD \*");

puts("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

puts("\*\*\* What should I do for you? \*\*\*\n");

puts("1) Register a new client");

puts("2) Quit");

op = multi\_choice("Select an option", options, 2);

return op - '1';

}

void get\_client\_inf(struct client \*client)

{

clear\_screen();

puts("\*\* Get client's information \*\*\n");

get\_input("Insert FiscalCode: ", ID\_LEN, client->code, false);

get\_input("Insert first name: ", NAME\_LEN, client->name, false);

get\_input("Insert surname: ", SURNAME\_LEN, client->surname, false);

while(true) {

if(validate\_date(get\_input("Insert birthday [YYYY-MM-DD]: ", DATE\_LEN, client->birthday, false)))

break;

fprintf(stderr, "Invalid date!\n");

}

while(true) {

if(validate\_date(get\_input("Insert registration date [YYYY-MM-DD]: ", DATE\_LEN, client->registration, false)))

break;

fprintf(stderr, "Invalid date!\n");

}

while(true) {

if(atoi(get\_input("Insert fax: ", NUMBER\_LEN, client->fax, false)) != 0)

break;

fprintf(stderr, "Invalid fax!\n");

}

get\_input("Insert email: ", NUMBER\_LEN, client->email, false);

get\_input("Insert address: ", NUMBER\_LEN, client->address, false);

while(true) {

if(atoi(get\_input("Insert phone: ", NUMBER\_LEN, client->phone, false)) != 0)

break;

fprintf(stderr, "Invalid phone!\n");

}

get\_input("Insert user's ID who is adding client: ", ID\_LEN, client->comm, false);

get\_input("Insert user's ID who is related to client: ", ID\_LEN, client->funz, false);

get\_input("Insert user's IVA if he/she has one: ", NAME\_LEN, client->IVA, false);

}

commerciale.h

#pragma once

#include "db.h"

enum actions {

CLIENT,

QUIT,

END\_OF\_ACTIONS

};

int get\_comm\_action(void);

void get\_client\_inf(struct client \*client);

commerciale\_app.c

#include <stdio.h>

#include <string.h>

#include "db.h"

#include "commerciale.h"

#include "commerciale\_app.h"

#include "io.h"

static bool register\_client(void)

{

struct client client;

memset(&client, 0, sizeof(client));

get\_client\_inf(&client);

do\_register\_client(&client);

return false;

}

static bool quit(void)

{

return true;

}

static struct {

enum actions action;

bool (\*control)(void);

} controls[END\_OF\_ACTIONS] = {

{.action = CLIENT, .control = register\_client},

{.action = QUIT, .control = quit}

};

void controller\_commerciale(void)

{

db\_switch\_to\_comm();

while(true) {

int action = get\_comm\_action();

if(action >= END\_OF\_ACTIONS) {

fprintf(stderr, "Error: unknown action\n");

continue;

}

if (controls[action].control())

break;

press\_anykey();

}

}

commerciale\_app.h

#pragma once

void controller\_commerciale(void);

db.c

#include <stdlib.h>

#include <stdio.h>

#include <string.h>

#include <mysql/mysql.h>

#include <assert.h>

#include "db.h"

#include "db\_prof.h"

static MYSQL \*conn;

static MYSQL\_STMT \*login\_procedure;

static MYSQL\_STMT \*agenda;

static MYSQL\_STMT \*client\_data;

static MYSQL\_STMT \*client\_notes;

static MYSQL\_STMT \*client\_bought;

static MYSQL\_STMT \*clients\_bound;;

static MYSQL\_STMT \*ins\_note;

static MYSQL\_STMT \*mod\_note;

static MYSQL\_STMT \*del\_note;

static MYSQL\_STMT \*opportunity;

static MYSQL\_STMT \*reg\_appointment;

static MYSQL\_STMT \*users;

static MYSQL\_STMT \*consultation;

static MYSQL\_STMT \*expire;

static MYSQL\_STMT \*get\_consultation;

static MYSQL\_STMT \*clients;

static MYSQL\_STMT \*reg\_contact;

static void close\_prepared\_stmts(void)

{

if(login\_procedure) {

mysql\_stmt\_close(login\_procedure);

login\_procedure = NULL;

}

if(agenda) {

mysql\_stmt\_close(agenda);

agenda = NULL;

}

if(client\_data) {

mysql\_stmt\_close(client\_data);

client\_data = NULL;

}

if(client\_notes) {

mysql\_stmt\_close(client\_notes);

client\_notes = NULL;

}

if(client\_bought) {

mysql\_stmt\_close(client\_bought);

client\_bought = NULL;

}

if(clients\_bound) {

mysql\_stmt\_close(clients\_bound);

clients\_bound = NULL;

}

if(ins\_note) {

mysql\_stmt\_close(ins\_note);

ins\_note = NULL;

}

if(mod\_note) {

mysql\_stmt\_close(mod\_note);

mod\_note = NULL;

}

if(del\_note) {

mysql\_stmt\_close(del\_note);

del\_note = NULL;

}

if(opportunity) {

mysql\_stmt\_close(opportunity);

opportunity = NULL;

}

if(reg\_appointment) {

mysql\_stmt\_close(reg\_appointment);

reg\_appointment = NULL;

}

if(users) {

mysql\_stmt\_close(users);

users = NULL;

}

if(consultation) {

mysql\_stmt\_close(consultation);

consultation = NULL;

}

if(expire) {

mysql\_stmt\_close(expire);

expire = NULL;

}

if(get\_consultation) {

mysql\_stmt\_close(get\_consultation);

get\_consultation = NULL;

}

if(clients) {

mysql\_stmt\_close(clients);

clients = NULL;

}

if(reg\_contact) {

mysql\_stmt\_close(reg\_contact);

reg\_contact = NULL;

}

}

static bool initialize\_prepared\_stmts(role\_t for\_role)

{

switch(for\_role) {

case LOGIN\_ROLE:

if(!setup\_prepared\_stmt(&login\_procedure, "call login(?, ?, ?)", conn)) {

print\_stmt\_error(login\_procedure, "Unable to initialize login statement\n");

return false;

}

break;

case UTENTE:

if(!setup\_prepared\_stmt(&agenda, "call agenda(?)", conn)) {

print\_stmt\_error(agenda, "Unable to initialize agenda statement\n");

return false;

}

if(!setup\_prepared\_stmt(&client\_data, "call visualizzare\_cliente\_dati(?)", conn)) {

print\_stmt\_error(client\_data, "Unable to initialize get client statement\n");

return false;

}

if(!setup\_prepared\_stmt(&client\_notes, "call visualizzare\_cliente\_note(?)", conn)) {

print\_stmt\_error(client\_notes, "Unable to initialize get client statement\n");

return false;

}

if(!setup\_prepared\_stmt(&client\_bought, "call visualizzare\_cliente\_acquisti(?)", conn)) {

print\_stmt\_error(client\_bought, "Unable to initialize get client statement\n");

return false;

}

if(!setup\_prepared\_stmt(&clients\_bound, "call visualizzare\_clienti\_assegnati(?)", conn)) {

print\_stmt\_error(clients\_bound, "Unable to initialize get user's clients statement\n");

return false;

}

if(!setup\_prepared\_stmt(&ins\_note, "call inserire\_nota(?, ?, ?, ?, ?)", conn)) {

print\_stmt\_error(ins\_note, "Unable to initialize notes statement\n");

return false;

}

if(!setup\_prepared\_stmt(&mod\_note, "call modificare\_nota(?, ?)", conn)) {

print\_stmt\_error(mod\_note, "Unable to initialize notes statement\n");

return false;

}

if(!setup\_prepared\_stmt(&del\_note, "call eliminare\_nota(?)", conn)) {

print\_stmt\_error(del\_note, "Unable to initialize notes statement\n");

return false;

}

if(!setup\_prepared\_stmt(&opportunity, "call inserire\_opportunità(?, ?)", conn)) {

print\_stmt\_error(opportunity, "Unable to initialize new opportunities statement\n");

return false;

}

if(!setup\_prepared\_stmt(&reg\_appointment, "call inserire\_appuntamento(?, ?, ?, ?, ?, ?)", conn)) {

print\_stmt\_error(reg\_appointment, "Unable to initialize new appointments statement\n");

return false;

}

if(!setup\_prepared\_stmt(&reg\_contact, "call inserire\_contatto(?, ?, ?)", conn)) {

print\_stmt\_error(reg\_contact, "Unable to initialize new appointments statement\n");

return false;

}

break;

case MANAGER:

if(!setup\_prepared\_stmt(&users, "call inserire\_utente(?, ?, ?, ?)", conn)) {

print\_stmt\_error(users, "Unable to initialize new users statement\n");

return false;

}

if(!setup\_prepared\_stmt(&consultation, "call inserire\_consulenza(?, ?, ?)", conn)) {

print\_stmt\_error(consultation, "Unable to initialize new consultations statement\n");

return false;

}

if(!setup\_prepared\_stmt(&expire, "call invalidare\_consulenza(?)", conn)) {

print\_stmt\_error(expire, "Unable to initialize new consultations statement\n");

return false;

}

if(!setup\_prepared\_stmt(&get\_consultation, "call visualizzare\_consulenze\_attive(?)", conn)) {

print\_stmt\_error(get\_consultation, "Unable to initialize new consultations statement\n");

return false;

}

break;

case COMMERCIALE:

if(!setup\_prepared\_stmt(&clients, "call inserire\_cliente(?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)", conn)) {

print\_stmt\_error(clients, "Unable to initialize new clients statement\n");

return false;

}

break;

default:

fprintf(stderr, "[FATAL] Unexpected role to prepare statements.\n");

exit(EXIT\_FAILURE);

}

return true;

}

bool init\_db(void)

{

unsigned int timeout = 300;

bool reconnect = true;

conn = mysql\_init(NULL);

if(conn == NULL) {

finish\_with\_error(conn, "mysql\_init() failed (probably out of memory)\n");

}

if(mysql\_real\_connect(conn, "localhost", "login", "login", "azienda",

3306, NULL,

CLIENT\_MULTI\_STATEMENTS | CLIENT\_MULTI\_RESULTS | CLIENT\_COMPRESS | CLIENT\_INTERACTIVE | CLIENT\_REMEMBER\_OPTIONS) == NULL) {

finish\_with\_error(conn, "mysql\_real\_connect() failed\n");

}

if (mysql\_options(conn, MYSQL\_OPT\_CONNECT\_TIMEOUT, &timeout)) {

print\_error(conn, "[mysql\_options] failed.");

}

if(mysql\_options(conn, MYSQL\_OPT\_RECONNECT, &reconnect)) {

print\_error(conn, "[mysql\_options] failed.");

}

#ifndef NDEBUG

mysql\_debug("d:t:O,/tmp/client.trace");

if(mysql\_dump\_debug\_info(conn)) {

print\_error(conn, "[debug\_info] failed.");

}

#endif

return initialize\_prepared\_stmts(LOGIN\_ROLE);

}

void fini\_db(void)

{

close\_prepared\_stmts();

mysql\_close(conn);

}

role\_t attempt\_login(struct credentials \*cred)

{

MYSQL\_BIND param[3]; // Used both for input and output

int role = 0;

// Prepare parameters

set\_binding\_param(&param[0], MYSQL\_TYPE\_VAR\_STRING, cred->username, strlen(cred->username));

set\_binding\_param(&param[1], MYSQL\_TYPE\_VAR\_STRING, cred->password, strlen(cred->password));

set\_binding\_param(&param[2], MYSQL\_TYPE\_LONG, &role, sizeof(role));

if(mysql\_stmt\_bind\_param(login\_procedure, param) != 0) { // Note \_param

print\_stmt\_error(login\_procedure, "Could not bind parameters for login");

role = FAILED\_LOGIN;

goto out;

}

// Run procedure

if(mysql\_stmt\_execute(login\_procedure) != 0) {

print\_stmt\_error(login\_procedure, "Could not execute login procedure");

role = FAILED\_LOGIN;

goto out;

}

// Prepare output parameters

set\_binding\_param(&param[0], MYSQL\_TYPE\_LONG, &role, sizeof(role));

if(mysql\_stmt\_bind\_result(login\_procedure, param)) {

print\_stmt\_error(login\_procedure, "Could not retrieve output parameter");

role = FAILED\_LOGIN;

goto out;

}

// Retrieve output parameter

if(mysql\_stmt\_fetch(login\_procedure)) {

print\_stmt\_error(login\_procedure, "Could not buffer results");

role = FAILED\_LOGIN;

goto out;

}

out:

// Consume the possibly-returned table for the output parameter

// while(mysql\_stmt\_next\_result(login\_procedure) != -1) {}

mysql\_stmt\_free\_result(login\_procedure);

mysql\_stmt\_reset(login\_procedure);

return role;

}

void db\_switch\_to\_login(void)

{

close\_prepared\_stmts();

/\*

if(mysql\_change\_user(conn, getenv("LOGIN\_USER"), getenv("LOGIN\_PASS"), getenv("DB"))) {

fprintf(stderr, "mysql\_change\_user() failed: %s\n", mysql\_error(conn));

exit(EXIT\_FAILURE);

}

\*/

mysql\_close(conn);

conn = mysql\_init(NULL);

if(mysql\_real\_connect(conn, "localhost", "login", "login", "azienda",

3306, NULL,

CLIENT\_MULTI\_STATEMENTS | CLIENT\_MULTI\_RESULTS | CLIENT\_COMPRESS | CLIENT\_INTERACTIVE | CLIENT\_REMEMBER\_OPTIONS) == NULL) {

finish\_with\_error(conn, "mysql\_real\_connect() failed\n");

}

if(!initialize\_prepared\_stmts(LOGIN\_ROLE)) {

fprintf(stderr, "[FATAL] Cannot initialize prepared statements.\n");

exit(EXIT\_FAILURE);

}

}

void db\_switch\_to\_utente(void)

{

close\_prepared\_stmts();

/\*

if(mysql\_change\_user(conn, getenv("ADMINISTRATOR\_USER"), getenv("ADMINISTRATOR\_PASS"), getenv("DB"))) {

fprintf(stderr, "mysql\_change\_user() failed: %s\n", mysql\_error(conn));

exit(EXIT\_FAILURE);

}

\*/

mysql\_close(conn);

conn = mysql\_init(NULL);

if(mysql\_real\_connect(conn, "localhost", "utente", "utente", "azienda",

3306, NULL,

CLIENT\_MULTI\_STATEMENTS | CLIENT\_MULTI\_RESULTS | CLIENT\_COMPRESS | CLIENT\_INTERACTIVE | CLIENT\_REMEMBER\_OPTIONS) == NULL) {

finish\_with\_error(conn, "mysql\_real\_connect() failed\n");

}

if(!initialize\_prepared\_stmts(UTENTE)) {

fprintf(stderr, "[FATAL] Cannot initialize prepared statements.\n");

exit(EXIT\_FAILURE);

}

}

void do\_register\_client(struct client \*client)

{

MYSQL\_BIND param[12];

MYSQL\_TIME giornonasc;

MYSQL\_TIME giornoreg;

int fax, phone, comm, funz;

date\_to\_mysql\_time(client->birthday, &giornonasc);

date\_to\_mysql\_time(client->registration, &giornoreg);

fax = atoi(client->fax);

phone = atoi(client->phone);

comm = atoi(client->comm);

funz = atoi(client->funz);

// Bind parameters

set\_binding\_param(&param[0], MYSQL\_TYPE\_VAR\_STRING, client->code, sizeof(client->code));

set\_binding\_param(&param[1], MYSQL\_TYPE\_VAR\_STRING, client->name, sizeof(client->name));

set\_binding\_param(&param[2], MYSQL\_TYPE\_VAR\_STRING, client->surname, sizeof(client->surname));

set\_binding\_param(&param[3], MYSQL\_TYPE\_TIME, &giornonasc, sizeof(giornonasc));

set\_binding\_param(&param[4], MYSQL\_TYPE\_TIME, &giornoreg, sizeof(giornoreg));

set\_binding\_param(&param[5], MYSQL\_TYPE\_LONG, &fax, sizeof(fax));

set\_binding\_param(&param[6], MYSQL\_TYPE\_VAR\_STRING, client->email, sizeof(client->email));

set\_binding\_param(&param[7], MYSQL\_TYPE\_VAR\_STRING, client->address, sizeof(client->address));

set\_binding\_param(&param[8], MYSQL\_TYPE\_LONG, &phone, sizeof(phone));

set\_binding\_param(&param[9], MYSQL\_TYPE\_LONG, &comm, sizeof(comm));

set\_binding\_param(&param[10], MYSQL\_TYPE\_LONG, &funz, sizeof(funz));

set\_binding\_param(&param[11], MYSQL\_TYPE\_VAR\_STRING, client->IVA, sizeof(client->IVA));

if(mysql\_stmt\_bind\_param(clients, param) != 0) {

print\_stmt\_error(clients, "Could not bind parameters for do\_register\_client");

return;

}

// Run procedure

if(mysql\_stmt\_execute(clients) != 0) {

print\_stmt\_error(clients, "Could not execute register clients procedure");

return;

}

mysql\_stmt\_free\_result(clients);

mysql\_stmt\_reset(clients);

return;

}

int do\_register\_appointment(struct appointment \*app)

{

MYSQL\_BIND param[6];

MYSQL\_TIME day;

MYSQL\_TIME time;

int id;

int ret;

int status;

date\_to\_mysql\_time(app->day, &day);

time\_to\_mysql\_time(app->time, &time);

id = atoi(app->func);

set\_binding\_param(&param[0], MYSQL\_TYPE\_DATE, &day, sizeof(day));

set\_binding\_param(&param[1], MYSQL\_TYPE\_TIME, &time, sizeof(time));

set\_binding\_param(&param[2], MYSQL\_TYPE\_VAR\_STRING, app->place, sizeof(app->place));

set\_binding\_param(&param[3], MYSQL\_TYPE\_LONG, &id, sizeof(id));

set\_binding\_param(&param[4], MYSQL\_TYPE\_VAR\_STRING, app->client, sizeof(app->client));

set\_binding\_param(&param[5], MYSQL\_TYPE\_LONG, &ret, sizeof(ret));

if(mysql\_stmt\_bind\_param(reg\_appointment, param) != 0) {

print\_stmt\_error(reg\_appointment, "Could not bind parameters for do\_register\_appointment");

ret = -1;

goto out;

}

if(mysql\_stmt\_execute(reg\_appointment) != 0){

print\_stmt\_error(reg\_appointment, "Could not execute register appointments procedure");

ret = -1;

goto out;

}

set\_binding\_param(&param[0], MYSQL\_TYPE\_LONG, &id, sizeof(id));

if(mysql\_stmt\_bind\_result(reg\_appointment, param)) {

print\_stmt\_error(reg\_appointment, "Could not bind output parameters");

ret = -1;

goto out;

}

if (mysql\_stmt\_store\_result(reg\_appointment)) {

print\_stmt\_error(reg\_appointment, "Unable to store appointment information result set.");

ret = -1;

goto out;

}

do{

while (true) {

status = mysql\_stmt\_fetch(reg\_appointment);

if (status == 1 || status == MYSQL\_NO\_DATA)

break;

ret = id;

}

}while(mysql\_stmt\_next\_result(reg\_appointment) == 0);

out:

mysql\_stmt\_free\_result(reg\_appointment);

mysql\_stmt\_reset(reg\_appointment);

return ret;

}

void db\_switch\_to\_manager(void)

{

close\_prepared\_stmts();

/\*

if(mysql\_change\_user(conn, getenv("AGENCY\_USER"), getenv("AGENCY\_PASS"), getenv("DB"))) {

fprintf(stderr, "mysql\_change\_user() failed: %s\n", mysql\_error(conn));

exit(EXIT\_FAILURE);

}

\*/

mysql\_close(conn);

conn = mysql\_init(NULL);

if(mysql\_real\_connect(conn, "localhost", "manager", "manager", "azienda",

3306, NULL,

CLIENT\_MULTI\_STATEMENTS | CLIENT\_MULTI\_RESULTS | CLIENT\_COMPRESS | CLIENT\_INTERACTIVE | CLIENT\_REMEMBER\_OPTIONS) == NULL) {

finish\_with\_error(conn, "mysql\_real\_connect() failed\n");

}

if(!initialize\_prepared\_stmts(MANAGER)) {

fprintf(stderr, "[FATAL] Cannot initialize prepared statements.\n");

exit(EXIT\_FAILURE);

}

}

void db\_switch\_to\_comm(void)

{

close\_prepared\_stmts();

/\*

if(mysql\_change\_user(conn, getenv("AGENCY\_USER"), getenv("AGENCY\_PASS"), getenv("DB"))) {

fprintf(stderr, "mysql\_change\_user() failed: %s\n", mysql\_error(conn));

exit(EXIT\_FAILURE);

}

\*/

mysql\_close(conn);

conn = mysql\_init(NULL);

if(mysql\_real\_connect(conn, "localhost", "commerciale", "commerciale", "azienda",

3306, NULL,

CLIENT\_MULTI\_STATEMENTS | CLIENT\_MULTI\_RESULTS | CLIENT\_COMPRESS | CLIENT\_INTERACTIVE | CLIENT\_REMEMBER\_OPTIONS) == NULL) {

finish\_with\_error(conn, "mysql\_real\_connect() failed\n");

}

if(!initialize\_prepared\_stmts(COMMERCIALE)) {

fprintf(stderr, "[FATAL] Cannot initialize prepared statements.\n");

exit(EXIT\_FAILURE);

}

}

struct appointments \*get\_agenda(struct user \*user)

{

MYSQL\_BIND param[1];

MYSQL\_BIND ret[3];

MYSQL\_TIME giorno;

MYSQL\_TIME ora;

char indirizzo[LOCATION\_LEN];

int row = 0;

int status;

int temp;

struct appointments \*appointments = NULL;

temp = atoi(user->id);

set\_binding\_param(&param[0], MYSQL\_TYPE\_LONG, &temp, sizeof(temp));

if(mysql\_stmt\_bind\_param(agenda, param) != 0){

print\_stmt\_error(agenda, "Could not bind input parameters");

goto out;

}

if(mysql\_stmt\_execute(agenda) != 0){

print\_stmt\_error(agenda, "Could not execute agenda procedure");

goto out;

}

mysql\_stmt\_store\_result(agenda);

appointments = malloc(sizeof(\*appointments) + sizeof(unsigned) + sizeof(struct appointment) \* mysql\_stmt\_num\_rows(agenda));

if(appointments == NULL)

goto out;

memset(appointments, 0, sizeof(\*appointments) + sizeof(unsigned) + sizeof(struct appointment) \* mysql\_stmt\_num\_rows(agenda));

appointments->num\_rows = mysql\_stmt\_num\_rows(agenda);

set\_binding\_param(&ret[0], MYSQL\_TYPE\_DATE, &giorno, sizeof(giorno));

set\_binding\_param(&ret[1], MYSQL\_TYPE\_TIME, &ora, sizeof(ora));

set\_binding\_param(&ret[2], MYSQL\_TYPE\_VAR\_STRING, indirizzo, sizeof(indirizzo));

if(mysql\_stmt\_bind\_result(agenda, ret)) {

print\_stmt\_error(agenda, "Unable to bind output parameters");

goto out;

}

while (1) {

status = mysql\_stmt\_fetch(agenda);

if (status == 1 || status == MYSQL\_NO\_DATA)

break;

mysql\_date\_to\_string(&giorno, appointments->appointment[row].day);

mysql\_time\_to\_string(&ora, appointments->appointment[row].time);

strcpy(appointments->appointment[row].place, indirizzo);

row++;

}

out:

while(mysql\_stmt\_next\_result(agenda) != -1) {}

mysql\_stmt\_free\_result(agenda);

mysql\_stmt\_reset(agenda);

return appointments;

}

void dispose\_agenda(struct appointments \*app)

{

free(app);

}

struct client \*\*get\_clients(struct user \*user)

{

MYSQL\_BIND param[1];

struct client \*\*clients;

int status, i, temp;

int row = 0;

char cod[ID\_LEN];

temp = atoi(user->id);

set\_binding\_param(&param[0], MYSQL\_TYPE\_LONG, &temp, sizeof(temp));

if(mysql\_stmt\_bind\_param(clients\_bound, param) != 0){

print\_stmt\_error(clients\_bound, "Could not bind input parameters");

clients = NULL;

goto out;

}

if(mysql\_stmt\_execute(clients\_bound) != 0){

print\_stmt\_error(clients\_bound, "Could not execute clients bound procedure");

clients = NULL;

goto out;

}

if (mysql\_stmt\_store\_result(clients\_bound)) {

print\_stmt\_error(clients\_bound, "Unable to store clients information result set.");

free(clients);

clients = NULL;

goto out;

}

clients = malloc(mysql\_stmt\_num\_rows(clients\_bound) \* sizeof(struct client\*));

if(clients == NULL)

goto out;

memset(clients, 0, sizeof(struct client\*) \* mysql\_stmt\_num\_rows(clients\_bound));

for(i = 0; i < mysql\_stmt\_num\_rows(clients\_bound); i++) {

clients[i] = malloc(sizeof(struct client));

memset(clients[i], 0, sizeof(struct client));

}

clients[i] = NULL;

set\_binding\_param(&param[0], MYSQL\_TYPE\_VAR\_STRING, cod, sizeof(cod));

if(mysql\_stmt\_bind\_result(clients\_bound, param)){

print\_stmt\_error(clients\_bound, "Unable to bind output parameters");

free(clients);

clients = NULL;

goto out;

}

while(true){

status = mysql\_stmt\_fetch(clients\_bound);

if (status == 1 || status == MYSQL\_NO\_DATA)

break;

strcpy(clients[row]->code, cod);

row++;

}

out:

while(mysql\_stmt\_next\_result(clients\_bound) != -1) {}

mysql\_stmt\_free\_result(clients\_bound);

mysql\_stmt\_reset(clients\_bound);

return clients;

}

void dispose\_clients(struct client \*\*clients)

{

free(clients);

}

void do\_register\_consultation(struct user \*user, struct cons \*cons)

{

MYSQL\_BIND param[3];

MYSQL\_TIME day;

date\_to\_mysql\_time(cons->day, &day);

set\_binding\_param(&param[0], MYSQL\_TYPE\_VAR\_STRING, user->credentials->username, sizeof(user->credentials->username));

set\_binding\_param(&param[1], MYSQL\_TYPE\_VAR\_STRING, cons->cod, sizeof(cons->cod));

set\_binding\_param(&param[2], MYSQL\_TYPE\_DATE, &day, sizeof(day));

if(mysql\_stmt\_bind\_param(consultation, param) != 0) {

print\_stmt\_error(consultation, "Unable to bind input parameters");

return;

}

if(mysql\_stmt\_execute(consultation) != 0){

print\_stmt\_error(consultation, "Unable to execute consultation registration procedure");

return;

}

mysql\_stmt\_free\_result(consultation);

mysql\_stmt\_reset(consultation);

return;

}

void do\_register\_opportunity(struct client \*client, struct cons \*cons)

{

MYSQL\_BIND param[2];

set\_binding\_param(&param[0], MYSQL\_TYPE\_VAR\_STRING, client->code, sizeof(client->code));

set\_binding\_param(&param[1], MYSQL\_TYPE\_VAR\_STRING, cons->cod, sizeof(cons->cod));

if (mysql\_stmt\_bind\_param(opportunity, param) != 0) {

print\_stmt\_error(opportunity, "Unable to bind input parameters");

return;

}

if (mysql\_stmt\_execute(opportunity) != 0) {

print\_stmt\_error(opportunity, "An error occurred while adding opportunity to a client.");

return;

}

mysql\_stmt\_free\_result(opportunity);

mysql\_stmt\_reset(opportunity);

return;

}

void do\_register\_note(struct note \*note, struct cons \*cons)

{

MYSQL\_BIND param[5];

int nb, fc, ca;

nb = atoi(note->number);

fc = atoi(note->func);

ca = atoi(note->input);

set\_binding\_param(&param[0], MYSQL\_TYPE\_LONG, &nb, sizeof(nb));

set\_binding\_param(&param[1], MYSQL\_TYPE\_VAR\_STRING, note->text, sizeof(note->text));

set\_binding\_param(&param[2], MYSQL\_TYPE\_LONG, &fc, sizeof(fc));

set\_binding\_param(&param[3], MYSQL\_TYPE\_LONG, &ca, sizeof(ca));

set\_binding\_param(&param[4], MYSQL\_TYPE\_VAR\_STRING, cons->cod, sizeof(cons->cod));

if (mysql\_stmt\_bind\_param(ins\_note, param) != 0) {

print\_stmt\_error(ins\_note, "Unable to bind input parameters");

return;

}

if (mysql\_stmt\_execute(ins\_note) != 0) {

print\_stmt\_error(ins\_note, "An error occurred while registering a note");

return;

}

mysql\_stmt\_free\_result(ins\_note);

mysql\_stmt\_reset(ins\_note);

return;

}

void do\_modify\_note(struct note \*note)

{

MYSQL\_BIND param[2];

int nb;

nb = atoi(note->number);

set\_binding\_param(&param[0], MYSQL\_TYPE\_VAR\_STRING, note->text, sizeof(note->text));

set\_binding\_param(&param[1], MYSQL\_TYPE\_LONG, &nb, sizeof(nb));

if (mysql\_stmt\_bind\_param(mod\_note, param) != 0) {

print\_stmt\_error(ins\_note, "Unable to bind input parameters");

return;

}

if (mysql\_stmt\_execute(mod\_note) != 0) {

print\_stmt\_error(ins\_note, "An error occurred while modifying a note.");

return;

}

mysql\_stmt\_free\_result(mod\_note);

mysql\_stmt\_reset(mod\_note);

return;

}

void do\_delete\_note(struct note \*note)

{

MYSQL\_BIND param[1];

int nb;

nb = atoi(note->number);

set\_binding\_param(&param[0], MYSQL\_TYPE\_LONG, &nb, sizeof(nb));

if (mysql\_stmt\_bind\_param(del\_note, param) != 0) {

print\_stmt\_error(del\_note, "Unable to bind input parameters");

return;

}

if (mysql\_stmt\_execute(del\_note) != 0) {

print\_stmt\_error(del\_note, "An error occurred while adding opportunity to a client.");

return;

}

mysql\_stmt\_free\_result(del\_note);

mysql\_stmt\_reset(del\_note);

return;

}

struct client \*get\_single\_client(char \*cod)

{

MYSQL\_BIND param[1];

int status;

MYSQL\_BIND ret[12];

struct client \*cl = NULL;

char CodiceFiscale[128];

char Nome[45];

char Cognome[45];

MYSQL\_TIME Data\_di\_nascita;

MYSQL\_TIME Data\_di\_registrazione;

int Fax;

char Email[45];

char Indirizzo[45];

int Recapito\_telefonico;

int IDComm;

int IDFunz;

char IVA[NAME\_LEN];

set\_binding\_param(&param[0], MYSQL\_TYPE\_VAR\_STRING, cod, sizeof(cod));

if (mysql\_stmt\_bind\_param(client\_data, param) != 0) {

print\_stmt\_error(client\_data, "Unable to bind input parameters");

cl = NULL;

goto out;

}

if (mysql\_stmt\_execute(client\_data) != 0) {

print\_stmt\_error(client\_data, "An error occurred while showing a client.");

cl = NULL;

goto out;

}

set\_binding\_param(&ret[0], MYSQL\_TYPE\_VAR\_STRING, CodiceFiscale, sizeof(CodiceFiscale));

set\_binding\_param(&ret[1], MYSQL\_TYPE\_VAR\_STRING, Nome, sizeof(Nome));

set\_binding\_param(&ret[2], MYSQL\_TYPE\_VAR\_STRING, Cognome, sizeof(Cognome));

set\_binding\_param(&ret[3], MYSQL\_TYPE\_DATE, &Data\_di\_nascita, sizeof(Data\_di\_nascita));

set\_binding\_param(&ret[4], MYSQL\_TYPE\_DATE, &Data\_di\_registrazione, sizeof(Data\_di\_registrazione));

set\_binding\_param(&ret[5], MYSQL\_TYPE\_LONG, &Fax, sizeof(Fax));

set\_binding\_param(&ret[6], MYSQL\_TYPE\_VAR\_STRING, Email, sizeof(Email));

set\_binding\_param(&ret[7], MYSQL\_TYPE\_VAR\_STRING, Indirizzo, sizeof(Indirizzo));

set\_binding\_param(&ret[8], MYSQL\_TYPE\_LONG, &Recapito\_telefonico, sizeof(Recapito\_telefonico));

set\_binding\_param(&ret[9], MYSQL\_TYPE\_LONG, &IDComm, sizeof(IDComm));

set\_binding\_param(&ret[10], MYSQL\_TYPE\_LONG, &IDFunz, sizeof(IDFunz));

set\_binding\_param(&ret[11], MYSQL\_TYPE\_VAR\_STRING, IVA, sizeof(IVA));

cl = malloc(sizeof(\*cl));

if(cl == NULL)

goto out;

memset(cl, 0, sizeof(\*cl));

if(mysql\_stmt\_bind\_result(client\_data, ret)) {

print\_stmt\_error(client\_data, "Unable to bind output parameters");

cl = NULL;

goto out;

}

while (1) {

status = mysql\_stmt\_fetch(client\_data);

if (status == 1 || status == MYSQL\_NO\_DATA)

break;

strcpy(cl->code, CodiceFiscale);

strcpy(cl->name, Nome);

strcpy(cl->surname, Cognome);

mysql\_date\_to\_string(&Data\_di\_nascita, cl->birthday);

mysql\_date\_to\_string(&Data\_di\_registrazione, cl->registration);

snprintf(cl->fax, NUMBER\_LEN, "%d", Fax);

strcpy(cl->email, Email);

strcpy(cl->address, Indirizzo);

snprintf(cl->phone, NUMBER\_LEN, "%d", Recapito\_telefonico);

snprintf(cl->comm, ID\_LEN, "%d", IDComm);

snprintf(cl->funz, ID\_LEN, "%d", IDFunz);

strcpy(cl->IVA, IVA);

}

out:

while(mysql\_stmt\_next\_result(client\_data) != -1) {}

mysql\_stmt\_free\_result(client\_data);

mysql\_stmt\_reset(client\_data);

return cl;

}

struct note \*\*get\_client\_note(char \*cod)

{

MYSQL\_BIND param[1];

int status, temp, i;

struct note \*\*nt = NULL;

int Numero;

set\_binding\_param(&param[0], MYSQL\_TYPE\_VAR\_STRING, cod, sizeof(cod));

if (mysql\_stmt\_bind\_param(client\_notes, param) != 0) {

print\_stmt\_error(client\_notes, "Unable to bind input parameters");

goto out;

}

if (mysql\_stmt\_execute(client\_notes) != 0) {

print\_stmt\_error(client\_notes, "An error occurred while showing client's notes.");

goto out;

}

if (mysql\_stmt\_store\_result(client\_notes)) {

print\_stmt\_error(client\_notes, "Unable to store client notes result set.");

goto out;

}

nt = malloc(sizeof(struct note\*) \* mysql\_stmt\_num\_rows(client\_notes));

if(nt == NULL)

goto out;

for(i = 0; i < mysql\_stmt\_num\_rows(client\_notes); i++) {

nt[i] = malloc(sizeof(struct note));

memset(nt[i], 0, sizeof(struct note));

}

nt[i] = NULL;

set\_binding\_param(&param[0], MYSQL\_TYPE\_LONG, &Numero, sizeof(Numero));

if(mysql\_stmt\_bind\_result(client\_notes, param)) {

print\_stmt\_error(client\_notes, "Unable to bind output parameters");

nt = NULL;

goto out;

}

temp = 0;

while (1) {

status = mysql\_stmt\_fetch(client\_notes);

if (status == 1 || status == MYSQL\_NO\_DATA)

break;

snprintf(nt[temp]->number, ID\_LEN, "%d", Numero);

temp++;

}

out:

while(mysql\_stmt\_next\_result(client\_notes) != -1) {}

mysql\_stmt\_free\_result(client\_notes);

mysql\_stmt\_reset(client\_notes);

return nt;

}

struct bought \*\*get\_client\_bought(char \*cod)

{

MYSQL\_BIND param[1];

int status, temp, i;

struct bought \*\*bt = NULL;

char Codice[128];

set\_binding\_param(&param[0], MYSQL\_TYPE\_VAR\_STRING, cod, sizeof(cod));

if (mysql\_stmt\_bind\_param(client\_bought, param) != 0) {

print\_stmt\_error(client\_bought, "Unable to bind input parameters");

goto out;

}

if (mysql\_stmt\_execute(client\_bought) != 0) {

print\_stmt\_error(client\_bought, "An error occurred while showing client's boughts.");

goto out;

}

if (mysql\_stmt\_store\_result(client\_bought)) {

print\_stmt\_error(client\_bought, "Unable to store client boughts result set.");

goto out;

}

set\_binding\_param(&param[0], MYSQL\_TYPE\_VAR\_STRING, Codice, sizeof(Codice));

bt = malloc(sizeof(struct bought\*) \* mysql\_stmt\_num\_rows(client\_bought));

if(bt == NULL)

goto out;

for(i = 0; i < mysql\_stmt\_num\_rows(client\_bought); i++) {

bt[i] = malloc(sizeof(struct bought));

memset(bt[i], 0, sizeof(struct bought));

}

bt[i] = NULL;

if(mysql\_stmt\_bind\_result(client\_bought, param)) {

print\_stmt\_error(client\_bought, "Unable to bind output parameters");

bt = NULL;

goto out;

}

temp = 0;

while (1) {

status = mysql\_stmt\_fetch(client\_bought);

if (status == 1 || status == MYSQL\_NO\_DATA)

break;

strcpy(bt[temp]->cod, Codice);

temp++;

}

out:

while(mysql\_stmt\_next\_result(client\_bought) != -1) {}

mysql\_stmt\_free\_result(client\_bought);

mysql\_stmt\_reset(client\_bought);

return bt;

}

struct fullclient \*get\_full\_client(struct client \*client)

{

struct fullclient \*fc;

struct client \*cl;

struct note \*\*note;

struct bought \*\*bought;

cl = get\_single\_client(client->code);

note = get\_client\_note(client->code);

bought = get\_client\_bought(client->code);

fc = malloc(sizeof(struct client\*) + sizeof(struct note\*) + sizeof(struct bought\*));

if(fc == NULL)

return NULL;

memset(fc, 0, sizeof(struct client\*) + sizeof(struct note\*) + sizeof(struct bought\*));

fc->client = cl;

fc->note = note;

fc->bought = bought;

return fc;

}

void dispose\_full(struct fullclient \*fc)

{

free(fc);

}

void do\_register\_user(struct user \*user)

{

MYSQL\_BIND param[4];

int id = 0, status;

set\_binding\_param(&param[0], MYSQL\_TYPE\_VAR\_STRING, user->credentials->username, sizeof(user->credentials->username));

set\_binding\_param(&param[1], MYSQL\_TYPE\_VAR\_STRING, user->credentials->password, sizeof(user->credentials->password));

set\_binding\_param(&param[2], MYSQL\_TYPE\_VAR\_STRING, user->role, sizeof(user->role));

set\_binding\_param(&param[3], MYSQL\_TYPE\_LONG, &id, sizeof(id));

if (mysql\_stmt\_bind\_param(users, param) != 0) {

print\_stmt\_error(users, "Unable to bind input parameters");

goto out;

}

if (mysql\_stmt\_execute(users) != 0) {

print\_stmt\_error(users, "An error occurred while registering a user");

goto out;

}

set\_binding\_param(&param[0], MYSQL\_TYPE\_LONG, &id, sizeof(id));

if(mysql\_stmt\_bind\_result(users, param)) {

print\_stmt\_error(users, "Unable to bind output parameters");

goto out;

}

while (1) {

status = mysql\_stmt\_fetch(users);

if (status == 1 || status == MYSQL\_NO\_DATA)

break;

snprintf(user->id, sizeof(id), "%d", id);

}

out:

while(mysql\_stmt\_next\_result(users) != -1) {}

mysql\_stmt\_free\_result(users);

mysql\_stmt\_reset(users);

return;

}

void expire\_consultation(struct cons \*cons)

{

MYSQL\_BIND param[1];

set\_binding\_param(&param[0], MYSQL\_TYPE\_VAR\_STRING, cons->cod, sizeof(cons->cod));

if (mysql\_stmt\_bind\_param(expire, param) != 0) {

print\_stmt\_error(expire, "Unable to bind input parameters");

return;

}

if (mysql\_stmt\_execute(expire) != 0) {

print\_stmt\_error(expire, "An error occurred while making a consultation expire");

return;

}

mysql\_stmt\_free\_result(expire);

mysql\_stmt\_reset(expire);

return;

}

struct consultations \*get\_active\_consultations(struct user \*user)

{

MYSQL\_BIND param[1];

int row = 0, status, temp;

char code[ID\_LEN];

struct consultations \*consultations = NULL;

temp = atoi(user->id);

set\_binding\_param(&param[0], MYSQL\_TYPE\_LONG, &temp, sizeof(temp));

if (mysql\_stmt\_bind\_param(get\_consultation, param) != 0) {

print\_stmt\_error(get\_consultation, "Unable to bind input parameters");

goto out;

}

if (mysql\_stmt\_execute(get\_consultation) != 0) {

print\_stmt\_error(get\_consultation, "An error occurred while getting consultation");

goto out;;

}

if (mysql\_stmt\_store\_result(get\_consultation)) {

print\_stmt\_error(get\_consultation, "Unable to store consultations information result set.");

goto out;

}

consultations = malloc(sizeof(\*consultations) + sizeof(struct cons) \* mysql\_stmt\_num\_rows(get\_consultation));

if(consultations == NULL)

goto out;

memset(consultations, 0, sizeof(\*consultations) + sizeof(struct cons) \* mysql\_stmt\_num\_rows(get\_consultation));

consultations->num\_rows = mysql\_stmt\_num\_rows(get\_consultation);

set\_binding\_param(&param[0], MYSQL\_TYPE\_VAR\_STRING, code, sizeof(code));

if(mysql\_stmt\_bind\_result(get\_consultation, param)) {

print\_stmt\_error(get\_consultation, "Unable to bind output parameters");

consultations = NULL;

goto out;

}

while (1) {

status = mysql\_stmt\_fetch(get\_consultation);

if (status == 1 || status == MYSQL\_NO\_DATA)

break;

strcpy(consultations->consultation[row].cod, code);

row++;

}

out:

while(mysql\_stmt\_next\_result(get\_consultation) != -1) {}

mysql\_stmt\_free\_result(get\_consultation);

mysql\_stmt\_reset(get\_consultation);

return consultations;

}

void dispose\_cons(struct consultations \*cons)

{

free(cons);

}

struct contact \*do\_register\_contact(struct contact \*contact)

{

MYSQL\_BIND param[3];

int id = 0, status, temp;

temp = atoi(contact->user);

set\_binding\_param(&param[0], MYSQL\_TYPE\_LONG, &id, sizeof(id));

set\_binding\_param(&param[1], MYSQL\_TYPE\_VAR\_STRING, contact->client, sizeof(contact->client));

set\_binding\_param(&param[2], MYSQL\_TYPE\_LONG, &temp, sizeof(temp));

if (mysql\_stmt\_bind\_param(reg\_contact, param) != 0) {

print\_stmt\_error(reg\_contact, "Unable to bind input parameters");

goto out;

}

if (mysql\_stmt\_execute(reg\_contact) != 0) {

print\_stmt\_error(reg\_contact, "An error occurred while registering a contact");

goto out;

}

set\_binding\_param(&param[0], MYSQL\_TYPE\_LONG, &id, sizeof(id));

if(mysql\_stmt\_bind\_result(reg\_contact, param)) {

print\_stmt\_error(reg\_contact, "Unable to bind output parameters");

goto out;

}

while (1) {

status = mysql\_stmt\_fetch(reg\_contact);

if (status == 1 || status == MYSQL\_NO\_DATA)

break;

snprintf(contact->ID, sizeof(id), "%d", id);

}

out:

while(mysql\_stmt\_next\_result(reg\_contact) != -1) {}

mysql\_stmt\_free\_result(reg\_contact);

mysql\_stmt\_reset(reg\_contact);

return contact;

}

db.h

#pragma once

#include <stdbool.h>

#include <stdlib.h>

extern bool init\_db(void);

extern void fini\_db(void);

#define DATE\_LEN 11

#define TIME\_LEN 6

#define DATETIME\_LEN (DATE\_LEN + TIME\_LEN)

#define USERNAME\_LEN 45

#define PASSWORD\_LEN 45

struct credentials {

char username[USERNAME\_LEN];

char password[PASSWORD\_LEN];

};

#define NAME\_LEN 45

#define SURNAME\_LEN 45

#define LOCATION\_LEN 128

#define ID\_LEN 128

#define NUMBER\_LEN 128

#define TEXT\_LEN 4096

typedef enum {

LOGIN\_ROLE,

UTENTE,

MANAGER,

COMMERCIALE,

FAILED\_LOGIN

} role\_t;

struct user{

char id[ID\_LEN];

char role[NAME\_LEN];

struct credentials \*credentials;

};

struct client{

char code[ID\_LEN];

char name[NAME\_LEN];

char surname[SURNAME\_LEN];

char birthday[DATE\_LEN];

char registration[DATE\_LEN];

char fax[NUMBER\_LEN];

char email[NUMBER\_LEN];

char address[NUMBER\_LEN];

char phone[NUMBER\_LEN];

char comm[ID\_LEN];

char funz[ID\_LEN];

char IVA[NAME\_LEN];

};

struct appointment{

char day[DATE\_LEN];

char time[TIME\_LEN];

char place[LOCATION\_LEN];

char func[ID\_LEN];

char client[ID\_LEN];

int id;

};

struct appointments{

unsigned num\_rows;

struct appointment appointment[];

};

struct cons{

char cod[ID\_LEN];

char day[DATE\_LEN];

};

struct consultations{

unsigned num\_rows;

struct cons consultation[];

};

struct note{

char number[ID\_LEN];

char text[TEXT\_LEN];

char func[ID\_LEN];

char input[ID\_LEN];

};

struct bought{

char client[ID\_LEN];

char cod[ID\_LEN];

};

struct fullclient{

struct client \*client;

struct note \*\*note;

struct bought \*\*bought;

};

struct contact{

char ID[ID\_LEN];

char client[ID\_LEN];

char user[ID\_LEN];

};

static void close\_prepared\_stmts(void);

static bool initialize\_prepared\_stmts(role\_t for\_role);

role\_t attempt\_login(struct credentials \*cred);

void db\_switch\_to\_login(void);

void db\_switch\_to\_utente(void);

void do\_register\_client(struct client \*client);

int do\_register\_appointment(struct appointment \*app);

void db\_switch\_to\_manager(void);

void db\_switch\_to\_comm(void);

void do\_register\_user(struct user \*user);

struct appointments \*get\_agenda(struct user \*user);

void dispose\_agenda(struct appointments \*app);

struct client \*\*get\_clients(struct user \*user);

void dispose\_clients(struct client \*\*clients);

void do\_register\_consultation(struct user \*user, struct cons \*cons);

void do\_register\_opportunity(struct client \*client, struct cons \*cons);

void do\_register\_note(struct note \*note, struct cons \*cons);

void do\_modify\_note(struct note \*note);

void do\_delete\_note(struct note \*note);

struct fullclient \*get\_full\_client(struct client \*client);

void dispose\_full(struct fullclient \*fc);

void expire\_consultation(struct cons \*cons);

struct consultations \*get\_active\_consultations(struct user \*user);

void dispose\_cons(struct consultations \*cons);

struct contact \*do\_register\_contact(struct contact \*contact);

db\_prof.c

#include <stdlib.h>

#include <stdio.h>

#include <stdbool.h>

#include <string.h>

#include <mysql/mysql.h>

#include <assert.h>

#include "db.h"

#include "db\_prof.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);

}

void set\_binding\_param(MYSQL\_BIND \*param, enum enum\_field\_types type, void \*buffer, unsigned long len)

{

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

param->buffer\_type = type;

param->buffer = buffer;

param->buffer\_length = len;

}

void date\_to\_mysql\_time(char \*str, MYSQL\_TIME \*time)

{

memset(time, 0, sizeof(\*time));

sscanf(str, "%4d-%2d-%2d", &time->year, &time->month, &time->day);

time->time\_type = MYSQL\_TIMESTAMP\_DATE;

}

void time\_to\_mysql\_time(char \*str, MYSQL\_TIME \*time)

{

memset(time, 0, sizeof(\*time));

sscanf(str, "%02d:%02d", &time->hour, &time->minute);

time->time\_type = MYSQL\_TIMESTAMP\_TIME;

}

void init\_mysql\_timestamp(MYSQL\_TIME \*time)

{

memset(time, 0, sizeof (\*time));

time->time\_type = MYSQL\_TIMESTAMP\_DATETIME;

}

void mysql\_timestamp\_to\_string(MYSQL\_TIME \*time, char \*str)

{

snprintf(str, DATETIME\_LEN, "%4d-%02d-%02d %02d:%02d", time->year, time->month, time->day, time->hour, time->minute);

}

void mysql\_date\_to\_string(MYSQL\_TIME \*date, char \*str)

{

snprintf(str, DATE\_LEN, "%4d-%02d-%02d", date->year, date->month, date->day);

}

void mysql\_time\_to\_string(MYSQL\_TIME \*time, char \*str)

{

snprintf(str, DATE\_LEN, "%2d:%02d", time->hour, time->minute);

}

db\_prof.c

#pragma once

#include <stdbool.h>

#include <mysql/mysql.h>

extern void print\_stmt\_error (MYSQL\_STMT \*stmt, char \*message);

extern void print\_error(MYSQL \*conn, char \*message);

extern bool setup\_prepared\_stmt(MYSQL\_STMT \*\*stmt, char \*statement, MYSQL \*conn);

extern void finish\_with\_error(MYSQL \*conn, char \*message);

extern void finish\_with\_stmt\_error(MYSQL \*conn, MYSQL\_STMT \*stmt, char \*message, bool close\_stmt);

extern void set\_binding\_param(MYSQL\_BIND \*param, enum enum\_field\_types type, void \*buffer, unsigned long len);

extern void date\_to\_mysql\_time(char \*str, MYSQL\_TIME \*time);

extern void time\_to\_mysql\_time(char \*str, MYSQL\_TIME \*time);

extern void init\_mysql\_timestamp(MYSQL\_TIME \*time);

extern void mysql\_timestamp\_to\_string(MYSQL\_TIME \*time, char \*str);

extern void mysql\_date\_to\_string(MYSQL\_TIME \*date, char \*str);

extern void mysql\_time\_to\_string(MYSQL\_TIME \*time, char \*str);

dotenv.c

#include <stdio.h>

#include <memory.h>

#include <stdlib.h>

#include <stdbool.h>

/\* strtok\_r() won't remove the whole ${ part, only the $ \*/

#define remove\_bracket(name) name + 1

#define remove\_space(value) value + 1

static char \*concat(char \*buffer, char \*string)

{

if (!buffer) {

return strdup(string);

}

if (string) {

size\_t length = strlen(buffer) + strlen(string) + 1;

char \*new = realloc(buffer, length);

return strcat(new, string);

}

return buffer;

}

static bool is\_nested(char \*value)

{

return strstr(value, "${") && strstr(value, "}");

}

/\*\*

\* @example With TEST\_DIR=${BASE\_DIR}/.test the first strtok\_r call will return

\* BASE\_DIR}/.test instead of NULL, or an empty string

\*/

static char \*prepare\_value(char \*value)

{

char \*new = malloc(strlen(value) + 2);

sprintf(new, " %s", value);

return new;

}

static char \*parse\_value(char \*value)

{

value = prepare\_value(value);

char \*search = value, \*parsed = NULL, \*tok\_ptr;

char \*name;

if (value && is\_nested(value)) {

while (1) {

parsed = concat(parsed, strtok\_r(search, "${", &tok\_ptr));

name = strtok\_r(NULL, "}", &tok\_ptr);

if (!name) {

break;

}

parsed = concat(parsed, getenv(remove\_bracket(name)));

search = NULL;

}

free(value);

return parsed;

}

return value;

}

static bool is\_commented(char \*line)

{

if ('#' == line[0]) {

return true;

}

int i = 0;

while (' ' == line[i]) {

if ('#' == line[++i]) {

return true;

}

}

return false;

}

static void set\_variable(char \*name, char \*original, bool overwrite)

{

char \*parsed;

if (original) {

parsed = parse\_value(original);

setenv(name, remove\_space(parsed), overwrite);

free(parsed);

}

}

static void parse(FILE \*file, bool overwrite)

{

char \*name, \*original, \*line = NULL, \*tok\_ptr;

size\_t len = 0;

while (-1 != getline(&line, &len, file)) {

if (!is\_commented(line)) {

name = strtok\_r(line, "=", &tok\_ptr);

original = strtok\_r(NULL, "\n", &tok\_ptr);

set\_variable(name, original, overwrite);

}

}

free(line);

}

static FILE \*open\_default(const char \*base\_path)

{

char path[strlen(base\_path) + strlen(".env") + 1];

sprintf(path, "%s/.env", base\_path);

return fopen(path, "rb");

}

int env\_load(const char \*path, bool overwrite)

{

FILE \*file = open\_default(path);

if (!file) {

file = fopen(path, "rb");

if (!file) {

return -1;

}

}

parse(file, overwrite);

fclose(file);

return 0;

}

dotenv.h

#ifndef DOTENV\_DOTENV\_H

#define DOTENV\_DOTENV\_H

#include <stdbool.h>

#ifdef \_\_cplusplus

extern "C"

{

#endif

/\*\*

\* @param path Can be a directory containing a file named .env, or the path of the env file itself

\* @param overwrite Existing variables will be overwritten

\* @return 0 on success, -1 if can't open the file

\*/

int env\_load(const char\* path, bool overwrite);

#ifdef \_\_cplusplus

}

#endif

#endif

io.c

#include <stdlib.h>

#include <stdio.h>

#include <string.h>

#include <ctype.h>

#include <stdbool.h>

#include <setjmp.h>

#ifdef \_\_unix\_\_

#include <termios.h>

#endif

#include "io.h"

jmp\_buf leave\_buff;

bool io\_initialized;

static void leave(void)

{

if(io\_initialized)

longjmp(leave\_buff, 1);

else

exit(EXIT\_SUCCESS);

}

char \*get\_input(char \*question, int len, char \*buff, bool hide)

{

printf("%s", question);

#ifdef \_\_unix\_\_

struct termios term, oterm;

if(hide) {

fflush(stdout);

if (tcgetattr(fileno(stdin), &oterm) == 0) {

memcpy(&term, &oterm, sizeof(struct termios));

term.c\_lflag &= ~(ECHO|ECHONL);

tcsetattr(fileno(stdin), TCSAFLUSH, &term);

} else {

memset(&term, 0, sizeof(struct termios));

memset(&oterm, 0, sizeof(struct termios));

}

}

#else

// Look at termio.h on MSDN to implement similar functionality on Windows

(void)hide;

#endif

if(fgets(buff, len, stdin) != NULL) {

buff[strcspn(buff, "\n")] = 0;

} else {

printf("EOF received, leaving...\n");

fflush(stdout);

leave();

}

// Empty stdin

if(strlen(buff) + 1 == len) {

int ch;

while(((ch = getchar()) != EOF) && (ch != '\n'));

if(ch == EOF) {

printf("EOF received, leaving...\n");

fflush(stdout);

leave();

}

}

#ifdef \_\_unix\_\_

if(hide) {

fwrite("\n", 1, 1, stdout);

tcsetattr(fileno(stdin), TCSAFLUSH, &oterm);

}

#endif

return buff;

}

bool yes\_or\_no(char \*question, char yes, char no, bool default\_answer, bool insensitive)

{

int extra;

// yes and no characters should be lowercase by default

yes = (char)tolower(yes);

no = (char)tolower(no);

// Which of the two is the default?

char s, n;

if(default\_answer) {

s = (char)toupper(yes);

n = no;

} else {

s = yes;

n = (char)toupper(no);

}

while(true) {

printf("%s [%c/%c]: ", question, s, n);

extra = 0;

char c = (char)getchar();

char ch = 0;

if(c != '\n') {

while(((ch = (char)getchar()) != EOF) && (ch != '\n'))

extra++;

}

if(c == EOF || ch == EOF) {

printf("EOF received, leaving...\n");

fflush(stdout);

leave();

}

if(extra > 0)

continue;

// Check the answer

if(c == '\n') {

return default\_answer;

} else if(c == yes) {

return true;

} else if(c == no) {

return false;

} else if(c == toupper(yes)) {

if(default\_answer || insensitive) return true;

} else if(c == toupper(no)) {

if(!default\_answer || insensitive) return false;

}

}

}

char multi\_choice(char \*question, const char choices[], int num)

{

char possibilities[2 \* num \* sizeof(char)];

int i, j = 0, extra;

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

possibilities[j++] = choices[i];

possibilities[j++] = '/';

}

possibilities[j - 1] = '\0'; // Remove last '/'

while(true) {

printf("%s [%s]: ", question, possibilities);

extra = 0;

char c = (char)getchar();

if(c == '\n')

continue;

char ch;

while(((ch = (char)getchar()) != EOF) && (ch != '\n'))

extra++;

if(c == EOF || ch == EOF) {

printf("EOF received, leaving...\n");

fflush(stdout);

leave();

}

if(extra > 1) // Need exactly one character on stdin

continue;

// Check if the choice is valid

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

if(c == choices[i])

return c;

}

}

}

void clear\_screen(void)

{

// To whom it may interest: this "magic" is a sequence of escape codes from VT100 terminals:

// https://www.csie.ntu.edu.tw/~r92094/c++/VT100.html

printf("\033[2J\033[H");

}

void press\_anykey(void)

{

char c;

puts("\nPress any key to continue...");

while((c = (char)getchar()) != '\n');

(void)c;

}

io.h

#pragma once

#include <stdbool.h>

#include <setjmp.h>

extern jmp\_buf leave\_buff;

extern bool io\_initialized;

#define initialize\_io() \

\_\_extension\_\_ ({ \

io\_initialized = true; \

int \_\_ret = setjmp(leave\_buff); \

\_\_ret == 0; \

})

extern char \*get\_input(char \*question, int len, char \*buff, bool hide);

extern bool yes\_or\_no(char \*question, char yes, char no, bool default\_answer, bool insensitive);

extern char multi\_choice(char \*question, const char choices[], int num);

extern void clear\_screen(void);

extern void press\_anykey(void);

login.c

#include <stdbool.h>

#include <stdio.h>

#include "login.h"

#include "io.h"

#include "db.h"

#include "utente\_app.h"

#include "manager\_app.h"

#include "commerciale\_app.h"

bool login(void)

{

struct credentials cred;

view\_login(&cred);

role\_t role = attempt\_login(&cred);

switch(role) {

case UTENTE:

controller\_utente();

break;

case MANAGER:

controller\_manager();

break;

case COMMERCIALE:

controller\_commerciale();

break;

default:

return false;

}

return true;

}

void view\_login(struct credentials \*cred)

{

clear\_screen();

puts("\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

puts("\* CRM SYSTEM \*");

puts("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

get\_input("Username: ", USERNAME\_LEN, cred->username, false);

get\_input("Password: ", PASSWORD\_LEN, cred->password, true);

}

bool ask\_for\_relogin(void)

{

return yes\_or\_no("Do you want to log in as a different user?", 'y', 'n', false, true);

}

login.h

#pragma once

#include "db.h"

extern bool ask\_for\_relogin(void);

extern void view\_login(struct credentials \*cred);

extern bool login(void);

main.c

#include <stdio.h>

#include "login.h"

#include "db.h"

#include "dotenv.h"

#include "io.h"

#include "validation.h"

#define check\_env\_failing(varname) \

if(getenv((varname)) == NULL) { \

fprintf(stderr, "[FATAL] env variable %s not set\n", (varname)); \

ret = false; \

}

static bool validate\_dotenv(void)

{

bool ret = true;

check\_env\_failing("HOST");

check\_env\_failing("DB");

check\_env\_failing("LOGIN\_USER");

check\_env\_failing("LOGIN\_PASS");

check\_env\_failing("ADMINISTRATOR\_USER");

check\_env\_failing("ADMINISTRATOR\_PASS");

check\_env\_failing("AGENCY\_USER");

check\_env\_failing("AGENCY\_PASS");

return ret;

}

#undef set\_env\_failing

int main()

{

if(env\_load(".", false) != 0)

return 1;

// if(!validate\_dotenv())

// return 1;

if(!init\_validation())

return 1;

if(!init\_db())

return 1;

if(initialize\_io()) {

do {

if(!login())

fprintf(stderr, "Login unsuccessful\n");

db\_switch\_to\_login();

} while(ask\_for\_relogin());

}

fini\_db();

fini\_validation();

puts("Bye!");

return 0;

}

manager.c

#include <stdio.h>

#include <string.h>

#include "db.h"

#include "manager.h"

#include "io.h"

#include "validation.h"

int get\_manager\_action(void)

{

char options[5] = {'1','2','3','4','5'};

char op;

clear\_screen();

puts("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

puts("\* USER'S DASHBOARD \*");

puts("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

puts("\*\*\* What should I do for you? \*\*\*\n");

puts("1) Add a user");

puts("2) Add a consultation");

puts("3) Make a consultation expire");

puts("4) Show active consultations");

puts("5) Quit");

op = multi\_choice("Select an option", options, 5);

return op - '1';

}

void get\_user\_inf(struct user \*user)

{

clear\_screen();

puts("\*\* Get information about user \*\*\n");

get\_input("Insert username: ", USERNAME\_LEN, user->credentials->username, false);

get\_input("Insert password: ", PASSWORD\_LEN, user->credentials->password, false);

get\_input("Insert role: ", NAME\_LEN, user->role, false);

}

void print\_user\_id(struct user \*user)

{

clear\_screen();

puts("\*\* User's ID \*\*\n");

if(strcmp(user->id, "") == 0) {

printf("\nInvalid role. Please enter only [utente], [manager] or [commerciale]\n");

return;

}

printf("\n%s\n", user->id);

}

void get\_cons\_inf(struct user \*user, struct cons \*cons)

{

clear\_screen();

puts("\*\* Get information about consultation \*\*\n");

get\_input("Insert your username: ", USERNAME\_LEN, user->credentials->username, false);

get\_input("Insert Code: ", ID\_LEN, cons->cod, false);

get\_input("Insert activation date: ", DATE\_LEN, cons->day, false);

}

void get\_cons\_code(struct cons \*cons)

{

clear\_screen();

puts("\*\* Get information about expiring consultation \*\*\n");

get\_input("Insert Code: ", ID\_LEN, cons->cod, false);

}

void get\_manager\_id(struct user \*user)

{

clear\_screen();

puts("\*\* Get information about manager \*\*\n");

get\_input("Insert your ID: ", ID\_LEN, user->id, false);

}

void print\_consultations(struct consultations \*consultations)

{

clear\_screen();

puts("\*\* Showing your consultations still active \*\*\n");

printf("Number of consultations still active: %d\n", consultations->num\_rows);

for(int i = 0; i < consultations->num\_rows; i++) {

printf("\n%s\n", consultations->consultation[i].cod);

}

}

manager.h

#pragma once

#include "db.h"

enum actions {

USER,

CONSULTATION,

EXPIRE,

SHOW,

QUIT,

END\_OF\_ACTIONS

};

int get\_manager\_action(void);

void get\_user\_inf(struct user \*user);

void print\_user\_id(struct user \*user);

void get\_cons\_inf(struct user \*user, struct cons \*cons);

void get\_cons\_code(struct cons \*cons);

void get\_manager\_id(struct user \*user);

void print\_consultations(struct consultations \*consultations);

manager\_app.c

#include <stdio.h>

#include <string.h>

#include "db.h"

#include "manager.h"

#include "manager\_app.h"

#include "io.h"

static bool register\_user(void)

{

struct user user;

struct credentials cred;

memset(&user, 0, sizeof(user));

memset(&cred, 0, sizeof(cred));

user.credentials = &cred;

get\_user\_inf(&user);

do\_register\_user(&user);

print\_user\_id(&user);

return false;

}

static bool register\_consultation(void)

{

struct cons cons;

struct user user;

struct credentials cred;

memset(&cons, 0, sizeof(cons));

memset(&user, 0, sizeof(user));

memset(&cred, 0, sizeof(cred));

user.credentials = &cred;

get\_cons\_inf(&user, &cons);

do\_register\_consultation(&user, &cons);

return false;

}

static bool expire(void)

{

struct cons cons;

memset(&cons, 0, sizeof(cons));

get\_cons\_code(&cons);

expire\_consultation(&cons);

return false;

}

static bool show\_consultation(void)

{

struct user user;

struct consultations \*consultations;

memset(&user, 0, sizeof(user));

get\_manager\_id(&user);

consultations = get\_active\_consultations(&user);

print\_consultations(consultations);

dispose\_cons(consultations);

return false;

}

static bool quit(void)

{

return true;

}

static struct {

enum actions action;

bool (\*control)(void);

} controls[END\_OF\_ACTIONS] = {

{.action = USER, .control = register\_user},

{.action = CONSULTATION, .control = register\_consultation},

{.action = EXPIRE, .control = expire},

{.action = SHOW, .control = show\_consultation},

{.action = QUIT, .control = quit}

};

void controller\_manager(void)

{

db\_switch\_to\_manager();

while(true) {

int action = get\_manager\_action();

if(action >= END\_OF\_ACTIONS) {

fprintf(stderr, "Error: unknown action\n");

continue;

}

if (controls[action].control())

break;

press\_anykey();

}

}

manager\_app.h

#pragma once

void controller\_manager(void);

utente.c

#include <stdio.h>

#include <stdlib.h>

#include "db.h"

#include "utente.h"

#include "io.h"

#include "validation.h"

int get\_user\_action(void)

{

char options[10] = {'1','2','3','4','5','6','7','8','9','0'};

char op;

clear\_screen();

puts("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

puts("\* USER'S DASHBOARD \*");

puts("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

puts("\*\*\* What should I do for you? \*\*\*\n");

puts("1) Show agenda");

puts("2) Show client");

puts("3) Show my clients");

puts("4) Insert note");

puts("5) Modify note");

puts("6) Delete note");

puts("7) Offer an opportunity");

puts("8) Register an appointment");

puts("9) Register a contact");

puts("0) Quit");

op = multi\_choice("Select an option", options, 10);

if(op == '0')

return 9;

return op - '1';

}

void get\_user\_id(struct user \*user)

{

clear\_screen();

puts("\*\* Get your personal information \*\*\n");

get\_input("Insert your ID: ", ID\_LEN, user->id, false);

}

void print\_agenda(struct appointments \*app)

{

clear\_screen();

puts("\*\* Agenda \*\*\n");

for(int i = 0; i < app->num\_rows; i++) {

printf("\n%s %s %s\n", app->appointment[i].day, app->appointment[i].time, app->appointment[i].place);

}

}

void get\_client\_id(struct client \*client)

{

clear\_screen();

puts("\*\* Get information about a specific client \*\*\n");

get\_input("Insert client's Fiscal Code: ", ID\_LEN, client->code, false);

}

void print\_client(struct fullclient \*fc)

{

clear\_screen();

puts("\*\* Client's informations \*\*\n");

printf("\nFiscal Code: %s", fc->client->code);

printf("\nName: %s", fc->client->name);

printf("\nSurname: %s", fc->client->surname);

printf("\nBirthday: %s", fc->client->birthday);

printf("\nDay of registration: %s", fc->client->registration);

printf("\nFax: %s", fc->client->fax);

printf("\nEmail: %s", fc->client->email);

printf("\nAddress: %s", fc->client->address);

printf("\nPhone number: %s", fc->client->phone);

printf("\nID of the user who added it: %s", fc->client->comm);

printf("\nID of the user who is related: %s", fc->client->funz);

if(atoi(fc->client->IVA) != 0) printf("\nIVA: %s\n", fc->client->IVA);

else printf("\nThis client has no IVA\n");

puts("\n\*\* Client's appoinments and contacts \*\*\n");

int row = 0;

if(fc->note != NULL){

while(fc->note[row] != NULL) row++;

for(size\_t i = 0; i < row; i++) {

printf("\n%s\n", fc->note[i]->number);

}

}

puts("\n\*\* Client's bought consultations \*\*\n");

row = 0;

if(fc->bought != NULL){

while(fc->bought[row] != NULL) row++;

for(size\_t i = 0; i < row; i++) {

printf("\n%s\n", fc->bought[i]->cod);

}

}

}

void show\_clients(struct client \*\*clients)

{

clear\_screen();

puts("\*\* Your Clients \*\*\n");

int row = 0;

while(clients[row] != NULL) row++;

for(int i = 0; i < row; i++) {

printf("\n%s\n", clients[i]->code);

}

}

void ins\_note(struct note \*note, struct cons \*cons)

{

clear\_screen();

puts("\*\* Get note's datas \*\*\n");

get\_input("Insert note's ID: ", ID\_LEN, note->number, false);

get\_input("Insert ID of the user who is writing it: ", ID\_LEN, note->func, false);

get\_input("Insert text: ", TEXT\_LEN, note->text, false);

get\_input("Insert appointment or contact's ID: ", ID\_LEN, note->input, false); //serve per distinguere se ci si riferisce ad un contatto o ad un appuntamento

get\_input("Insert consultation's ID if someone has received affermative answer: ", ID\_LEN, cons->cod, false);

}

void mod\_note(struct note \*note)

{

clear\_screen();

puts("\*\* Get note's datas \*\*\n");

get\_input("Insert Note's ID: ", ID\_LEN, note->number, false);

get\_input("Insert text: ", TEXT\_LEN, note->text, false);

}

void del\_note(struct note \*note)

{

clear\_screen();

puts("\*\* Get note's ID \*\*\n");

get\_input("Insert Note's ID: ", ID\_LEN, note->number, false);

}

void get\_opportunity(struct cons \*cons)

{

get\_input("Insert opportunity's ID: ", ID\_LEN, cons->cod, false);

}

void get\_appointment\_information(struct appointment \*app)

{

clear\_screen();

puts("\*\* Get appointment information \*\*\n");

while(true) {

if(validate\_date(get\_input("Insert day [YYYY-MM-DD]: ", DATE\_LEN, app->day, false)))

break;

fprintf(stderr, "Invalid date!\n");

}

while(true) {

if(validate\_time(get\_input("Insert time [XX:XX]: ", TIME\_LEN, app->time, false)))

break;

fprintf(stderr, "Invalid time!\n");

}

get\_input("Insert location: ", LOCATION\_LEN, app->place, false);

while(true) {

if(atoi(get\_input("Insert user's ID: ", ID\_LEN, app->func, false)) != 0)

break;

fprintf(stderr, "Invalid ID!\n");

}

get\_input("Insert client's ID: ", ID\_LEN, app->client, false);

}

void print\_appointment(int cod)

{

clear\_screen();

puts("\*\* Your Appointment ID \*\*\n");

printf("\n%d\n", cod);

}

void get\_contact\_inf(struct contact \*contact)

{

clear\_screen();

puts("\*\* Get contact information \*\*\n");

while(true) {

if(atoi(get\_input("Insert user's ID: ", ID\_LEN, contact->user, false)) != 0)

break;

fprintf(stderr, "Invalid ID!\n");

}

get\_input("Insert client's ID: ", ID\_LEN, contact->client, false);

}

void print\_contact\_id(struct contact \*contact)

{

clear\_screen();

puts("\*\* Your Contact ID \*\*\n");

printf("\n%d\n", atoi(contact->ID));

}

utente.h

#pragma once

#include "db.h"

enum actions {

AGENDA,

CLIENT,

MY\_CLIENTS,

INS\_NOTE,

MOD\_NOTE,

DEL\_NOTE,

OPPORTUNITY,

APPOINTMENT,

CONTACT,

QUIT,

END\_OF\_ACTIONS

};

extern int get\_user\_action(void);

extern void get\_user\_id(struct user \*user);

extern void print\_agenda(struct appointments \*app);

extern void get\_client\_id(struct client \*client);

extern void print\_client(struct fullclient \*fc);

extern void show\_clients(struct client \*\*clients);

extern void ins\_note(struct note \*note, struct cons \*cons);

extern void mod\_note(struct note \*note);

extern void del\_note(struct note \*note);

extern void get\_opportunity(struct cons \*cons);

extern void get\_appointment\_information(struct appointment \*app);

extern void print\_appointment(int cod);

extern void get\_contact\_inf(struct contact \*contact);

extern void print\_contact\_id(struct contact \*contact);

utente\_app.c

#include <stdio.h>

#include <string.h>

#include "db.h"

#include "utente.h"

#include "utente\_app.h"

#include "io.h"

static bool show\_agenda(void)

{

struct user user;

struct appointments \*app;

memset(&user, 0, sizeof(user));

get\_user\_id(&user);

app = get\_agenda(&user);

print\_agenda(app);

dispose\_agenda(app);

return false;

}

static bool show\_client(void)

{

struct client client;

struct fullclient \*fc;

memset(&client, 0, sizeof(client));

get\_client\_id(&client);

fc = get\_full\_client(&client);

print\_client(fc);

dispose\_full(fc);

return false;

}

static bool show\_my\_clients(void)

{

struct user user;

struct client \*\*clients;

memset(&user, 0, sizeof(user));

get\_user\_id(&user);

clients = get\_clients(&user);

show\_clients(clients);

dispose\_clients(clients);

return false;

}

static bool insert\_note(void)

{

struct note note;

struct cons cons;

memset(&note, 0, sizeof(note));

memset(&cons, 0, sizeof(cons));

ins\_note(&note, &cons);

do\_register\_note(&note, &cons);

return false;

}

static bool modify\_note(void)

{

struct note note;

memset(&note, 0, sizeof(note));

mod\_note(&note);

do\_modify\_note(&note);

return false;

}

static bool delete\_note(void)

{

struct note note;

memset(&note, 0, sizeof(note));

del\_note(&note);

do\_delete\_note(&note);

return false;

}

static bool offer\_opportunity(void)

{

struct client client;

struct cons cons;

memset(&client, 0, sizeof(client));

get\_client\_id(&client);

memset(&cons, 0, sizeof(cons));

get\_opportunity(&cons);

do\_register\_opportunity(&client, &cons);

return false;

}

static bool register\_appointment(void)

{

struct appointment app;

int cod;

memset(&app, 0, sizeof(app));

get\_appointment\_information(&app);

cod = do\_register\_appointment(&app);

if(cod < 0) {

printf("Something gone wrong. Try again later\n");

return true;

}

print\_appointment(cod);

return false;

}

static bool register\_contact(void)

{

struct contact cont;

memset(&cont, 0, sizeof(cont));

get\_contact\_inf(&cont);

do\_register\_contact(&cont);

print\_contact\_id(&cont);

return false;

}

static bool quit(void)

{

return true;

}

static struct {

enum actions action;

bool (\*control)(void);

} controls[END\_OF\_ACTIONS] = {

{.action = AGENDA, .control = show\_agenda},

{.action = CLIENT, .control = show\_client},

{.action = MY\_CLIENTS, .control = show\_my\_clients},

{.action = INS\_NOTE, .control = insert\_note},

{.action = MOD\_NOTE, .control = modify\_note},

{.action = DEL\_NOTE, .control = delete\_note},

{.action = OPPORTUNITY, .control = offer\_opportunity},

{.action = APPOINTMENT, .control = register\_appointment},

{.action = CONTACT, .control = register\_contact},

{.action = QUIT, .control = quit}

};

void controller\_utente(void)

{

db\_switch\_to\_utente();

while(true) {

int action = get\_user\_action();

if(action >= END\_OF\_ACTIONS) {

fprintf(stderr, "Error: unknown action\n");

continue;

}

if (controls[action].control())

break;

press\_anykey();

}

}

utente.h

#pragma once

void controller\_utente(void);

validation.c

#include <stdbool.h>

#include <regex.h>

#include <stdio.h>

#include <string.h>

#include <errno.h>

#include "validation.h"

regex\_t regex\_date;

regex\_t regex\_time;

static void print\_regerror(int errcode, size\_t length, regex\_t \*compiled)

{

char buffer[length];

(void) regerror(errcode, compiled, buffer, length);

fprintf(stderr, "Regex match failed: %s\n", buffer);

}

bool init\_validation(void)

{

int ret1;

int ret2;

ret1 = regcomp(&regex\_date, "^([12][0-9]{3}-(0[1-9]|1[0-2])-(0[1-9]|[12][0-9]|3[01]))", REG\_EXTENDED);

if(ret1) {

if(ret1 == REG\_ESPACE) {

fprintf(stderr, "%s\n", strerror(ENOMEM));

} else {

fprintf(stderr, "Fatal error while setting up date validation regex.\n");

}

}

ret2 = regcomp(&regex\_time, "^([0-1]?[0-9]|2?[0-3]|[0-9]):([0-5][0-9]|[0-9])$", REG\_EXTENDED);

if(ret2) {

if(ret2 == REG\_ESPACE) {

fprintf(stderr, "%s\n", strerror(ENOMEM));

} else {

fprintf(stderr, "Fatal error while setting up time validation regex.\n");

}

}

return ret1 == 0 && ret2 == 0;

}

void fini\_validation(void)

{

regfree(&regex\_time);

regfree(&regex\_date);

}

bool validate\_date(char \*str)

{

int ret = regexec(&regex\_date, str, 0, NULL, REG\_NOTEOL);

if(ret != 0 && ret != REG\_NOMATCH) {

size\_t length = regerror(ret, &regex\_date, NULL, 0);

print\_regerror(ret, length, &regex\_date);

return false;

}

return ret == 0;

}

bool validate\_time(char \*str)

{

int ret = regexec(&regex\_time, str, 0, NULL, 0);

if(ret != 0 && ret != REG\_NOMATCH) {

size\_t length = regerror(ret, &regex\_time, NULL, 0);

print\_regerror(ret, length, &regex\_time);

return false;

}

return ret == 0;

}

validation.h

#pragma once

#include <stdbool.h>

extern bool init\_validation(void);

extern void fini\_validation(void);

extern bool validate\_time(char \*str);

extern bool validate\_date(char \*str);

1. Indicare con E le entità, con R le relazioni [↑](#footnote-ref-1)
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. [↑](#footnote-ref-2)
3. PK = primary key, NN = not null, UQ = unique, UN = unsigned, AI = auto increment. È ovviamente possibile specificare più di un attributo per ciascuna colonna. [↑](#footnote-ref-3)
4. PK = primary key, NN = not null, UQ = unique, UN = unsigned, AI = auto increment. È ovviamente possibile specificare più di un attributo per ciascuna colonna. [↑](#footnote-ref-4)
5. PK = primary key, NN = not null, UQ = unique, UN = unsigned, AI = auto increment. È ovviamente possibile specificare più di un attributo per ciascuna colonna. [↑](#footnote-ref-5)
6. PK = primary key, NN = not null, UQ = unique, UN = unsigned, AI = auto increment. È ovviamente possibile specificare più di un attributo per ciascuna colonna. [↑](#footnote-ref-6)
7. PK = primary key, NN = not null, UQ = unique, UN = unsigned, AI = auto increment. È ovviamente possibile specificare più di un attributo per ciascuna colonna. [↑](#footnote-ref-7)
8. PK = primary key, NN = not null, UQ = unique, UN = unsigned, AI = auto increment. È ovviamente possibile specificare più di un attributo per ciascuna colonna. [↑](#footnote-ref-8)
9. PK = primary key, NN = not null, UQ = unique, UN = unsigned, AI = auto increment. È ovviamente possibile specificare più di un attributo per ciascuna colonna. [↑](#footnote-ref-9)
10. PK = primary key, NN = not null, UQ = unique, UN = unsigned, AI = auto increment. È ovviamente possibile specificare più di un attributo per ciascuna colonna. [↑](#footnote-ref-10)
11. PK = primary key, NN = not null, UQ = unique, UN = unsigned, AI = auto increment. È ovviamente possibile specificare più di un attributo per ciascuna colonna. [↑](#footnote-ref-11)
12. PK = primary key, NN = not null, UQ = unique, UN = unsigned, AI = auto increment. È ovviamente possibile specificare più di un attributo per ciascuna colonna. [↑](#footnote-ref-12)
13. IDX = index, UQ = unique, FT = full text, PR = primary. [↑](#footnote-ref-13)
14. IDX = index, UQ = unique, FT = full text, PR = primary. [↑](#footnote-ref-14)
15. IDX = index, UQ = unique, FT = full text, PR = primary. [↑](#footnote-ref-15)
16. IDX = index, UQ = unique, FT = full text, PR = primary. [↑](#footnote-ref-16)
17. IDX = index, UQ = unique, FT = full text, PR = primary. [↑](#footnote-ref-17)
18. IDX = index, UQ = unique, FT = full text, PR = primary. [↑](#footnote-ref-18)
19. IDX = index, UQ = unique, FT = full text, PR = primary. [↑](#footnote-ref-19)
20. IDX = index, UQ = unique, FT = full text, PR = primary. [↑](#footnote-ref-20)
21. IDX = index, UQ = unique, FT = full text, PR = primary. [↑](#footnote-ref-21)
22. IDX = index, UQ = unique, FT = full text, PR = primary. [↑](#footnote-ref-22)
23. IDX = index, UQ = unique, FT = full text, PR = primary. [↑](#footnote-ref-23)