Basi di Dati e Conoscenza

Progetto A.A. 2020/2021

Sistema di prenotazione di esami in una ASL

0228612

Filippo Maria Briscese

Indice

1. Descrizione del Minimondo **2**

2. Analisi dei Requisiti **4**

3. Progettazione concettuale **9**

4. Progettazione logica **15**

5. Progettazione fisica **25**

Appendice: Implementazione **58**

1. Descrizione del Minimondo

Si vuole realizzare il sistema informativo di gestione di un sistema di prenotazioni di esami medici all'interno di una Azienda Sanitaria Locale (ASL), tenendo conto delle seguenti informazioni.

Ciascun paziente è identificato da un codice di tessera sanitaria ed è caratterizzato da un nome, un cognome, la data ed il luogo di nascita, un indirizzo di residenza ed un insieme arbitrario di recapiti (email, telefono, cellulare). La gestione dei pazienti è in capo al personale del CUP, che può gestire nella sua interezza l'anagrafica e le prenotazioni degli esami. In fase di prenotazione, è possibile prenotare con un unico codice di prenotazione un numero arbitrario di esami.

Gli esami medici che possono essere eseguiti sono identificati da un codice numerico e sono caratterizzati dalla descrizione di esame medico (ad esempio Radiografia, ecc.). L'insieme degli esami disponibili presso la ASL sono gestiti dagli amministratori del sistema. Ciascun esame è associato ad un insieme di valori numerici, riportanti i risultati dei parametri legati allo specifico esame. Inoltre, per ciascun esame è possibile inserire da parte del personale medico una diagnosi testuale.

Gli ospedali della ASL sono identificati da un codice numerico e sono caratterizzati da un nome, un indirizzo e dal nome di un responsabile. Anche la gestione degli ospedali è in capo all'amministrazione.

I laboratori che eseguono gli esami sono identificati da un codice univoco all'interno di un ospedale della ASL e sono caratterizzati dal nome del laboratorio, dal piano di ubicazione e dal numero di stanza. Anche per i laboratori è prevista la designazione di un responsabile.

Per ogni prenotazione di un esame da parte di un paziente si vuole memorizzare la data e l'ora dell'esame, il laboratorio presso cui è eseguito, il costo del ticket e se tale esame è prescritto con urgenza. Si tenga presente che ogni paziente può effettuare più prenotazioni dello stesso esame in date diverse. Si noti inoltre che lo stesso esame non può essere ripetuto nello stesso giorno dallo stesso paziente.

Ogni ospedale è suddiviso in reparti identificati da un codice numerico univoco all'interno dell'ospedale di appartenenza e caratterizzati dal nome del reparto e da un numero di telefono. Il personale del reparto è identificato attraverso il codice fiscale; sono noti inoltre il

nome, il cognome e l'indirizzo di domicilio. Tra il personale, nel caso dei medici primari del reparto è noto l'elenco delle specializzazioni, mentre per il personale volontario è noto il nome dell'associazione di appartenenza, se disponibile. I responsabili degli ospedali e dei laboratori vanno individuati all'interno del personale medico, che può essere gestito unicamente dall'amministrazione.

Per motivi di storicizzazione, gli amministratori possono generare dei report che mostrano ciascun membro del personale quanti esami (e quali esami) ha svolto, su base mensile e/o annuale. Il personale del CUP, altresì, ha la possibilità di generare dei report che riportano i risultati di un insieme di esami associati ad una prenotazione, e/o mostrare lo storico di tutti gli esami svolti da un determinato paziente dalla sua registrazione nel sistema.

2. Analisi dei Requisiti

Identificazione dei termini ambigui e correzioni possibili

Linea	Termine	Nuovo termine	Motivo correzione
11, 12, 25	Esame	Tipo di esame	Per esplicitare il fatto che si parla della descrizione dell'esame medico (es. Radiografia, ecc.), e non dell'esame inteso come prestazione.
22	Prenotazione di un esame	Esame prenotato	Le caratteristiche da memorizzare che seguono il termine originale riferiscono a "esame"; ho scelto dunque di aggettivare "prenotazione" e usare "esame" come sostantivo per rendere più evidente che i riferimenti sono a "esame".

Specifica disambiguata

Si vuole realizzare il sistema informativo di gestione di un sistema di prenotazioni di esami medici all'interno di una Azienda Sanitaria Locale (ASL), tenendo conto delle seguenti informazioni.

Ciascun paziente è identificato da un codice di tessera sanitaria ed è caratterizzato da un nome, un cognome, la data ed il luogo di nascita, un indirizzo di residenza ed un insieme arbitrario di recapiti (email, telefono, cellulare). La gestione dei pazienti è in capo al personale del CUP, che può gestire nella sua interezza l'anagrafica e le prenotazioni degli esami. In fase di prenotazione, è possibile prenotare con un unico codice di prenotazione un numero arbitrario di esami.

Gli esami medici che possono essere eseguiti sono identificati da un codice numerico e sono caratterizzati dalla descrizione del tipo di esame medico (ad esempio Radiografia, ecc.). L'insieme dei tipi di esami disponibili presso la ASL sono gestiti dagli amministratori del sistema. Ciascun esame è associato ad un insieme di valori numerici, riportanti i risultati dei parametri legati allo specifico esame. Inoltre, per ciascun esame è possibile inserire da parte del personale medico una diagnosi testuale.

Gli ospedali della ASL sono identificati da un codice numerico e sono caratterizzati da un nome, un indirizzo e dal nome di un responsabile. Anche la gestione degli ospedali è in capo all'amministrazione.

I laboratori che eseguono gli esami sono identificati da un codice univoco all'interno di un ospedale della ASL e sono caratterizzati dal nome del laboratorio, dal piano di ubicazione e dal numero di stanza. Anche per i laboratori è prevista la designazione di un responsabile.

Per ogni esame prenotato da parte di un paziente si vuole memorizzare la data e l'ora dell'esame, il laboratorio presso cui è eseguito, il costo del ticket e se tale esame è prescritto con urgenza. Si tenga presente che ogni paziente può effettuare più prenotazioni dello stesso tipo di esame in date diverse. Si noti inoltre che lo stesso tipo di esame non può essere ripetuto nello stesso giorno dallo stesso paziente.

Ogni ospedale è suddiviso in reparti identificati da un codice numerico univoco all'interno dell'ospedale di appartenenza e caratterizzati dal nome del reparto e da un numero di telefono. Il personale del reparto è identificato attraverso il codice fiscale; sono noti inoltre il nome, il cognome e l'indirizzo di domicilio. Tra il personale, nel caso dei medici primari del reparto è noto l'elenco delle specializzazioni, mentre per il personale volontario è noto il nome dell'associazione di appartenenza, se disponibile. I responsabili degli ospedali e dei laboratori vanno individuati all'interno del personale medico, che può essere gestito unicamente dall'amministrazione.

Per motivi di storicizzazione, gli amministratori possono generare dei report che mostrano ciascun membro del personale quanti esami (e quali esami) ha svolto, su base mensile e/o annuale. Il personale del CUP, altresì, ha la possibilità di generare dei report che riportano i risultati di un insieme di esami associati ad una prenotazione, e/o mostrare lo storico di tutti gli esami svolti da un determinato paziente dalla sua registrazione nel sistema.

Glossario dei Termini

Termine	Descrizione	Sinonimi	Collegamenti
Esame	Prestazione medica erogata dalla ASL su un paziente	Esame medico	Paziente, prenotazione, laboratorio, tipo di esame
Prenotazione	Appuntamento preso da un paziente per fare un esame in una data futura		Esame, paziente
Paziente	Persona che richiede un esame alla ASL		Esame, prenotazione

Tipo di esame	Descrizione dell'esame		Esame
Ospedale	Edificio gestito dalla ASL dove vengono erogate le sue prestazioni, coordinato da un medico detto responsabile		Laboratorio, reparto, personale medico
Laboratorio	Locale interno ad un ospedale dove vengono eseguiti gli esami; ogni laboratorio ha un medico responsabile		Ospedale, esame, personale medico
Reparto	Suddivisione dei locali di un ospedale e del personale che si occupano di una determinata specializzazione medica		Ospedale, medico primario, personale
Personale	Dipendenti della ASL		Reparto
Medico primario	Medico che ha l'incarico di coordinare un reparto		Reparto, specializzazione
Personale volontario	Persone che svolgono un servizio all'interno della ASL a titolo gratuito		Reparto
Personale medico	Membro del personale dottore in medica		Reparto

Raggruppamento dei requisiti in insiemi omogenei

Frasi relative agli esami
Gli esami medici che possono essere eseguiti sono identificati da un codice numerico e sono caratterizzati dalla descrizione del tipo di esame medico (ad esempio Radiografia, ecc.). L'insieme dei tipi di esami disponibili presso la ASL sono gestiti dagli amministratori del sistema. Ciascun esame è associato ad un insieme di valori numerici, riportanti i risultati dei parametri legati allo specifico esame. Inoltre, per ciascun esame è possibile inserire da parte del personale medico una diagnosi testuale. Per ogni esame prenotato da parte di un paziente si vuole memorizzare la data e l'ora dell'esame, il

laboratorio presso cui è eseguito, il costo del ticket e se tale esame è prescritto con urgenza. Si noti inoltre che lo stesso tipo di esame non può essere ripetuto nello stesso giorno dallo stesso paziente.

Per motivi di storicizzazione, gli amministratori possono generare dei report che mostrano ciascun membro del personale quanti esami (e quali esami) ha svolto, su base mensile e/o annuale. Il personale del CUP, altresì, ha la possibilità di generare dei report che riportano i risultati di un insieme di esami associati ad una prenotazione, e/o mostrare lo storico di tutti gli esami svolti da un determinato paziente dalla sua registrazione nel sistema.

Frasi relative alle prenotazioni

Si vuole realizzare il sistema informativo di gestione di un sistema di prenotazioni di esami medici all'interno di una Azienda Sanitaria Locale (ASL), tenendo conto delle seguenti informazioni.

La gestione dei pazienti è in capo al personale del CUP, che può gestire nella sua interezza l'anagrafica e le prenotazioni degli esami. In fase di prenotazione, è possibile prenotare con un unico codice di prenotazione un numero arbitrario di esami.

Frasi relative ai pazienti

Ciascun paziente è identificato da un codice di tessera sanitaria ed è caratterizzato da un nome, un cognome, la data ed il luogo di nascita, un indirizzo di residenza ed un insieme arbitrario di recapiti (email, telefono, cellulare). La gestione dei pazienti è in capo al personale del CUP, che può gestire nella sua interezza l'anagrafica e le prenotazioni degli esami.

Si tenga presente che ogni paziente può effettuare più prenotazioni dello stesso tipo di esame in date diverse.

Frasi relative agli ospedali

Gli ospedali della ASL sono identificati da un codice numerico e sono caratterizzati da un nome, un indirizzo e dal nome di un responsabile. Anche la gestione degli ospedali è in capo all'amministrazione.

Ogni ospedale è suddiviso in reparti […].

I responsabili degli ospedali […] vanno individuati all'interno del personale medico, che può essere gestito unicamente dall'amministrazione.

Frasi relative ai laboratori

I laboratori che eseguono gli esami sono identificati da un codice univoco all'interno di un ospedale della ASL e sono caratterizzati dal nome del laboratorio, dal piano di ubicazione e dal numero di stanza. Anche per i laboratori è prevista la designazione di un responsabile.

I responsabili […] dei laboratori vanno individuati all'interno del personale medico, che può essere gestito unicamente dall'amministrazione.

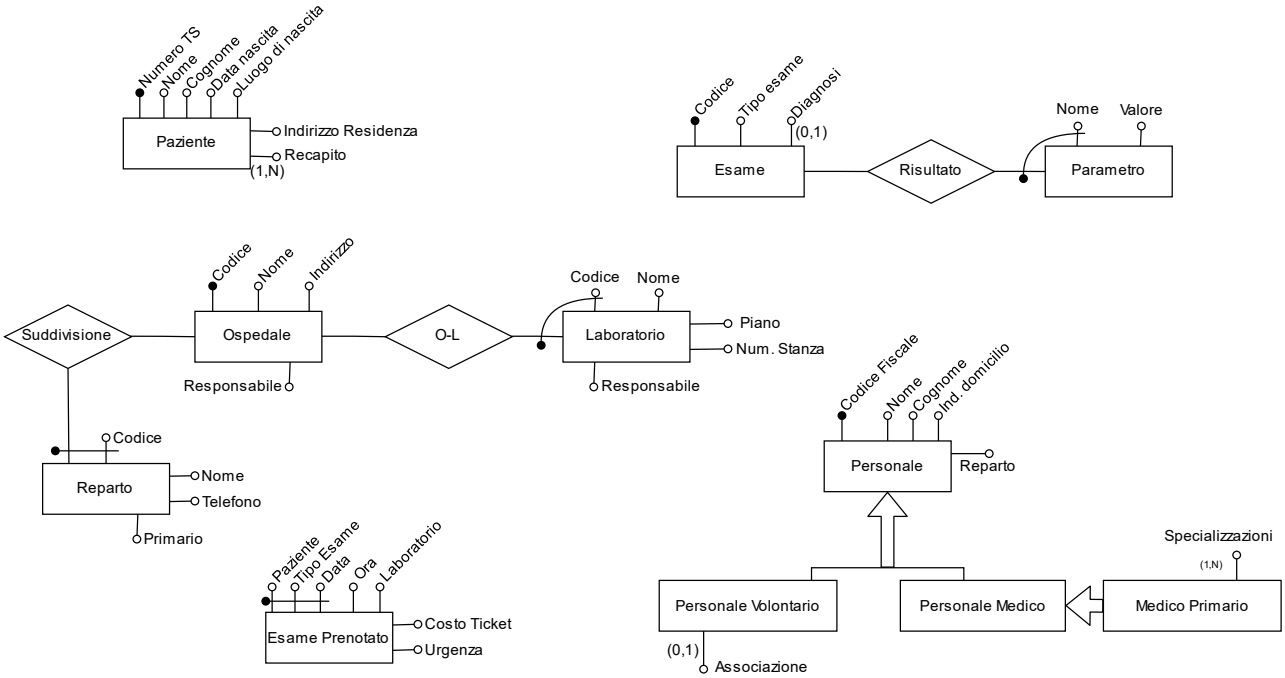
Frasi relative ai reparti
Ogni ospedale è suddiviso in reparti identificati da un codice numerico univoco all'interno dell'ospedale di appartenenza e caratterizzati dal nome del reparto e da un numero di telefono.

Frasi relative al personale
Il personale del reparto è identificato attraverso il codice fiscale; sono noti inoltre il nome, il cognome e l'indirizzo di domicilio. Tra il personale, nel caso dei medici primari del reparto è noto l'elenco delle specializzazioni, mentre per il personale volontario è noto il nome dell'associazione di appartenenza, se disponibile.

3. Progettazione concettuale

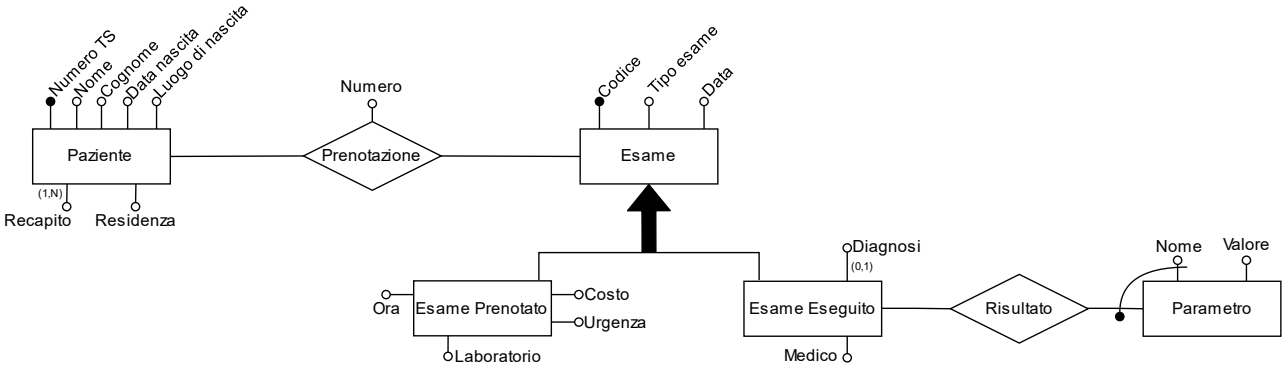
Costruzione dello schema E-R

Utilizzando un approccio bottom-up, ho cominciato a costruire lo schema E-R leggendo il testo della specifica disambiguata e disegnando passo passo le entità con i propri attributi, e quelle associazioni necessarie per disegnare le entità deboli. I disegni prodotti in questa prima fase sono riportati qui sotto.



Ho proseguito cercando di chiarire il concetto di esame, al momento presente nelle due entità "Esame" ed "Esame Prenotato"; per fare questo ho pensato di usare il pattern della storicizzazione sull'entità "Esame", sostituendo i concetti di *passato* e *futuro* rispettivamente con *eseguito* e *prenotato*. Si può notare che ho deciso di spostare l'attributo "Data" dall'entità figlia a quella madre, poiché ho ritenuto fosse importante tenerne traccia anche per gli esami eseguiti, e dell'introduzione di un nuovo attributo "Medico" nell'entità "Esame Eseguito", visto che nelle righe finali della specifica è richiesto sapere quali esami hanno eseguito i diversi membri del personale. Collegando "Esame" con l'entità "Paziente" attraverso l'associazione "Prenotazione" ottengo un piccolo sottoschema, che si aggiunge ai due già disegnati in precedenza, quello sul personale e quello sull'organizzazione della ASL.

Nella figura sopra, la generalizzazione di “Personale” con “Personale Volontario” e “Personale Medico” è rappresentata come una generalizzazione parziale, questo è un piccolo refuso, infatti in realtà è una generalizzazione totale.



Nei tre sottoschemi si può notare che molti attributi delle entità sono in realtà riferimenti ad entità in altri sottoschemi; questi attributi vanno dunque sostituiti con delle associazioni che colleghino concetti già esistenti.

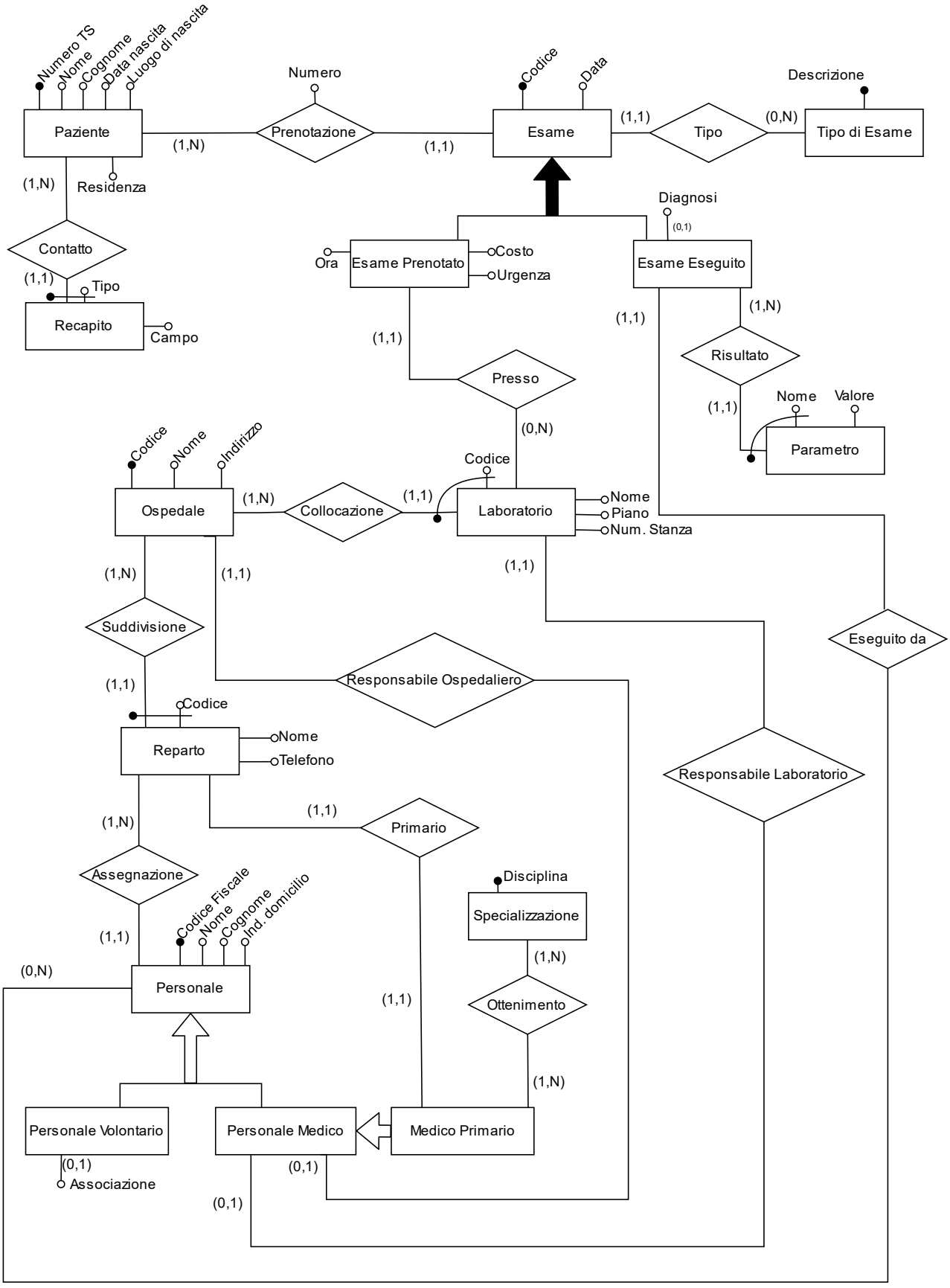
Rimangono infine alcuni attributi multipli da reificare, insieme all’attributo “Tipo” dell’entità “Esame” che ho deciso di reificare perché nella specifica viene chiesto che questi possano essere gestiti dagli amministratori di sistema.

Le associazioni e le reificazioni appena descritte si possono osservare in figura nel paragrafo che segue.

Integrazione finale

Come già anticipato, per integrare i tre sottoschemi ho “trasformato” alcuni attributi in associazioni. In particolare, ho reificato degli attributi usando entità già esistenti in altri sottoschemi e le associazioni per collegare queste con le entità che ora hanno perso l’attributo. Le associazioni aggiunte sono “Assegnazione”, “Responsabile Ospedaliero”, “Responsabile Laboratorio”, “Primario”, “Eseguito da” e “Presso”.

Anche nella figura qui sotto è presente il piccolo refuso della generalizzazione di “Personale”, la freccia che punta questa entità dovrebbe essere colorata di nero.


Numero TS
Nome
Cognome
Data nascita
Luogo di nascita
Paziente
(1,N)
Residenza
Numero
Prenotazione
(1,1)
Codice
Data
Esame
(1,1)
Tipo
(0,N)
Descrizione
Tipo di Esame
(1,N)
Contatto
(1,1)
Tipo
Recapito
Campo
Diagnosi
(0,1)
Ora
Esame Prenotato
Costo
Urgenza
Esame Eseguito
(1,1)
(1,N)
Risultato
(1,1)
Presso
(0,N)
Nome
Valore
Parametro
Codice
Nome
Indirizzo
Ospedale
(1,N)
Collocazione
(1,1)
Codice
Laboratorio
Nome
Piano
Num. Stanza
(1,N)
(1,1)
(1,1)
Suddivisione
Eseguito da
(1,1)
Responsabile Ospedaliero
Codice
Reparto
Nome
Telefono
Responsabile Laboratorio
(1,N)
(1,1)
Primario
Assegnazione
Disciplina
Specializzazione
(1,1)
Codice Fiscale
Nome
Cognome
Ind. domicilio
(1,N)
(0,N)
Personale
(1,1)
Ottenimento
(1,N)
Personale Volontario
Personale Medico
Medico Primario
(0,1)
Associazione
(0,1)
(0,1)

Regole aziendali

1. Esami dello stesso tipo prenotati dallo stesso paziente DEVONO avere data differente.
2. Esami prenotati dallo stesso paziente nello stesso giorno DEVONO essere di tipo differente.
3. Il responsabile di un laboratorio DEVE essere assegnato ad un reparto dell'ospedale dove si trova il laboratorio stesso.
4. Il responsabile di un ospedale DEVE essere assegnato ad un reparto dell'ospedale dove si trova il reparto stesso.
5. Il primario di un reparto DEVE essere assegnato al reparto di cui è primario.

Dizionario dei dati

Entità	Descrizione	Attributi	Identificatori
Paziente	Una persona che richiede un esame	Numero TS, Nome, Cognome, Data nascita, Luogo nascita, Residenza, Recapito	Numero TS
Prenotazione	Associa un paziente con un esame: rappresenta la richiesta del paziente di poter fare un esame	Numero, Paziente, Esame	Paziente, Esame
Esame	Un esame medico	Codice, Tipo esame, Data	Codice
Esame Prenotato	Un esame ancora da effettuare	Codice, Tipo Esame, Data, Ora, Costo, Urgenza	Codice
Esame Eseguito	Un esame svolto su un paziente, sono noti i risultati	Codice, Tipo Esame, Data, Diagnosi	Codice
Parametro	Parametro di un esame	Nome, Valore	Esame Eseguito (id. esterno) + Nome
Risultato	Associa un esame eseguito con un parametro	Esame Eseguito, Parametro	Esame Eseguito, Parametro
Ospedale	Edificio dove la ASL eroga i suoi servizi	Codice, Nome, Indirizzo	Codice

Collocazione	Associa un ospedale con un laboratorio	Ospedale, Laboratorio	Ospedale, Laboratorio
Laboratorio	Luogo in un ospedale dove si eseguono gli esami	Codice, Nome, Piano, Num. Stanza	Ospedale (id. esterno) + Codice
Presso	Associa un esame prenotato con un laboratorio: indica dove si effettuerà l'esame	Esame Prenotato, Laboratorio	Esame Prenotato, Laboratorio
Suddivisione	Associa un ospedale con un reparto	Ospedale, Reparto	Ospedale, Reparto
Reparto	Divisione del personale e degli ambienti di un ospedale	Codice, Nome, Telefono	Ospedale (id. esterno) + Codice
Assegnazione	Associa un reparto con un membro del personale	Reparto, Personale	Reparto, Personale
Personale	Un operatore sanitario della ASL	Cod. Fiscale, Nome, Cognome, Domicilio	Cod. Fiscale
Personale Volontario	Persona che lavora nella ASL a titolo gratuito	Cod. Fiscale, Nome, Cognome, Domicilio, Associazione	Cod. Fiscale
Personale Medico	Dottore della ASL	Cod. Fiscale, Nome, Cognome, Domicilio	Cod. Fiscale
Medico Primario	Dottore responsabile di un reparto	Cod. Fiscale, Nome, Cognome, Domicilio, Specializzazioni	Cod. Fiscale
Primario	Associa un reparto con un medico primario	Reparto, Medico Primario	Reparto, Medico Primario
Responsabile Ospedaliero	Associa un membro del personale medico con un ospedale	Ospedale, Personale Medico	Ospedale, Personale Medico
Responsabile Laboratorio	Associa un membro del personale medico con un laboratorio	Laboratorio, Personale Medico	Laboratorio, Personale Medico
Eseguito da	Associa un esame eseguito con un membro del personale: indica chi ha eseguito quell'esame	Esame Eseguito, Personale	Esame Eseguito, Personale
Tipo	Associa un esame con un tipo di esame	Esame, Tipo Esame	Esame, Tipo Esame
Tipo Esame	Descrizione dell'esame medico	Descrizione	Descrizione

Contatto	Associa un paziente e un recapito	Paziente, Recapito	Paziente, Recapito
Recapito	Email, cellulare o telefono di un paziente	Tipo, Campo	Paziente (id. esterno) + Tipo
Specializzazione	Specializzazione in una disciplina medica	Disciplina	Disciplina
Ottenimento	Associa una specializzazione ad un medico primario	Specializzazione, Medico Primario	Specializzazione, Medico Primario

4. Progettazione logica

Volume dei dati

Concetto nello schema	Tipo[1]	Volume atteso
Paziente	E	200.000
Prenotazione	R	5.000.000
Esame	E	5.000.000
Esame Prenotato	E	200.000
Esame Eseguito	E	4.800.000
Parametro	E	48.000.000
Risultato	R	48.000.000
Ospedale	E	5
Collocazione	R	25
Laboratorio	E	25
Presso	R	200.000
Suddivisione	R	50
Reparto	E	50
Assegnazione	R	500
Personale	E	1000
Personale Volontario	E	500
Personale Medico	E	500
Medico Primario	E	50
Primario	R	50
Responsabile Ospedaliero	R	5
Responsabile Laboratorio	R	25
Eseguito da	R	4.800.000
Recapito	E	400.000
Contatto	R	400.000
Tipo Esame	E	50

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

Tipo	R	5.000.000
Specializzazione	E	25
Ottenimento	R	50

Tavola delle operazioni

Cod.	Descrizione	Frequenza attesa
01	Inserire un nuovo paziente	5/giorno
02	Cancellare un paziente	5/giorno
03	Prenotare un esame	1.000/giorno
04	Inserire un nuovo tipo di esame	1/anno
05	Cancellare un tipo di esame	1/anno
06	Inserire una diagnosi ad un esame eseguito	1.000/giorno
07	Inserire un nuovo ospedale	1/ 5 anni
08	Cancellare un ospedale	1/ 10 anni
09	Inserire un membro del personale	10/mese
10	Cancellare un membro del personale	10/mese
11	Generare report mensile degli esami eseguiti da un membro del personale	500/mese
12	Generare un report annuale degli esami eseguiti da un membro del personale	500/anno
13	Report risultati esami in una prenotazione	1.000/giorno
14	Report storico esami di un paziente	100/giorno
15	Inserire i risultati di un esame	1.000/giorno

Costo delle operazioni

Operazione 01			
Concetto	**Costrutto**	**Accessi**	**Tipo**
Paziente	E	1	S
Recapito	E	1	S
Contatto	R	1	S

Costo: 6

Operazione 02			
Concetto	**Costrutto**	**Accessi**	**Tipo**
Paziente	E	1	S
Contatto	R	2	S
Recapito	E	2	S
Prenotazione	R	25	S
Esame	E	25	S
Tipo	R	25	S
Esame Eseguito	E	24	S
Eseguito da	R	24	S
Risultato	R	240	S
Parametro	E	240	S
Esame Prenotato	E	1	S
Presso	R	1	S

Costo: 1220

Operazione 03			
Concetto	**Costrutto**	**Accessi**	**Tipo**
Esame	E	1	S
Prenotazione	R	1	S
Tipo	R	1	S
Esame Prenotato	E	1	S
Presso	R	1	S

Costo: 10

Operazione 04			
Concetto	**Costrutto**	**Accessi**	**Tipo**
Tipo Esame	E	1	S

Costo: 2

Operazione 05			
Concetto	**Costrutto**	**Accessi**	**Tipo**
Tipo Esame	E	1	S
Tipo	R	100.000	S

Costo: 200.002

Operazione 06

Concetto	Costrutto	Accessi	Tipo
Esame Eseguito	E	1	S

Costo: 2

Operazione 07			
Concetto	**Costrutto**	**Accessi**	**Tipo**
Ospedale	E	1	S
Laboratorio	E	5	S
Collocazione	R	5	S
Reparto	E	10	S
Suddivisione	R	10	S

Costo: 62

Operazione 08			
Concetto	**Costrutto**	**Accessi**	**Tipo**
Ospedale	E	1	S
Collocazione	R	5	S
Laboratorio	E	5	S
Presso	R	1.000.000	S
Responsabile Laboratorio	R	5	S
Suddivisione	R	10	S
Reparto	E	10	S
Primario	R	10	S
Responsabile Ospedaliero	R	1	S
Assegnazione	R	200	S

Costo: 2.000.494

Operazione 09			
Concetto	**Costrutto**	**Accessi**	**Tipo**
Personale	E	1	S
Assegnazione	R	1	S
Personale Volontario / Personale Medico	E	1	S

Costo: 6

Operazione 10			
Concetto	**Costrutto**	**Accessi**	**Tipo**
Personale	E	1	S

Assegnazione	R	1	S
Eseguito da	R	5000	S
Personale Volontario / Personale Medico	E	1	S

Costo: 10.006

Operazione 11			
Concetto	**Costrutto**	**Accessi**	**Tipo**
Personale	E	1	L
Eseguito da	R	30	L
Esame Eseguito	E	30	L

Costo: 61

Operazione 12			
Concetto	**Costrutto**	**Accessi**	**Tipo**
Personale	E	1	L
Eseguito da	R	365	L
Esame Eseguito	E	365	L

Costo: 731

Operazione 13			
Concetto	**Costrutto**	**Accessi**	**Tipo**
Prenotazione	R	2	L
Esame Eseguito	E	2	L
Risultato	R	20	L
Parametro	E	20	L

Costo: 44

Operazione 14			
Concetto	**Costrutto**	**Accessi**	**Tipo**
Paziente	E	1	L
Prenotazione	R	25	L
Esame Eseguito	E	25	L
Risultato	R	250	L
Parametro	E	250	L

Costo: 551

Operazione 15

Concetto	Costrutto	Accessi	Tipo
Esame Prenotato	E	1	S
Presso	R	1	S
Esame Eseguito	E	1	S
Risultato	R	10	S
Parametro	E	10	S

Costo: 46

Ristrutturazione dello schema E-R

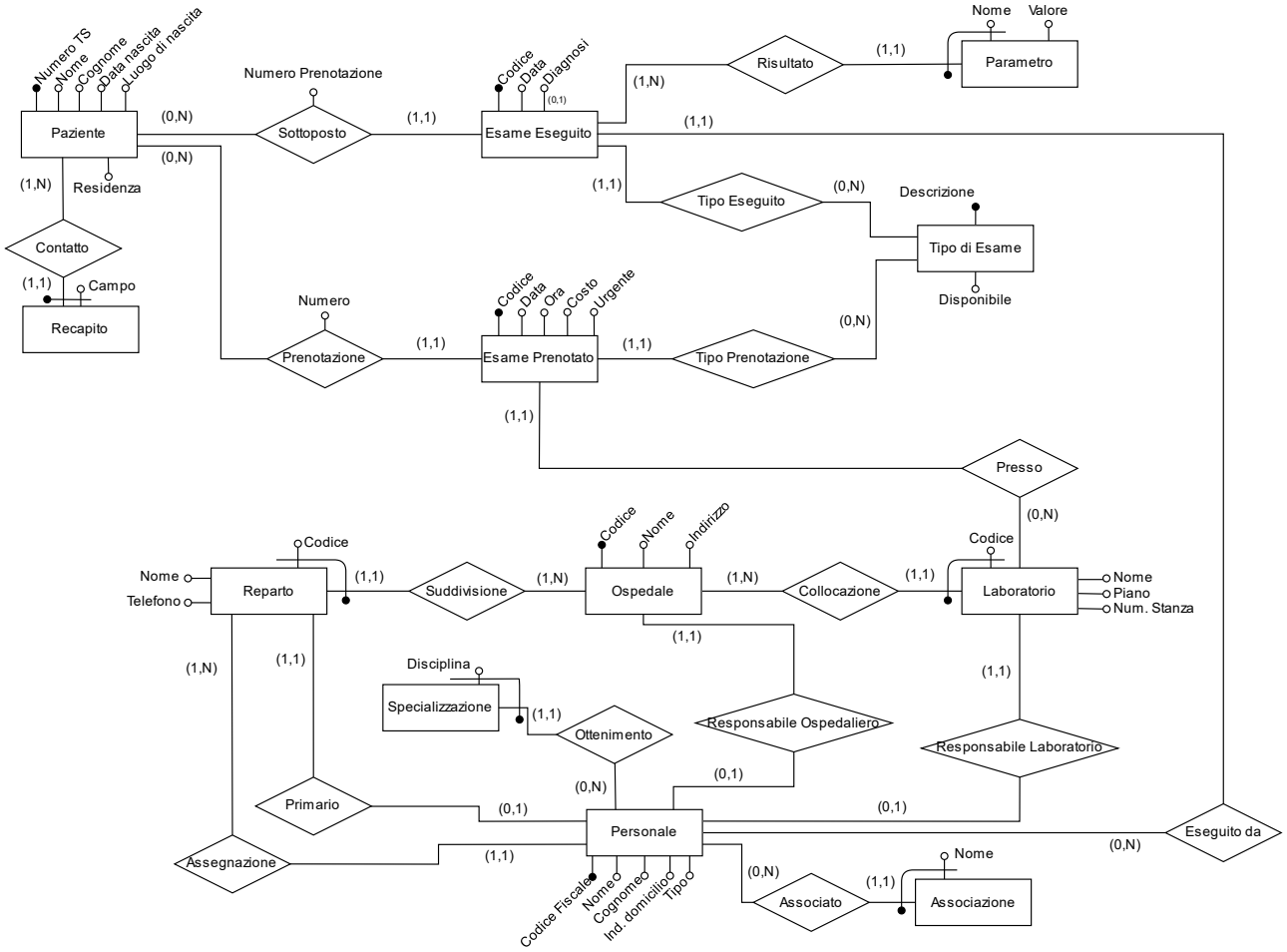
Lo schema E-R frutto della fase di progettazione concettuale non presenta ridondanze eliminabili come attributi derivabili o associazioni derivabili, infatti i "cicli" di associazioni che si possono riscontrare nello schema portano tutti con loro un "valore semantico" e per questo la rimozione dei cicli implicherebbe la perdita di completezza dello schema.

Tre generalizzazioni sono state eliminate con due approcci: la gerarchia dell'entità "Esame" è stata eliminata accorpando il genitore nei figli, questo perché ha senso accedere in modo separato agli esami già effettuati e a quelli ancora da effettuare, inoltre questi due concetti portano con loro diversi attributi non condivisi; la gerarchia del personale, con anche la generalizzazione Personale Medico – Medico Primario, è stato eliminato accorpando tutto nell'entità genitore ed inserendo un attributo "Tipo", questo perché le entità figlie si differenziavano per lo più per le associazioni in cui partecipavano piuttosto che per gli attributi non condivisi. Per fare questo ho anche deciso di reificare l'attributo opzionale "Associazione" di "Personale Volontario" perché avrebbe avuto valore NULL per almeno la metà delle istanze della nuova entità "Personale" che accorpa ora tutte le sue specializzazioni.

Considerando l'operazione 05 (cancellare un tipo di esame), ho deciso di introdurre per l'entità "Tipo Esame" un attributo "Disponibile", questo perché un tipo di esame potrebbe essere non erogato per un periodo limitato di tempo e perché nel caso un tipo di esame non fosse più erogato dalla ASL non si vuole perderne traccia per gli esami già effettuati di quel tipo. L'operazione 05 dunque cambia leggermente nome in "rendere non disponibile un tipo di esame" e il suo costo scende da 200.002 a 2. Va aggiunta una regola aziendale che vieti la prenotazione per tipi di esame non disponibili.

Le attività di ristrutturazione hanno comportato la necessità di aggiungere ulteriori regole aziendali. Ho rimosso la regola aziendale 1 perché "ridondante" con la 2.

Infine, ho modificato l'entità Recapito eliminando l'attributo tipo e usando come chiave la coppia Paziente-Campo perché in questo modo si possono mettere più recapiti dello stesso tipo.



Regole Aziendali:

1. Esami prenotati dallo stesso paziente nello stesso giorno DEVONO essere di tipo differente.
2. Il responsabile di un laboratorio DEVE essere assegnato ad un reparto dell'ospedale dove si trova il laboratorio stesso.
3. Il responsabile di un ospedale DEVE essere assegnato ad un reparto dell'ospedale dove si trova il reparto stesso.
4. Il primario di un reparto DEVE essere assegnato al reparto di cui è primario.
5. I membri del personale che sono responsabili di ospedale o laboratorio DEVONO avere tipo "medico" o "primario".
6. I membri del personale che sono primari di un reparto DEVONO avere tipo "primario".
7. I membri del personale che hanno tipo "medico" o "primario" NON DEVONO essere associati con una associazione di volontariato.

8. Un qualsiasi esame eseguito NON DEVE avere lo stesso codice di un qualsiasi esame prenotato.
9. Il primario di un reparto DEVE avere almeno una specializzazione.
10. Specializzazione NON DEVE essere associata con un membro del personale di tipo “medico” o “volontario”.
11. Un nuovo esame prenotato NON DEVE essere associato ad un tipo di esame non disponibile.

Trasformazione di attributi e identificatori

Le entità Reparto e Laboratorio acquisiscono un nuovo attributo “ID” usato come identificatore al posto della coppia (“Codice”, “Ospedale”). Questa trasformazione non è riportata nella figura precedente perché arriva indietro come miglioramento dalla fase di progettazione fisica, verrà però considerata nei paragrafi che seguono.

Un altro cambiamento avvenuto in fase di progettazione fisica è l’aggiunta dell’attributo “Misura” alla entità Parametro; questa aggiunta non viene riportata nei paragrafi prima del capitolo 5, eccetto che nel paragrafo “Normalizzazione del modello relazionale” poiché c’è un importante appunto da fare.

Traduzione di entità e associazioni

PAZIENTE (NumTs, Nome, Cognome, DataNascita, LuogoNascita, Residenza)

RECAPITO (Paziente, Campo)

RECAPITO(Paziente) ⊂ PAZIENTE(NumTS)

ESAME ESEGUITO (Codice, Data, Diagnosi*, NumPrenotazione, Paziente, Tipo, Medico)

ESAME ESEGUITO(Paziente) ⊂ PAZIENTE(NumTS)
ESAME ESEGUITO(Tipo) ⊂ TIPO ESAME(Nome)
ESAME ESEGUITO(Medico) ⊂ PERSONALE(CodFiscale)

ESAME PRENOTATO (Codice, Data, Ora, Costo, Urgente, NumPrenotazione, Paziente, Tipo, Laboratorio)

ESAME PRENOTATO(Paziente) ⊂ PAZIENTE(NumTS)
ESAME PRENOTATO(Tipo) ⊂ TIPO ESAME(Nome)
ESAME PRENOTATO(Laboratorio) ⊂ LABORATORIO(Id)

TIPO ESAME (Nome, Disponibile)

PARAMETRO (Esame, Nome, Valore)

PARAMETRO(Esame) ⊂ ESAME ESEGUITO(Codice)

OSPEDALE (Codice, Nome, Indirizzo, Responsabile)

OSPEDALE(Responsabile) ⊂ PERSONALE(CodFiscale)

LABORATORIO (Id, Ospedale, Codice, Nome, Piano, NumStanza, Responsabile)

LABORATORIO(Ospedale) ⊂ OSPEDALE(Codice)

LABORATORIO(Responsabile) ⊂ PERSONALE(CodFiscale)

REPARTO (Id, Ospedale, Codice, Nome, Telefono, Primario)

REPARTO(Ospedale) ⊂ OSPEDALE(Codice)

REPARTO(Primario) ⊂ PERSONALE(CodFiscale)

PERSONALE (CodFiscale, Nome, Cognome, Domicilio, Tipo, Reparto)

PERSONALE(Reparto) ⊂ REPARTO(Id)

SPECIALIZZAZIONE (Primario, Disciplina)

SPECIALIZZAZIONE(Primario) ⊂ PERSONALE(CodFiscale)

ASSOCIAZIONE (Volontario, Nome)

ASSOCIAZIONE(Volontario) ⊂ PERSONALE(CodFiscale)

Normalizzazione del modello relazionale

Possiamo dire che dopo l'attività di ristrutturazione ogni relazione è "per costruzione" in 1NF, poiché ogni relazione ha una chiave primaria, gli attributi sono definiti su valori atomici e non ci sono attributi multipli.

La 2NF richiede che non ci siano *dipendenze parziali*, ovvero che non esistano dipendenze fra sottoinsiemi propri della chiave e altri attributi; dunque, tutte le relazioni che hanno una chiave composta da un solo attributo lo sono automaticamente, così come le relazioni che non hanno attributi che non siano parte della chiave. L'unica relazione che non rientra nelle caratteristiche appena descritte è la relazione Parametro:

- PARAMETRO(Esame, Nome, Valore): ogni esame ha più di un valore, quindi non può essere Esame→Valore; vale lo stesso per Nome, infatti lo stesso parametro sarà riportato in diversi esami che avranno valori diversi, dunque non può essere Nome→Valore.

Per verificare se una relazione *r* sia in 3FN bisogna vedere se ogni dipendenza funzionale non banale $X \rightarrow A$ si verifichi almeno una tra le seguenti condizioni:

- *X* contiene una chiave *K* di *r*
- *A* appartiene ad almeno una chiave di *r*

Per molte relazioni non c'è bisogno di particolari verifiche poiché l'unica chiave è la stessa scelta come identificatore; discorso diverso per altre, che sono Esame Eseguito, Esame Prenotato, Ospedale, Laboratorio e Reparto: ognuna di queste relazioni ha anche almeno un'altra chiave, ma è facile vedere che tutte le dipendenze funzionali non banali sono del tipo X→A con X sempre una chiave della relazione.

Nel capitolo 5, riguardante la progettazione fisica, si potrà notare che la relazione Parametro ha acquisito un nuovo attributo "Misura", come anticipato nel paragrafo "Trasformazione di attributi e identificatori". Questo attributo viene utilizzato per salvare l'informazione sull'unità di misura relativa a un parametro di un esame eseguito, dato che l'attributo "Valore" preso da solo nel suo essere numero non è una informazione utile. Distrattamente, questa idea non mi è minimamente venuta in mente in fase di progettazione concettuale e logica, fino a quando, nello stampare i valori di esami eseguiti, in progettazione fisica ben che avviata, la cosa non è stata lampante.

La relazione Parametro, divenuta PARAMETRO(Esame, Nome, Valore, Misura), va riconsiderata per quanto riguarda le forme normali; infatti si potrebbe avere la presenza della dipendenza parziale Nome→Misura che renderebbe la relazione non in 2NF. Il progetto da qui in avanti non considererà Nome→Misura una dipendenza parziale, ma riporto qui la decomposizione che sarebbe stata necessaria affinché lo schema fosse stato in 2NF e 3NF:

- schema non normalizzato: PARAMETRO(Esame, Nome, Valore, Misura)
- schema normalizzato:
  - ✓ RISULTATO(Esame, Parametro, Valore)
  - ✓ PARAMETRO(Nome, Misura)

5. Progettazione fisica

Utenti e privilegi

Sono stati creati i seguenti utenti: cup, login, amministratore, personale.

Gli utenti non hanno privilegi su nessuna tabella, ma solamente privilegi di esecuzione su Stored Procedures per prevenire attacchi SQL injection.

Vengono riportati gli utenti e le stored procedures sulle quali hanno i privilegi di EXECUTE:

Utente	**Procedures con privilegio di EXECUTE**
login	login
cup	anagrafica_paziente, cancella_paziente, cancella_recapito, esami_disponibili, inserisci_paziente, inserisci_recapito, lista_laboratori, lista_recapiti, modifica_paziente, prenota_esame, report_prenotazione, report_storico_paz
amministratore	aggiorna_associazione_vol, aggiungi_specializzazione, anagrafica_personale, cancella_laboratorio, cancella_ospedale, cancella_personale, cancella_reparto, crea_utente, info_ospedale, inserisci_laboratorio, inserisci_ospedale, inserisci_personale, inserisci_reparto, inserisci_tipo_esame, lista_ospedali, modifica_laboratorio, modifica_ospedale, modifica_personale, modifica_reparto, report_esami_eseguiti, stato_tipo_esame
personale	aggiorna_diagnosi, esegui_esame, inserisci_risultati

Strutture di memorizzazione

Tabella associazione		
Attributo	**Tipo di dato**	**Attributi**[2]
Volontario	Varchar(16)	PK, NN
Nome	Varchar(45)	NN

Tabella esame_eseguito

[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.

Attributo	Tipo di dato	Attributi
Codice	Int	PK, NN
Data	Date	NN
Prenotazione	Int	NN
Paziente	Int	NN
Tipo	Varchar(45)	NN
Medico	Varchar(16)	NN
Diagonosi	Varchar(256)	

Tabella esame_prenotato		
Attributo	**Tipo di dato**	**Attributi**
Codice	Int	PK, NN, AI
Data	Date	NN
Ora	Time	NN
Costo	Float	NN
Urgente	Tinyint	NN
Prenotazione	Int	
Paziente	Int	NN
Tipo	Varchar(45)	NN
Laboratorio	Int	NN

Tabella laboratorio		
Attributo	**Tipo di dato**	**Attributi**
Id	Int	PK, NN, AI
Codice	Int	NN
Ospedale	Int	NN
Nome	Varchar(45)	NN
Piano	Int	NN
Stanza	Int	NN
Responsabile	Varchar(16)	NN

Tabella ospedale		
Attributo	**Tipo di dato**	**Attributi**
Codice	Int	PK, NN, AI
Nome	Varchar(45)	NN
Indirizzo	Varchar(45)	NN
Responsabile	Varchar(16)	NN

Tabella parametro		
Attributo	**Tipo di dato**	**Attributi**
Esame	Int	PK, NN
Nome	Varchar(45)	PK, NN
Valore	Double	NN
Misura	Varchar(45)	

Tabella paziente		
Attributo	**Tipo di dato**	**Attributi**
Num_ts	Int	PK, NN
Cf	Varchar(16)	NN
Nome	Varchar(45)	NN
Cognome	Varchar(45)	NN
Data_nascita	Date	NN
Luogo_nascita	Varchar(45)	NN
Residenza	Varchar(45)	NN

Tabella personale		
Attributo	**Tipo di dato**	**Attributi**
Cf	Varchar(16)	PK, NN
Nome	Varchar(45)	NN
Cognome	Varchar(45)	NN
Domicilio	Varchar(45)	NN

Tipo	Enum(‘medico’, ‘primario’, ‘volontario’)	NN
Reparto	Int	NN
Username	Varchar(45)	NN

Tabella recapito		
Attributo	**Tipo di dato**	**Attributi**
Paziente	Int	PK, NN
Campo	Varchar(45)	PK. NN

Tabella reparto		
Attributo	**Tipo di dato**	**Attributi**
Id	Int	PK, NN, AI
Codice	Int	NN
Ospedale	Int	NN
Nome	Varchar(45)	NN
Telefono	Varchar(45)	NN
Primario	Varchar(16)	NN

Tabella specializzazione		
Attributo	**Tipo di dato**	**Attributi**
Primario	Varchar(16)	PK, NN
Disciplina	Varchar(45)	PK, NN

Tabella tipo_esame		
Attributo	**Tipo di dato**	**Attributi**
Nome	Varchar(45)	PK, NN
Disponibile	Tinyint	NN

Tabella utenti

Attributo	Tipo di dato	Attributi
Username	Varchar(45)	PK, NN
Password	Char(32)	NN
Ruolo	Enum('amministratore', 'cup', 'personale')	NN

Indici

Tutti gli indici con nome "PRIMARY" e quelli che iniziano con "fk_" sono stati generati automaticamente da MySQL Workbench rispettivamente per le chiavi e le foreign key di ogni tabella, pertanto non mi soffermerò a commentare nessuno di questi indici.

Tabella associazione	
Indice PRIMARY	**Tipo[3]:**
Volontario	PR

Tabella esame_eseguito	
Indice PRIMARY	**Tipo:**
Codice	PR
Indice fk_esame_eseguito_paz_idx	**Tipo:**
Paziente	IDX
Indice fk_esame_eseguito_tipo_idx	**Tipo:**
Tipo	IDX

Tabella esame_prenotato	
Indice PRIMARY	**Tipo:**
Codice	PR
Indice fk_esame_prenotato_paz_idx	**Tipo:**

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

Paziente	IDX
Indice fk_esame_prenotato_tipo_idx	**Tipo:**
Tipo	IDX
Indice fk_esame_prenotato_lab_idx	**Tipo:**
Laboratorio	IDX
Indice uq_paziente_tipo_data	**Tipo:**
Paziente, Tipo, Data	UQ

L'indice "uq_paziente_tipo_data" è stato inserito per implementare la Regola Aziendale 1.

Tabella laboratorio	
Indice PRIMARY	**Tipo:**
ID	PR
Indice fk_laboratorio_osp_idx	**Tipo:**
Ospedale	IDX
Indice fk_laboraorio_resp_idx	**Tipo:**
Responsabile	IDX
Indice uq_lab_osp	**Tipo:**
Codice, Ospedale	UQ

L'indice "uq_lab_osp" è stato inserito per avere il codice di laboratorio univoco all'interno dello stesso ospedale. Laboratori in ospedali diversi, dunque, potranno avere anche lo stesso codice.

Tabella ospedale	
Indice PRIMARY	**Tipo:**
Codice	PR
Indice fk_ospedale_resp_idx	**Tipo:**
Responsabile	IDX

Tabella parametro

Indice PRIMARY	**Tipo:**
Esame, Nome	PR

Tabella paziente	
Indice PRIMARY	**Tipo:**
Num_ts	PR

Tabella personale	
Indice PRIMARY	**Tipo:**
CF	PR
Indice fk_personale_reparto_idx	**Tipo:**
Reparto	IDX
Indice fk_personale_username_idx	**Tipo:**
Username	IDX

Tabella recapito	
Indice PRIMARY	**Tipo:**
Paziente, Campo	PR

Tabella reparto	
Indice PRIMARY	**Tipo:**
ID	PR
Indice fk_reparto_osp_idx	**Tipo:**
Ospedale	IDX
Indice fk_reparto_primario_idx	**Tipo:**
Primario	IDX
Indice uq_rep_osp	**Tipo:**
Codice, Ospedale	UQ

L'indice "uq_rep_osp" è stato inserito per avere il codice di reparto univoco all'interno dello stesso ospedale. Reparti in ospedali diversi, dunque, potranno avere anche lo stesso codice.

Tabella specializzazione	
Indice PRIMARY	**Tipo:**
Primario, Disciplina	PR

Tabella tipo_esame	
Indice PRIMARY	**Tipo:**
Nome	PR

Tabella utenti	
Indice PRIMARY	**Tipo:**
Username	PR

Trigger

Per la tabella "associazione":

associazione_BEFORE_INSERT

```
CREATE DEFINER = CURRENT_USER TRIGGER `gestione_asl`.`associazione_BEFORE_INSERT` BEFORE INSERT
ON `associazione` FOR EACH ROW

BEGIN
  declare var_tipo enum('medico', 'primario', 'volontario');

  select `tipo` into var_tipo
  from `personale`
  where `cf` = new.`volontario`;

  If var_tipo <> 'volontario' then
    signal sqlstate '45000' set message_text = 'The personnel memeber is not "volontario" type';
  end if;
END
```

associazione_BEFORE_UPDATE

```
CREATE DEFINER = CURRENT_USER TRIGGER `gestione_asl`.`associazione_BEFORE_UPDATE` BEFORE UPDATE
ON `associazione` FOR EACH ROW

BEGIN
```

```
    declare var_tipo enum('medico', 'primario', 'volontario');

    select `tipo` into var_tipo
    from `personale`
    where `cf` = new.`volontario`;

    if var_tipo <> 'volontario' then
       signal sqlstate '45000' set message_text = 'The personnel memeber is not "volontario"
type';
    end if;
END
```

Per la tabella “esame_prenotato”:

esame_prenotato_BEFORE_INSERT

```
CREATE DEFINER = CURRENT_USER TRIGGER `gestione_asl`.`esame_prenotato_BEFORE_INSERT` BEFORE
INSERT ON `esame_prenotato` FOR EACH ROW

BEGIN
    -- date must be in the future
    if new.data <= curdate() then
       signal sqlstate '45000' set message_text = 'Reserving an exam today or in the past is
impossible';
    end if;
END
```

Per la tabella “personale”:

personale_BEFORE_INSERT

```
CREATE DEFINER = CURRENT_USER TRIGGER `gestione_asl`.`personale_BEFORE_INSERT` BEFORE INSERT ON
`personale` FOR EACH ROW

BEGIN
    declare var_ruolo enum('amministratore', 'cup', 'personale');

    select `ruolo` into var_ruolo
    from `utenti`
    where `utenti`.`username` = new.`username`;

    if var_ruolo <> 'personale' then
       signal sqlstate '45000' set message_text = 'This user has not role "personale" so cannot
be linked with a personnel member';
    end if;
END
```

personale_BEFORE_UPDATE

```
CREATE DEFINER = CURRENT_USER TRIGGER `gestione_asl`.`personale_BEFORE_UPDATE` BEFORE UPDATE ON
`personale` FOR EACH ROW
BEGIN
    declare var_ruolo enum('amministratore', 'cup', 'personale');

    select `ruolo` into var_ruolo
    from `utenti`
    where `utenti`.`username` = new.`username`;
```

```
    if var_ruolo <> 'personale' then
       signal sqlstate '45000' set message_text = 'This user has not role "personale" so cannot
be linked with a personnel member';
    end if;
END
```

personale_AFTER_UPDATE

```
CREATE DEFINER = CURRENT_USER TRIGGER `gestione_asl`.`personale_AFTER_UPDATE` AFTER UPDATE ON
`personale` FOR EACH ROW
BEGIN
    if old.`tipo` = 'primario' AND new.`tipo` <> 'primario' then
       delete from `specializzazione`
       where `primario` = old.`cf`;
    end if;
END
```

Per la tabella “specializzazione”:

specializzazione_BEFORE_INSERT

```
CREATE DEFINER = CURRENT_USER TRIGGER `gestione_asl`.`specializzazione_BEFORE_INSERT` BEFORE
INSERT ON `specializzazione` FOR EACH ROW
BEGIN
    declare var_tipo enum('medico', 'primario', 'volontario');

    select `tipo` into var_tipo
    from `personale`
    where `cf` = new.`primario`;

    if var_tipo <> 'primario' then
       signal sqlstate '45000' set message_text = 'The personnel memeber is not "primario" type';
    end if;
END
```

specializzazione_BEFORE_UPDATE

```
CREATE DEFINER = CURRENT_USER TRIGGER `gestione_asl`.`specializzazione_BEFORE_UPDATE` BEFORE
UPDATE ON `specializzazione` FOR EACH ROW
BEGIN
    declare var_tipo enum('medico', 'primario', 'volontario');

    select `tipo` into var_tipo
    from `personale`
    where `cf` = new.`primario`;

    if var_tipo <> 'primario' then
       signal sqlstate '45000' set message_text = 'The personnel memeber is not "primario" type';
    end if;
END
```

Eventi

Non sono stati implementati eventi.

Viste

Non sono state implementate viste.

Stored Procedures e transazioni

aggiorna_associazione_vol

```
CREATE PROCEDURE `aggiorna_associazione_vol` (in var_cf varchar(16), in var_associazione
varchar(45))
BEGIN
    declare exist int;

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

    set transaction isolation level repeatable read;
    start transaction;
        select count(*) into exist
        from `associazione`
        where `volontario` = var_cf;

        if exist = 0 then
                    insert into `associazione` values (var_cf, var_associazione);
        elseif exist = 1 then
                    update `associazione`
                    set `nome` = var_associazione
                    where `volontario` = var_cf;
        end if;

    commit;
END
```

Per inserire o modificare l’associazione di volontariato di un volontario.

aggiorna_diagnosi

```
CREATE PROCEDURE `aggiorna_diagnosi` (in var_codice_esame int, in var_username varchar(45), in
var_diagnosi varchar(256))
BEGIN

    declare var_caller varchar(16);
    declare var_medico varchar(16);
    declare exist int;

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

    set transaction isolation level repeatable read;
    start transaction;
```

```
        select count(*) into exist
        from `esame_eseguito`
        where `codice` = var_codice_esame;

        if exist <> 1 then
              signal sqlstate '45000' set message_text = 'An exam with this code does not exist
or is not yet executed';
       end if;

       -- 1. check if that personnel member actually executed that exam
       select `cf` into var_caller
       from `personale`
       where `username` = var_username;

       select `medico` into var_medico
       from `esame_eseguito`
       where `codice` = var_codice_esame;

       if var_caller <> var_medico then
              signal sqlstate '45009' set message_text = 'You cannot modify this exam because you
did not executed it';
       end if;

       -- 2. update
       update `esame_eseguito`
       set `diagnosi` = var_diagnosi
       where `codice` = var_codice_esame;

   commit;
END
```

Per inserire o modificare la diagnosi di un esame eseguito.

aggiungi_specializzazione

```
CREATE PROCEDURE `aggiungi_specializzazione` (in var_cf varchar(16), in var_specializzazione
varchar(45))
BEGIN
       insert into `specializzazione` values (var_cf, var_specializzazione);
END
```

Per inserire ulteriori specializzazioni di un primario.

anagrafica_paziente

```
CREATE PROCEDURE `anagrafica_paziente` (in var_num_ts int)
BEGIN
       set transaction isolation level read committed;
       start transaction;
              select `num_ts` as 'Numero TS', `nome` as 'Nome', `cognome` as 'Cognome',
`data_nascita` as 'Data nasc.', `luogo_nascita` as 'Luogo di nascita', `residenza` as 'Residenza'
              from `paziente` where `num_ts` = var_num_ts;
       commit;
END
```

Per stampare le informazioni relative a un paziente.

anagrafica_personale

```
CREATE PROCEDURE `anagrafica_personale` (in var_cf varchar(16))
BEGIN
       declare var_tipo enum('medico', 'primario', 'volontario');

       set transaction isolation level repeatable read;
       start transaction;
```

```
            select `cf` as 'Cod. Fiscale', `nome` as 'Nome', `cognome` as 'Cognome',
`domicilio` as 'Domicilio', `tipo` as 'Tipo', `reparto` as 'Reparto', `username`
            from `personale` where `cf` = var_cf;

        select `tipo` into var_tipo
        from `personale` where `cf` = var_cf;

        -- print the volounteer's "associazione" or the degrees of the ward responsible
        if var_tipo = 'primario' then
                    select `disciplina` as 'Specializzazione'
            from `specializzazione` where `primario` = var_cf;
        elseif var_tipo = 'volontario' then
                    select `nome` as 'Associazione di Volontariato'
            from `associazione` where `volontario` = var_cf;
        end if;

       commit;
END
```

Per stampare le informazioni relative ad un membro del personale.

cancella_laboratorio

```
CREATE PROCEDURE `cancella_laboratorio` (in var_id int)
BEGIN
    declare num_lab int;
    declare var_osp int;

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

    set transaction isolation level serializable;
    -- repeatable read is not enough because reads from a snapshot
    -- so num_lab can be not consistent with the state of the DB
    -- and is important to have it consistent to prevent the deletion of the only lab in a
hospital
    start transaction;

        select `ospedale` into var_osp
        from `laboratorio`
        where `id` = var_id;

        select count(*) into num_lab
        from `laboratorio`
        where `ospedale` = var_osp;

        if num_lab < 2 then
              signal sqlstate '45000' set message_text = 'the last lab in the hospital cannot be
deleted';
        end if;

        delete from `laboratorio` where `id` = var_id;

    commit;
END
```

Per eliminare un laboratorio dal database. Controlla che non si stia eliminando l'unico laboratorio interno ad un ospedale.

cancella_ospedale

```
CREATE PROCEDURE `cancella_ospedale` (in var_codice int, out var_new_reparto int)
BEGIN
    declare exist int;
    declare num_osp int;
    declare new_reparto int;

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

    set transaction isolation level serializable;
    -- repeatable read is not enough because reads from a snapshot
    -- so num_osp can be not consistent with the state of the DB
    -- and is important to have it consistent to prevent the deletion of the only hospital in the
ASL
    start transaction;
            -- check if the hospital exists and is not the only hospital in the ASL
        select count(*) into exist
        from `ospedale`
        where `codice` = var_codice;

        if exist < 1 then
                    signal sqlstate '45000' set message_text = 'This hospital does not exist';
        end if;

        select count(*) into num_osp
        from `ospedale`;

        if num_osp < 2 then
              signal sqlstate '45000' set message_text = 'Cannot delete the last hospital';
       end if;

        -- find a hospital and a ward to move the personnel of the hospital being deleted
        select max(`id`) into new_reparto
        from `reparto`
        where `reparto`.`ospedale` = (select max(`codice`) from `ospedale` where `codice` <>
var_codice);

        set var_new_reparto = new_reparto;

        -- convert ward responsibles ('primario' type) in 'medico', delete their "degrees" first
        delete from `specializzazione`
              where `primario` in ( select `cf`
                                    from `personale` join `reparto` on `personale`.`reparto` =
`reparto`.`id`
                                   where `reparto`.`ospedale` = var_codice
                                    );

        update `personale`
        set `tipo` = 'medico'
        where `tipo` = 'primario' and `reparto` in (      select `id`
                                                          from `reparto`
                                                          where `reparto`.`ospedale` = var_codice
                                                 );

       -- move the personnel
       update `personale`
       set `reparto` = new_reparto
       where `reparto` in (  select `id`
                             from `reparto`
                             where `reparto`.`ospedale` = var_codice
                             );

        -- delete the hospital, the laboratories and wards will be deleted on cascade
```

```
        delete from `ospedale` where `codice` = var_codice;

  commit;
END
```

Per eliminare un ospedale dal database. Controlla che non si stia eliminando l'unico ospedale della ASL. Prima dell'eliminazione, sposta tutto il personale dell'ospedale in cancellazione in un altro ospedale della ASL.

cancella_paziente

```
CREATE PROCEDURE `cancella_paziente` (in var_num_ts int)
BEGIN
    declare patient_exists int;

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

    set transaction isolation level repeatable read;
    start transaction;
       select count(num_ts) into patient_exists from `paziente` where `num_ts` = var_num_ts;

       if patient_exists = 1 then
           delete from `paziente` where `num_ts` = var_num_ts;
       else
           signal sqlstate '45003' set message_text = 'Deleting a non existent patient';
      end if;
   commit;
END
```

Per eliminare un paziente dal database.

cancella_personale

```
CREATE PROCEDURE `cancella_personale` (in var_cf varchar(16))
BEGIN
    declare exist int;
    declare is_resp_lab int;
    declare is_resp_osp int;
    declare is_primario int;

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

    set transaction isolation level serializable;
    start transaction;

        select count(*) into exist
        from `personale`
        where `cf` = var_cf;

        if exist < 1 then
              signal sqlstate '45000' set message_text = 'The personnel member you are trying to
update does not exists';
        end if;

        -- check if is responsible of a hospital/laboratory/ward
```

```
        select count(*) into is_resp_lab
        from `laboratorio`
        where `responsabile` = var_cf;

        select count(*) into is_resp_osp
        from `ospedale`
        where `responsabile` = var_cf;

        select count(*) into is_primario
        from `reparto`
        where `primario` = var_cf;

        if (is_resp_lab + is_resp_osp + is_primario > 0) then
              signal sqlstate '45001' set message_text = 'Cannot delete this personnel member
because she/he is responsible for something. Move this personnel member first.';
        end if;

        delete from `personale` where `cf` = var_cf;

    commit;
END
```

Per eliminare un membro del personale dal database. Blocca l'eliminazione se il membro del personale è responsabile di un ospedale o di un laboratorio oppure se è un primario.

cancella_recapito

```
CREATE PROCEDURE `cancella_recapito` (in var_num_ts int, in var_campo varchar(45))
BEGIN
    declare num_recapiti int;
    declare contact_exists int;

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

    set transaction isolation level repeatable read;
    start transaction;
       -- check if the contact the user want to delete actually exists
       select count(campo) into contact_exists     from `recapito` where `paziente` = var_num_ts
and `campo` = var_campo;

       if contact_exists < 1 then
              signal sqlstate '45001' set message_text = 'The contact you are trying to delete
does not exists';
       end if;

       select count(paziente) from `recapito` where `paziente` = var_num_ts into num_recapiti;

       if num_recapiti > 1 then
              delete from `recapito` where `paziente` = var_num_ts and `campo` = var_campo;
       else
              signal sqlstate '45002' set message_text = 'Deleting the only contact for this
patient is forbidden';
       end if;
    commit;
END
```

Per eliminare un recapito di un paziente. Blocca l'eliminazione se si sta cancellando l'unico recapito di un paziente.

cancella_reparto

```
CREATE PROCEDURE `cancella_reparto` (in var_id int, out var_new_reparto int)
BEGIN
    declare exist int;
    declare var_osp int;
    declare num_reparti int;
    declare new_reparto int;

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

    set transaction isolation level serializable;
    -- repeatable read is not enough because reads from a snapshot
    -- so num_reparti can be not consistent with the state of the DB
    -- and is important to have it consistent to prevent the deletion of the only ward in a
hospital
    start transaction;
        -- check if the ward exist and is not the only one in its hospital
        select count(*) into exist
        from `reparto`
        where `id` = var_id;

        if exist < 1 then
              signal sqlstate '45000' set message_text = 'This ward does not exist';
        end if;

        select `ospedale` into var_osp
       from `reparto`
       where `id` = var_id;

        select count(*) into num_reparti
        from `reparto`
        where `ospedale` = var_osp;

        if num_reparti < 2 then
              signal sqlstate '45000' set message_text = 'Cannot delete the last ward in the
hospital';
       end if;

        -- find a ward to move the personnel in the ward being deleted
        select max(`id`) into new_reparto
        from `reparto`
        where `reparto`.`ospedale` = var_osp and `id` <> var_id;

        set var_new_reparto = new_reparto;

        -- convert the responisble in 'medico', delete her/his degrees first
        delete from `specializzazione`
       where `primario` in ( select `cf`
                             from `personale`
                             where `reparto` = var_id
                             );

        update `personale`
        set `tipo` = 'medico'
        where `tipo` = 'primario' and `reparto` = var_id;

        -- move the personnel in the other ward
       update `personale`
       set `reparto` = new_reparto
       where `reparto` = var_id;
```

```
        delete from `reparto` where `id` = var_id;
    commit;
END
```

Per eliminare un reparto dal database. Blocca l'eliminazione dell'unico reparto in un ospedale. Trasferisce il personale assegnato al reparto in eliminazione ad un altro reparto.

crea_utente

```
CREATE PROCEDURE `crea_utente` (IN username VARCHAR(45), IN pass VARCHAR(45), IN ruolo
varchar(45))
BEGIN
        insert into utenti VALUES(username, MD5(pass), ruolo);
END
```

Permette di creare un nuovo account interno al database (non un utente MySQL) per accedere ed utilizzare il programma a seconda del ruolo.

esami_disponibili

```
CREATE PROCEDURE `esami_disponibili` ()
BEGIN
    set transaction isolation level read committed;
    start transaction;
        select `nome` as 'Tipo Esame' from `tipo_esame` where `disponibile` = true;
    commit;
END
```

Stampa la lista dei tipi di esami che possono essere prenotati

esegui_esame

```
CREATE PROCEDURE `esegui_esame` (in var_codice int, in var_username varchar(45), in var_diagnosi
varchar(256))
BEGIN
    declare var_ruolo enum('amministratore', 'cup', 'personale');
    declare exist int;
    declare var_medico varchar(16);

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

    set transaction isolation level repeatable read;
    start transaction;

        -- check if exists the reservation of this exam
        select count(*) into exist
        from `esame_prenotato`
        where `codice` = var_codice;

        if exist <> 1 then
              signal sqlstate '45000' set message_text = 'This exam does not exist or is already
executed';
        end if;

        -- check if the caller has 'personale' role
        select `ruolo` into var_ruolo
        from `utenti`
        where `username` = var_username;
```

```
        if var_ruolo <> 'personale' then
                signal sqlstate '45008' set message_text = 'You are not a personnel member';
        end if;

        -- get the doc's fiscal code
        select `cf` into var_medico
        from `personale`
        where `username` = var_username;

        -- delete from reservations and insert in executed exams
        insert into `esame_eseguito` (codice, `data`, prenotazione, paziente, tipo, medico,
diagnosi)

        select `esame_prenotato`.`codice`, `esame_prenotato`.`data`,
`esame_renotato`.`prenotazione`, `esame_prenotato`.`paziente`, `esame_prenotato`.`tipo`,
var_medico, var_diagnosi
        from `esame_prenotato`
        where `esame_prenotato`.`codice` = var_codice;

        delete from `esame_prenotato` where `codice` = var_codice;
    commit;
END
```

Converte un esame prenotato in esame eseguito.

info_ospedale

```
CREATE PROCEDURE `info_ospedale` (in var_codice int)
BEGIN
        set transaction isolation level read committed;
        start transaction;
                select * from `ospedale` where `codice` = var_codice;
                select * from `reparto` where `ospedale` = var_codice;
               select * from `laboratorio` where `ospedale` = var_codice;
        commit;
END
```

Stampa le informazioni di un ospedale, i suoi reparti e i suoi laboratori.

inserisci_laboratorio

```
CREATE PROCEDURE `inserisci_laboratorio` (in var_osp int, in var_nome varchar(45), in var_piano
int, in var_stanza int, in var_resp varchar(16))
BEGIN
    declare var_tipo enum('medico', 'volontario', 'primario');
    declare ospedale_resp int;
    declare is_resp int;
    declare new_cod int;

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

    set transaction isolation level serializable;
    start transaction;

        -- check if var_resp is not 'volontario'
        select `tipo` into var_tipo
        from `personale`
        where `cf` = var_resp;

        if var_tipo = 'volontario' then
                signal sqlstate '45000' set message_text = 'the new responsible cannot be type
"volontario"';
```

```
        end if;

        -- check if var_resp works in the same hospital of the lab
        select `ospedale` into ospedale_resp
        from `personale` join `reparto` on `personale`.`reparto` = `reparto`.`id`
        where `cf` = var_resp;

        if var_osp <> ospedale_resp then
              signal sqlstate '45000' set message_text = 'the new responsible cannot work in
another hospital';
        end if;

        -- check var_resp is not already responsable of a lab
        select count(*) into is_resp
        from `laboratorio`
        where `responsabile` = var_resp;

        if is_resp <> 0 then
              signal sqlstate '45000' set message_text = 'the new responsible is already
responsible of a lab';
        end if;

        -- find the 'codice' to assign to the new lab
        select max(codice) + 1 into new_cod
        from `laboratorio`
        where `ospedale` = var_osp;
        -- if can't find a 'codice', set 1
        if new_cod is null then
              set new_cod = 1;
        end if;

        insert into `laboratorio` values (null, new_cod, var_osp, var_nome, var_piano,
var_stanza, var_resp);
    commit;
END
```

Inserisce un nuovo laboratorio nel database.

inserisci_ospedale

```
CREATE PROCEDURE `inserisci_ospedale` (in var_nome_osp varchar(45), in var_ind_osp varchar(45) ,
in var_medico varchar(16), in var_nome_rep varchar(45), in var_tel_rep varchar(45), in
var_specializzazione varchar(45), out osp_code int)
BEGIN

    declare var_tipo enum('medico', 'primario', 'volontario');
    declare is_resp_lab int;
    declare is_resp_osp int;

    declare exit handler for sqlexception
    begin
        rollback; -- rollback any changes made in the transaction
        resignal; -- raise again the sql exception to the caller
    end;

    set transaction isolation level serializable;
    start transaction;
       -- seleziona personale di tipo medico che non sia responsabile di qualcosa
       -- crea un ospedale e assegna quel dottore come responsabile
       -- crea un nuovo reparto e assegna quel dottore come primario
       -- aggiorna il tipo e il reparto del dottore ora primario
       -- assegna una specializzazione

       -- check if is type 'medico'
       select `tipo` into var_tipo
       from `personale`
```

```
       where `cf` = var_medico;

       if var_tipo <> 'medico' then
              signal sqlstate '45000' set message_text = 'The personnel chosen is not type
medico';
       end if;

       -- check the doctor is not responsable of a hospital or a lab
       select count(*) into is_resp_lab
       from `laboratorio`
       where `responsabile` = var_medico;

       select count(*) into is_resp_osp
       from `ospedale`
       where `responsabile` = var_medico;

       if is_resp_lab + is_resp_osp > 0 then
              signal sqlstate '45000' set message_text = 'Cannot move this doctor because she/he
is already responsible somewhere else';
       end if;

       -- add the hospital
       insert into `ospedale` values (null, var_nome_osp, var_ind_osp, var_medico);

       -- add the ward
       set osp_code = last_insert_id();
       insert into `reparto` values(null, 1, last_insert_id(), var_nome_rep, var_tel_rep,
var_medico);

       -- update the doctor
       update `personale`
       set `reparto` = last_insert_id(), `tipo` = 'primario'
       where `cf` = var_medico;

       -- add a degree
       insert into `specializzazione` values (var_medico, var_specializzazione);

   commit;
END
```

Inserisce un nuovo ospedale nel database, insieme ad un reparto. Necessita in input di un medico non responsabile di laboratorio o ospedale che verrà usato come responsabile ospedale e primario del reparto.

inserisci_paziente

```
CREATE PROCEDURE `inserisci_paziente` (in num_ts int, in nome varchar(45), in cognome
varchar(45), in data_nascita date, in luogo_nascita varchar(45), in residenza varchar(45), in
recapito varchar(45))
BEGIN
    declare exit handler for sqlexception
    begin
        rollback; -- rollback any changes made in the transaction
        resignal; -- raise again the sql exception to the caller
    end;

    start transaction;
        insert into `paziente` values (num_ts, nome, cognome, data_nascita, luogo_nascita,
residenza);
        insert into `recapito` values (num_ts, recapito);
   commit;
END
```

Inserisce un paziente e un suo recapito nel database.

inserisci_personale

```
CREATE PROCEDURE `inserisci_personale` (in var_cf varchar(16), in var_nome varchar(45), in
var_cognome varchar(45), in var_domicilio varchar(45), in var_tipo varchar(45), in var_reparto
int, in var_username varchar(45))
BEGIN
    insert into `personale` values (var_cf, var_nome, var_cognome, var_domicilio, var_tipo,
var_reparto, var_username);
END
```

Inserisce un membro del personale nel database.

inserisci_recapito

```
CREATE PROCEDURE `inserisci_recapito` (IN num_ts INT, IN recapito VARCHAR(45))
BEGIN
    insert into `recapito` values(num_ts, recapito);
END
```

Inserisce un ulteriore recapito di un paziente.

inserisci_reparto

```
CREATE PROCEDURE `inserisci_reparto` (in var_osp int, in var_nome varchar(45), in var_tel
varchar(45), in var_resp varchar(16), in var_specializzazione varchar(45))
BEGIN
    declare var_osp_check int;
    declare var_tipo enum('medico', 'primario', 'volontario');
    declare var_codice_rep int;

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

    set transaction isolation level serializable;
    start transaction;
        select `reparto`.`ospedale`, `personale`.`tipo` into var_osp_check, var_tipo
        from `personale` join `reparto` on `personale`.`reparto` = `reparto`.`id`
        where `personale`.`cf` = var_resp;

        if var_osp <> var_osp_check then
              signal sqlstate '45000' set message_text = 'The doctor must work in the same
hospital of the new ward';
        end if;

        if var_tipo <> 'medico' then
              signal sqlstate '45000' set message_text = 'reparto.primario cannot be a personnel
with "volontario" type or someone already responsible of a ward';
        end if;

        -- update var_resp to 'primario' type, add a degree
        update `personale`
        set `tipo` = 'primario'
        where `cf` = var_resp;

        insert into `specializzazione` values (var_resp, var_specializzazione);

        -- find a value for 'codice' and add the ward
        select max(codice) + 1 into var_codice_rep
        from `reparto`
        where `ospedale` = var_osp;

        insert into `reparto` values (null, var_codice_rep, var_osp, var_nome, var_tel,
var_resp);
```

```
        -- move the 'primario' to the new ward
        update `personale`
        set `reparto` = last_insert_id()
        where `cf` = var_resp;
    commit;
END
```

Inserisce un nuovo reparto. Necessita in input di un medico che verrà usato come primario del reparto.

inserisci_risultati

```
CREATE PROCEDURE `inserisci_risultati` (in var_esame int, in var_parametro varchar(45), in
var_valore double, in var_misura varchar(45))
BEGIN
       insert into `parametro` values(var_esame, var_parametro, var_valore, var_misura);
END
```

Inserisce un risultato di un esame esguito.

inserisci_tipo_esame

```
CREATE PROCEDURE `inserisci_tipo_esame` (in nome varchar(45))
BEGIN
       insert into `tipo_esame` values (nome, true);
END
```

Inserisce un tipo di esame.

lista_laboratori

```
CREATE PROCEDURE `lista_laboratori` ()
BEGIN
    set transaction isolation level read committed;
    start transaction;
        select `laboratorio`.`codice` as 'Cod. Lab.', `laboratorio`.`nome` as 'Laboratorio',
`laboratorio`.`ospedale` as 'Cod. Osp.', `ospedale`.`nome` as 'Ospedale', `laboratorio`.`id` as
'ID Laboratorio'
        from `laboratorio` join `ospedale` on `laboratorio`.`ospedale` = `ospedale`.`codice`;
    commit;
END
```

Stampa la lista dei laboratori della ASL.

lista_ospedali

```
CREATE PROCEDURE `lista_ospedali` ()
BEGIN
    set transaction isolation level read committed;
    start transaction;
        select `ospedale`.`codice` as 'Cod. Ospedale', `ospedale`.`nome` as 'Nome Ospedale'
        from `ospedale`;
    commit;
END
```

Stampa la lista degli ospedali della ASL.

lista_recapiti

```
CREATE PROCEDURE `lista_recapiti` (in var_paziente int)
BEGIN
    set transaction isolation level read committed;
    start transaction;
              select `campo` as 'Recapiti' from `recapito` where `paziente` = var_paziente;
    commit;
```

```
END
```

Stampa la lista dei recapiti di un paziente.

login

```
CREATE PROCEDURE `login` (in var_username varchar(45), in var_pass varchar(45), out var_role int)
BEGIN
	declare var_user_role enum('amministratore', 'cup', 'personale');

    select `ruolo` from `utenti`
    where `username` = var_username
    and `password` = md5(var_pass)
    into var_user_role;

    -- See the corrisponding enum in the client
		if var_user_role = 'amministratore' then
			set var_role = 1;
		elseif var_user_role = 'cup' then
			set var_role = 2;
		elseif var_user_role = 'personale' then
			set var_role = 3;
		else
			set var_role = 4;
		end if;
END
```

Permette l'accesso al programma di un utente.

modifica_laboratorio

```
CREATE PROCEDURE `modifica_laboratorio` (in var_id int, in colonna int, in var_str varchar(45),
in var_int int)
BEGIN
    declare var_tipo enum('medico', 'volontario', 'primario');
    declare ospedale_resp int;
    declare ospedale_lab int;
    declare is_resp int;
    declare lab_exist int;

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

    set transaction isolation level serializable;
    start transaction;
        -- check if var_id exist
        select count(*) into lab_exist
        from `laboratorio`
        where `id` = var_id;

        if lab_exist < 1 then
			signal sqlstate '45000' set message_text = 'this laboratory does not exist';
        end if;

        if colonna = 1 then
			update `laboratorio` set `nome` = var_str where `id` = var_id;
        elseif colonna = 2 then
			update `laboratorio` set `piano` = var_int where `id` = var_id;
        elseif colonna = 3 then
			update `laboratorio` set `stanza` = var_int where `id` = var_id;
        elseif colonna = 4 then
			-- check new responsable is not 'volontario' type
			select `tipo` into var_tipo
```

```
               from `personale`
               where `cf` = var_str;

               if var_tipo = 'volontario' then
                      signal sqlstate '45000' set message_text = 'the new responsible cannot be
type "volontario"';
               end if;

               -- check if she/he works in the same hospital of the lab
               select `ospedale` into ospedale_resp
               from `personale` join `reparto` on `personale`.`reparto` = `reparto`.`id`
               where `cf` = var_str;

               select `ospedale` into ospedale_lab
               from `laboratorio`
               where `id` = var_id;

               if ospedale_lab <> ospedale_resp then
                    signal sqlstate '45000' set message_text = 'the new responsible cannot work in
another hospital';
               end if;

                -- check she/he is not already responsable of a lab
                select count(*) into is_resp
                from `laboratorio`
                where `responsabile` = var_str;

                if is_resp <> 0 then
                   signal sqlstate '45000' set message_text = 'the new responsible is already
responsible of a lab';
                end if;

                 update `laboratorio`
                 set `responsabile` = var_str
                 where `id` = var_id;
       end if;
    commit;
END
```

Modifica un attributo di un laboratorio. Nel caso della modifica di un responsabile, controlla che questo non sia già responsabile di un laboratorio, e se lavora in un reparto dell'ospedale in cui si trova il laboratorio.

modifica_ospedale

```
CREATE PROCEDURE `modifica_ospedale` (in var_codice int, in colonna int, in var_str varchar(45))
BEGIN
    declare exist int;
    declare hosp_of_new_resp int;

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

    set transaction isolation level serializable;  -- how is count(*) protected?
    start transaction;

        select count(*) into exist
        from `ospedale`
        where `codice` = var_codice;

        if exist < 1 then
```

```
            signal sqlstate '45000' set message_text = 'The hospital you are trying to update
does not exists';
        end if;

        -- check that if colonna = 3 (i.e. update responsable), the doc "pointed" works in a ward
inside the hospital
        if colonna = 3 then
            select `ospedale`.`codice` into hosp_of_new_resp
            from `personale` join `reparto` on `personale`.`reparto` = `reparto`.`id`
            join `ospedale` on `reparto`.`ospedale` = `ospedale`.`codice`
            where `personale`.`cf` = var_str;

            if var_codice <> hosp_of_new_resp then
                signal sqlstate '45001' set message_text = 'Cannot update this attr because
this doctor does not work in this hospital.';
            end if;
       end if;


       if colonna = 1 then
            update `ospedale`
            set `nome` = var_str
            where `codice` = var_codice;
       elseif colonna = 2 then
            update `ospedale`
             set `indirizzo` = var_str
             where `codice` = var_codice;
       elseif colonna = 3 then
            update `ospedale`
            set `responsabile` = var_str
            where `codice` = var_codice;
       else
            signal sqlstate '45002' set message_text = 'The second parameter must be between 1
and 3 both included';
       end if;
    commit;
END
```

Modifica un attributo di un ospedale. Nel caso del responsabile controlla se il nuovo responsabile lavora nell'ospedale che si sta modificando.

modifica_paziente

```
CREATE PROCEDURE `modifica_paziente` (in var_num_ts int, in colonna int, in var_new_num_ts int,
in var_str varchar(45), in var_date date)
BEGIN
    declare exist int;

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

    set transaction isolation level repeatable read;
    start transaction;

        select count(*) into exist
        from `paziente`
        where `num_ts` = var_num_ts;

        if exist < 1 then
            signal sqlstate '45004' set message_text = 'The patient you are trying to update
does not exists';
        end if;
```

```
        if colonna = 1 then
               update `paziente`
               set `num_ts` = var_new_num_ts
               where `num_ts` = var_num_ts;
       elseif colonna = 2 then
               update `paziente`
               set `nome` = var_str
               where `num_ts` = var_num_ts;
       elseif colonna = 3 then
               update `paziente`
               set `cognome` = var_str
               where `num_ts` = var_num_ts;
       elseif colonna = 4 then
               update `paziente`
               set `data_nascita` = var_date
               where `num_ts` = var_num_ts;
       elseif colonna = 5 then
               update `paziente`
               set `luogo_nascita` = var_str
               where `num_ts` = var_num_ts;
       elseif colonna = 6 then
               update `paziente`
               set `residenza` = var_str
               where `num_ts` = var_num_ts;
       else
               signal sqlstate '45005' set message_text = 'The second parameter must be between 1
and 6 both included';
       end if;
    commit;
END
```

Modifica un attributo di un paziente.

modifica_personale

```
REATE PROCEDURE `modifica_personale` (in var_cf varchar(16), in colonna int, in var_str
varchar(45), in var_reparto int)
BEGIN
    declare exist int;
    declare is_resp_lab int;
    declare is_resp_osp int;
    declare is_primario int;

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

    set transaction isolation level serializable;  -- is count(*) protected?
    start transaction;

        select count(*) into exist
        from `personale`
        where `cf` = var_cf;

        if exist < 1 then
               signal sqlstate '45000' set message_text = 'The personnel member you are trying to
update does not exists';
       end if;

       -- check that not(colonna=5 or 6 AND not responsable of a lab/ward/hospital)
       select count(*) into is_resp_lab
       from `laboratorio`
       where `responsabile` = var_cf;
```

```
	select count(*) into is_resp_osp
	from `ospedale`
	where `responsabile` = var_cf;

	select count(*) into is_primario
	from `reparto`
	where `primario` = var_cf;

	if (is_resp_lab + is_resp_osp + is_primario > 0) AND (colonna = 5 or colonna = 6) then
		signal sqlstate '45001' set message_text = 'Cannot update this attr because she/he
is responsible for something. Move this personnel member first.';
	end if;

	if colonna = 1 then
		update `personale`
		set `cf` = var_str
		where `cf` = var_cf;
	elseif colonna = 2 then
		update `personale`
		 set `nome` = var_str
		 where `cf` = var_cf;
	elseif colonna = 3 then
		update `personale`
		set `cognome` = var_str
		where `cf` = var_cf;
	elseif colonna = 4 then
		update `personale`
		set `domicilio` = var_str
		where `cf` = var_cf;
	elseif colonna = 5 then
		update `personale`
		set `tipo` = var_str
		where `cf` = var_cf;
	elseif colonna = 6 then
		update `personale`
		set `reparto` = var_reparto
		where `cf` = var_cf;
	elseif colonna = 7 then
		update `personale`
		set `username` = var_str
		where `cf` = var_cf;
	else
		signal sqlstate '45002' set message_text = 'The second parameter must be between 1
and 7 both included';
	end if;
    commit;
END
```

Modifica un attributo di un membro del personale.

modifica_reparto

```
CREATE PROCEDURE `modifica_reparto` (in var_id int, in colonna int, in var_str varchar(45), in
var_specializzazione varchar(45))
BEGIN
    declare exist int;
    declare var_tipo enum('medico', 'primario', 'volontario');
    declare var_osp int;
    declare osp_new_resp int;
    declare old_primario varchar(16);

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

```
    end;

    set transaction isolation level serializable;
    start transaction;

        select count(*) into exist
        from `reparto`
        where `id` = var_id;

        if exist <> 1 then
              signal sqlstate '45000' set message_text = 'this ward does not exist';
        end if;

        if colonna = 1 then
              update `reparto` set `nome` = var_str where `id` = var_id;
        elseif colonna = 2 then
              update `reparto` set `telefono` = var_str where `id` = var_id;
        elseif colonna = 3 then

            select `ospedale`, `tipo` into osp_new_resp, var_tipo
            from `personale` join `reparto` on `personale`.`reparto` = `reparto`.`id`
            where `personale`.`cf` = var_str;

            select `ospedale` into var_osp
            from `reparto`
            where `id` = var_id;

            if var_tipo <> 'medico' then
                signal sqlstate '45000' set message_text = 'the new ward must be "medico" type';
            end if;

            if var_osp <> osp_new_resp then
                signal sqlstate '45000' set message_text = 'the new ward responsible works in a
different hospital';
            end if;

            -- update new ward responsable type and assigned ward, and add a degree
            update `personale` set `tipo` = 'primario', `reparto` = var_id where `cf` = var_str;
            insert into `specializzazione` values (var_str, var_specializzazione);

            -- remember the old ward responsable
            select `primario` into old_primario
            from `reparto`
            where `id` = var_id;

            -- update the responsable attribute in 'reparto'
            update `reparto` set `primario` = var_str where `id` = var_id;

            -- update the old ward responsable type and delete her/his degrees
            delete from `specializzazione` where `primario` = old_primario;
            update `personale` set `tipo` = 'medico' where `cf` = old_primario;

        end if;
    commit;
END
```

Modifica un attributo di un reparto. Nel caso del primario necessita di un medico all'interno dell'ospdale del reparto che verrà promosso a primario e aggiunta una specializzazione, mentre il vecchio primario verrà retrocesso a semplice medico e le sue specializzazioni saranno eliminate.

prenota_esame

```
CREATE PROCEDURE `prenota_esame` (in var_data date, in var_ora time, in var_costo float, in
var_urgente boolean, inout var_prenotazione int, in var_paziente int, in var_tipo varchar(45), in
```

```
var_id_lab int)
BEGIN
    declare is_available int;

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

    set transaction isolation level serializable;
    start transaction;
        select `disponibile` into is_available
        from `tipo_esame`
        where `nome` = var_tipo;

        if is_available = 0 then
              signal sqlstate '45000' set message_text = 'This type of exam is not avialable';
       end if;

       insert into `esame_prenotato` values (null, var_data, var_ora, var_costo, var_urgente,
null, var_paziente, var_tipo, var_id_lab);

       -- this is a work-aroud to have esame_prenotato.prenotazione like a second auto-increment
attribute using the AI PK esame_prenotato.codice

       if var_prenotazione is null then -- the only exam in a reservation OR the first of
multiple in the same reservation
              update `esame_prenotato`
               set `prenotazione` = LAST_INSERT_ID()
               where `codice` = LAST_INSERT_ID();
               set var_prenotazione = LAST_INSERT_ID(); -- return the reservation number given
       else -- if va_prenotazione is not null => reserving an exam "inside" a reservation with
multiple exams with reservation number = var_prenotazione
              update `esame_prenotato`
               set `prenotazione` = var_prenotazione
               where `codice` = LAST_INSERT_ID();
       end if;
    commit;
END
```

Permette di prenotare un esame per un paziente. Per avere l'attributo "prenotazione" di esame_prenotato come se fosse univoco sulla tabella e AutoIncrement sfrutto il valore di "codice" che è chiave e AI per esame_prenotato. Accetta il valore di "prenotazione" in input per permettere la prenotazione di più esami nella stessa prenotazione.

report_esami_eseguiti

```
CREATE PROCEDURE `report_esami_eseguiti` (in var_cf varchar(16), in opzione int)
BEGIN
    -- opzione = 0 => monthly report; opzione = 1 => annual report
    declare var_time_length date;

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

    set transaction isolation level read committed;
    start transaction;

              if opzione = 0 then
```

```
                        select curdate() - interval 30 day into var_time_length;
                elseif opzione = 1 then
                        select curdate() - interval 365 day into var_time_length;
                else
                        signal sqlstate '45000' set message_text = 'Second parameter must be 0 or
1';
                end if;

                select `codice`, `tipo`, `data`, `paziente`.`num_ts` as 'TS Paziente', `diagnosi`
                 from `esame_eseguito` join `paziente` on `esame_eseguito`.`paziente` =
`paziente`.`num_ts`
                 where `medico` = var_cf and `data` > var_time_length;
    commit;
END
```

Mostra tutti gli esami che un membro del personale ha eseguito nell'ultimo mese o anno.

report_prenotazione

```
CREATE PROCEDURE `report_prenotazione` (in var_prenotazione int)
BEGIN
    declare var_esame int;

    declare done int default false;
    declare cur cursor for select `codice` from `esame_eseguito` where `prenotazione` =
var_prenotazione order by `codice` desc;
    declare continue handler for not found set done = true;

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

    set transaction isolation level serializable;
    start transaction;

        -- 1) esami prenotati
                -- the exams reserved and not executed yet with reservation number =
var_prenotazione
                select `tipo`, `paziente`.`nome`, `paziente`.`cognome`, `data`, `ora`,
`laboratorio`.`nome`, `ospedale`.`nome`, `urgente`, `costo`
                from `esame_prenotato` join `paziente` on `esame_prenotato`.`paziente` =
`paziente`.`num_ts`
        join `laboratorio` on `esame_prenotato`.`laboratorio` = `laboratorio`.`id`
        join `ospedale` on `laboratorio`.`ospedale` = `ospedale`.`codice`
                where `prenotazione` = var_prenotazione;

        -- 2) loop esami eseguiti to print their results
                        -- a) get a table   esame|data|medico|diagnosi   for this patient
        select `esame_eseguito`.`tipo`, `data`, `personale`.`nome`, `personale`.`cognome`,
`diagnosi`, `paziente`.`nome`, `paziente`.`cognome`
        from `esame_eseguito` join `paziente` on `esame_eseguito`.`paziente` =
`paziente`.`num_ts`
        join `personale` on `esame_eseguito`.`medico` = `personale`.`cf`
        where `prenotazione` = var_prenotazione
        order by `codice` desc;

                        -- b) for each esame eseguito print the relative table   parametri|valori
        open cur;
        read_loop: loop

            fetch cur into var_esame;
            if done then
                                leave read_loop;
                        end if;
```

```
            select `nome` , `valore`, `misura`
            from `parametro`
            where `esame` = var_esame;

               end loop;
        close cur;

       commit;
END
```

Mostra gli esami prenotati ancora da eseguire e gli esami già eseguiti all'interno della stessa prenotazione.

report_storico_paz

```
CREATE PROCEDURE `report_storico_paz` (in var_num_ts int)
BEGIN
    declare var_esame int;

    declare done int default false;
    declare cur cursor for select `codice` from `esame_eseguito` where `paziente` = var_num_ts
order by `codice` desc;
    declare continue handler for not found set done = true;

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

    set transaction isolation level serializable;
    start transaction;

        -- 1) get a table   esame|data|medico|diagnosi   for this patient
        select `esame_eseguito`.`tipo`, `data`, `personale`.`nome`, `personale`.`cognome`,
`diagnosi`
        from `esame_eseguito` join `personale` on `esame_eseguito`.`medico` = `personale`.`cf`
        where `paziente` = var_num_ts
        order by `codice` desc;

        -- 2) for each esame eseguito print the relative table   parametri|valori
        open cur;
        read_loop: loop

            fetch cur into var_esame;
            if done then
                        leave read_loop;
                  end if;

            select `nome` , `valore`, `misura`
            from `parametro`
            where `esame` = var_esame;

               end loop;
        close cur;

       commit;
END
```

Mostra tutti gli esami che un paziente ha eseguito nella sua vita.

stato_tipo_esame

```
CREATE PROCEDURE `stato_tipo_esame` (in var_tipo_esame varchar(45), in disponibile boolean)
```

```
BEGIN
	declare exist int;

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

	set transaction isolation level read committed;
	start transaction;

	 select count(*) into exist
	 from `tipo_esame`
	 where `nome` = var_tipo_esame;

	 if exist <> 1 then
			signal sqlstate '45000' set message_text = 'This type of exam does not
exists';
		end if;

		update `tipo_esame`
		set `disponibile` = disponibile
		where `nome` = var_tipo_esame;
	commit;
END
```

Cambia lo stato di un tipo di esame.

Appendice: Implementazione

Codice SQL per instanziare il database

Viene riportato il file .sql generato per l'istanziazione del database. Tutte le righe di codice SQL riportate in sezioni precedenti sono state omesse dal codice qui sotto.

```
-- MySQL Workbench Forward Engineering

SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0;
SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0;
SET @OLD_SQL_MODE=@@SQL_MODE, SQL_MODE='TRADITIONAL,ALLOW_INVALID_DATES';

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

-- -----------------------------------------------------
-- Schema gestione_asl
-- -----------------------------------------------------
CREATE SCHEMA IF NOT EXISTS `gestione_asl` DEFAULT CHARACTER SET utf8 ;
USE `gestione_asl` ;

-- -----------------------------------------------------
-- Table `gestione_asl`.`paziente`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `gestione_asl`.`paziente` ;

CREATE TABLE IF NOT EXISTS `gestione_asl`.`paziente` (
  `num_ts` INT NOT NULL,
  `nome` VARCHAR(45) NOT NULL,
  `cognome` VARCHAR(45) NOT NULL,
  `data_nascita` DATE NOT NULL,
  `luogo_nascita` VARCHAR(45) NOT NULL,
  `residenza` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`num_ts`))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `gestione_asl`.`recapito`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `gestione_asl`.`recapito` ;

CREATE TABLE IF NOT EXISTS `gestione_asl`.`recapito` (
  `paziente` INT NOT NULL,
  `campo` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`paziente`, `campo`),
  CONSTRAINT `Paziente`
    FOREIGN KEY (`paziente`)
    REFERENCES `gestione_asl`.`paziente` (`num_ts`)
    ON DELETE CASCADE
    ON UPDATE CASCADE)
ENGINE = InnoDB
COMMENT = '           ';


-- -----------------------------------------------------
-- Table `gestione_asl`.`tipo_esame`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `gestione_asl`.`tipo_esame` ;
```

```
CREATE TABLE IF NOT EXISTS `gestione_asl`.`tipo_esame` (
  `nome` VARCHAR(45) NOT NULL,
  `disponibile` TINYINT NOT NULL,
  PRIMARY KEY (`nome`))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `gestione_asl`.`esame_eseguito`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `gestione_asl`.`esame_eseguito` ;

CREATE TABLE IF NOT EXISTS `gestione_asl`.`esame_eseguito` (
  `codice` INT NOT NULL,
  `data` DATE NOT NULL,
  `prenotazione` INT NOT NULL,
  `paziente` INT NOT NULL,
  `tipo` VARCHAR(45) NOT NULL,
  `medico` VARCHAR(16) NOT NULL,
  `diagnosi` VARCHAR(256) NULL,
  PRIMARY KEY (`codice`),
  CONSTRAINT `fk_esame_eseguito_paz`
    FOREIGN KEY (`paziente`)
    REFERENCES `gestione_asl`.`paziente` (`num_ts`)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  CONSTRAINT `fk_esame_eseguito_tipo`
    FOREIGN KEY (`tipo`)
    REFERENCES `gestione_asl`.`tipo_esame` (`nome`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;

CREATE INDEX `fk_esame_eseguito_paz_idx` ON `gestione_asl`.`esame_eseguito` (`paziente` ASC);

CREATE INDEX `fk_esame_eseguito_tipo_idx` ON `gestione_asl`.`esame_eseguito` (`tipo` ASC);


-- -----------------------------------------------------
-- Table `gestione_asl`.`reparto`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `gestione_asl`.`reparto` ;

CREATE TABLE IF NOT EXISTS `gestione_asl`.`reparto` (
  `id` INT NOT NULL AUTO_INCREMENT,
  `codice` INT NOT NULL,
  `ospedale` INT NOT NULL,
  `nome` VARCHAR(45) NOT NULL,
  `telefono` VARCHAR(45) NOT NULL,
  `primario` VARCHAR(16) NOT NULL,
  PRIMARY KEY (`id`),
  CONSTRAINT `fk_reparto_osp`
    FOREIGN KEY (`ospedale`)
    REFERENCES `gestione_asl`.`ospedale` (`codice`)
    ON DELETE CASCADE
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_reparto_primario`
    FOREIGN KEY (`primario`)
    REFERENCES `gestione_asl`.`personale` (`cf`)
    ON DELETE NO ACTION
    ON UPDATE CASCADE)
ENGINE = InnoDB;

CREATE INDEX `fk_reparto_osp_idx` ON `gestione_asl`.`reparto` (`ospedale` ASC);
```

```
CREATE INDEX `fk_reparto_primario_idx` ON `gestione_asl`.`reparto` (`primario` ASC);

CREATE UNIQUE INDEX `uq_rep_osp` ON `gestione_asl`.`reparto` (`codice` ASC, `ospedale` ASC);


-- -----------------------------------------------------
-- Table `gestione_asl`.`utenti`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `gestione_asl`.`utenti` ;

CREATE TABLE IF NOT EXISTS `gestione_asl`.`utenti` (
  `username` VARCHAR(45) NOT NULL,
  `password` CHAR(32) NOT NULL,
  `ruolo` ENUM('amministratore', 'cup', 'personale') NOT NULL,
  PRIMARY KEY (`username`))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `gestione_asl`.`personale`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `gestione_asl`.`personale` ;

CREATE TABLE IF NOT EXISTS `gestione_asl`.`personale` (
  `cf` VARCHAR(16) NOT NULL,
  `nome` VARCHAR(45) NOT NULL,
  `cognome` VARCHAR(45) NOT NULL,
  `domicilio` VARCHAR(45) NOT NULL,
  `tipo` ENUM('medico', 'primario', 'volontario') NOT NULL,
  `reparto` INT NOT NULL,
  `username` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`cf`),
  CONSTRAINT `fk_personale_reparto`
    FOREIGN KEY (`reparto`)
    REFERENCES `gestione_asl`.`reparto` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_personale_username`
    FOREIGN KEY (`username`)
    REFERENCES `gestione_asl`.`utenti` (`username`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;

CREATE INDEX `fk_personale_reparto_idx` ON `gestione_asl`.`personale` (`reparto` ASC);

CREATE INDEX `fk_personale_username_idx` ON `gestione_asl`.`personale` (`username` ASC);


-- -----------------------------------------------------
-- Table `gestione_asl`.`ospedale`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `gestione_asl`.`ospedale` ;

CREATE TABLE IF NOT EXISTS `gestione_asl`.`ospedale` (
  `codice` INT NOT NULL AUTO_INCREMENT,
  `nome` VARCHAR(45) NOT NULL,
  `indirizzo` VARCHAR(45) NOT NULL,
  `responsabile` VARCHAR(16) NOT NULL,
  PRIMARY KEY (`codice`),
  CONSTRAINT `fk_ospedale_resp`
    FOREIGN KEY (`responsabile`)
    REFERENCES `gestione_asl`.`personale` (`cf`)
    ON DELETE NO ACTION
    ON UPDATE CASCADE)
ENGINE = InnoDB;
```

```
CREATE INDEX `fk_ospedale_resp_idx` ON `gestione_asl`.`ospedale` (`responsabile` ASC);


-- -----------------------------------------------------
-- Table `gestione_asl`.`laboratorio`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `gestione_asl`.`laboratorio` ;

CREATE TABLE IF NOT EXISTS `gestione_asl`.`laboratorio` (
  `id` INT NOT NULL AUTO_INCREMENT,
  `codice` INT NOT NULL,
  `ospedale` INT NOT NULL,
  `nome` VARCHAR(45) NOT NULL,
  `piano` INT NOT NULL,
  `stanza` INT NOT NULL,
  `responsabile` VARCHAR(16) NOT NULL,
  PRIMARY KEY (`id`),
  CONSTRAINT `fk_laboratorio_osp`
    FOREIGN KEY (`ospedale`)
    REFERENCES `gestione_asl`.`ospedale` (`codice`)
    ON DELETE CASCADE
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_laboratorio_resp`
    FOREIGN KEY (`responsabile`)
    REFERENCES `gestione_asl`.`personale` (`cf`)
    ON DELETE NO ACTION
    ON UPDATE CASCADE)
ENGINE = InnoDB;

CREATE INDEX `fk_laboratorio_osp_idx` ON `gestione_asl`.`laboratorio` (`ospedale` ASC);

CREATE INDEX `fk_laboratorio_resp_idx` ON `gestione_asl`.`laboratorio` (`responsabile` ASC);

CREATE UNIQUE INDEX `uq_lab_osp` ON `gestione_asl`.`laboratorio` (`codice` ASC, `ospedale` ASC);


-- -----------------------------------------------------
-- Table `gestione_asl`.`esame_prenotato`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `gestione_asl`.`esame_prenotato` ;

CREATE TABLE IF NOT EXISTS `gestione_asl`.`esame_prenotato` (
  `codice` INT NOT NULL AUTO_INCREMENT,
  `data` DATE NOT NULL,
  `ora` TIME NOT NULL,
  `costo` FLOAT NOT NULL,
  `urgente` TINYINT NOT NULL,
  `prenotazione` INT NULL,
  `paziente` INT NOT NULL,
  `tipo` VARCHAR(45) NOT NULL,
  `laboratorio` INT NOT NULL,
  PRIMARY KEY (`codice`),
  CONSTRAINT `fk_esame_prenotato_paz`
    FOREIGN KEY (`paziente`)
    REFERENCES `gestione_asl`.`paziente` (`num_ts`)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  CONSTRAINT `fk_esame_prenotato_tipo`
    FOREIGN KEY (`tipo`)
    REFERENCES `gestione_asl`.`tipo_esame` (`nome`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_esame_prenotato_lab`
    FOREIGN KEY (`laboratorio`)
    REFERENCES `gestione_asl`.`laboratorio` (`id`)
```

```
    ON DELETE CASCADE
    ON UPDATE NO ACTION)
ENGINE = InnoDB;

CREATE INDEX `fk_esame_prenotato_paz_idx` ON `gestione_asl`.`esame_prenotato` (`paziente` ASC);

CREATE INDEX `fk_esame_prenotato_tipo_idx` ON `gestione_asl`.`esame_prenotato` (`tipo` ASC);

CREATE INDEX `fk_esame_prenotato_lab_idx` ON `gestione_asl`.`esame_prenotato` (`laboratorio`
ASC);

CREATE UNIQUE INDEX `uq_paziente_tipo_data` ON `gestione_asl`.`esame_prenotato` (`paziente` ASC,
`tipo` ASC, `data` ASC)  COMMENT 'RA1 un paziente non puo prenotare lo stesso tipo di esame nella
stessa data';


-- -----------------------------------------------------
-- Table `gestione_asl`.`parametro`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `gestione_asl`.`parametro` ;

CREATE TABLE IF NOT EXISTS `gestione_asl`.`parametro` (
  `esame` INT NOT NULL,
  `nome` VARCHAR(45) NOT NULL,
  `valore` DOUBLE NOT NULL,
  `misura` VARCHAR(45) NULL,
  PRIMARY KEY (`esame`, `nome`),
  CONSTRAINT `fk_param_esame`
    FOREIGN KEY (`esame`)
    REFERENCES `gestione_asl`.`esame_eseguito` (`codice`)
    ON DELETE CASCADE
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `gestione_asl`.`specializzazione`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `gestione_asl`.`specializzazione` ;

CREATE TABLE IF NOT EXISTS `gestione_asl`.`specializzazione` (
  `primario` VARCHAR(16) NOT NULL,
  `disciplina` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`primario`, `disciplina`),
  CONSTRAINT `fk_specializzazione_personale`
    FOREIGN KEY (`primario`)
    REFERENCES `gestione_asl`.`personale` (`cf`)
    ON DELETE CASCADE
    ON UPDATE CASCADE)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `gestione_asl`.`associazione`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `gestione_asl`.`associazione` ;

CREATE TABLE IF NOT EXISTS `gestione_asl`.`associazione` (
  `volontario` VARCHAR(16) NOT NULL,
  `nome` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`volontario`),
  CONSTRAINT `fk_associazione_vol`
    FOREIGN KEY (`volontario`)
    REFERENCES `gestione_asl`.`personale` (`cf`)
    ON DELETE CASCADE
    ON UPDATE CASCADE)
```

```
ENGINE = InnoDB;

USE `gestione_asl` ;


DELIMITER ;
USE `gestione_asl`;


DELIMITER ;
SET SQL_MODE = '';
GRANT USAGE ON *.* TO cup;
 DROP USER cup;
SET SQL_MODE='TRADITIONAL,ALLOW_INVALID_DATES';
CREATE USER 'cup' IDENTIFIED BY 'cup';

GRANT EXECUTE ON procedure `gestione_asl`.`inserisci_recapito` TO 'cup';
GRANT EXECUTE ON procedure `gestione_asl`.`cancella_paziente` TO 'cup';
GRANT EXECUTE ON procedure `gestione_asl`.`esami_disponibili` TO 'cup';
GRANT EXECUTE ON procedure `gestione_asl`.`prenota_esame` TO 'cup';
GRANT EXECUTE ON procedure `gestione_asl`.`lista_laboratori` TO 'cup';
GRANT EXECUTE ON procedure `gestione_asl`.`inserisci_paziente` TO 'cup';
GRANT EXECUTE ON procedure `gestione_asl`.`anagrafica_paziente` TO 'cup';
GRANT EXECUTE ON procedure `gestione_asl`.`lista_recapiti` TO 'cup';
GRANT EXECUTE ON procedure `gestione_asl`.`modifica_paziente` TO 'cup';
GRANT EXECUTE ON procedure `gestione_asl`.`cancella_recapito` TO 'cup';
GRANT EXECUTE ON procedure `gestione_asl`.`report_prenotazione` TO 'cup';
GRANT EXECUTE ON procedure `gestione_asl`.`report_storico_paz` TO 'cup';
SET SQL_MODE = '';
GRANT USAGE ON *.* TO login;
 DROP USER login;
SET SQL_MODE='TRADITIONAL,ALLOW_INVALID_DATES';
CREATE USER 'login' IDENTIFIED BY 'login';

GRANT EXECUTE ON procedure `gestione_asl`.`login` TO 'login';
SET SQL_MODE = '';
GRANT USAGE ON *.* TO amministratore;
 DROP USER amministratore;
SET SQL_MODE='TRADITIONAL,ALLOW_INVALID_DATES';
CREATE USER 'amministratore' IDENTIFIED BY 'amministratore';

GRANT EXECUTE ON procedure `gestione_asl`.`crea_utente` TO 'amministratore';
GRANT EXECUTE ON procedure `gestione_asl`.`inserisci_tipo_esame` TO 'amministratore';
GRANT EXECUTE ON procedure `gestione_asl`.`stato_tipo_esame` TO 'amministratore';
GRANT EXECUTE ON procedure `gestione_asl`.`inserisci_ospedale` TO 'amministratore';
GRANT EXECUTE ON procedure `gestione_asl`.`inserisci_personale` TO 'amministratore';
GRANT EXECUTE ON procedure `gestione_asl`.`anagrafica_personale` TO 'amministratore';
GRANT EXECUTE ON procedure `gestione_asl`.`modifica_personale` TO 'amministratore';
GRANT EXECUTE ON procedure `gestione_asl`.`cancella_personale` TO 'amministratore';
GRANT EXECUTE ON procedure `gestione_asl`.`report_esami_eseguiti` TO 'amministratore';
GRANT EXECUTE ON procedure `gestione_asl`.`info_ospedale` TO 'amministratore';
GRANT EXECUTE ON procedure `gestione_asl`.`lista_ospedali` TO 'amministratore';
GRANT EXECUTE ON procedure `gestione_asl`.`modifica_ospedale` TO 'amministratore';
GRANT EXECUTE ON procedure `gestione_asl`.`cancella_ospedale` TO 'amministratore';
GRANT EXECUTE ON procedure `gestione_asl`.`aggiorna_associazione_vol` TO 'amministratore';
GRANT EXECUTE ON procedure `gestione_asl`.`aggiungi_specializzazione` TO 'amministratore';
GRANT EXECUTE ON procedure `gestione_asl`.`inserisci_reparto` TO 'amministratore';
GRANT EXECUTE ON procedure `gestione_asl`.`modifica_reparto` TO 'amministratore';
GRANT EXECUTE ON procedure `gestione_asl`.`cancella_reparto` TO 'amministratore';
GRANT EXECUTE ON procedure `gestione_asl`.`inserisci_laboratorio` TO 'amministratore';
GRANT EXECUTE ON procedure `gestione_asl`.`modifica_laboratorio` TO 'amministratore';
GRANT EXECUTE ON procedure `gestione_asl`.`cancella_laboratorio` TO 'amministratore';
SET SQL_MODE = '';
GRANT USAGE ON *.* TO personale;
 DROP USER personale;
SET SQL_MODE='TRADITIONAL,ALLOW_INVALID_DATES';
```

```
CREATE USER 'personale' IDENTIFIED BY 'personale';

GRANT EXECUTE ON procedure `gestione_asl`.`aggiorna_diagnosi` TO 'personale';
GRANT EXECUTE ON procedure `gestione_asl`.`esegui_esame` TO 'personale';
GRANT EXECUTE ON procedure `gestione_asl`.`inserisci_risultati` TO 'personale';

-- -----------------------------------------------------
-- Data for table `gestione_asl`.`paziente`
-- -----------------------------------------------------
START TRANSACTION;
USE `gestione_asl`;
INSERT INTO `gestione_asl`.`paziente` (`num_ts`, `nome`, `cognome`, `data_nascita`,
`luogo_nascita`, `residenza`) VALUES (1, 'mario', 'rossi', '2000-01-01', 'roma', 'roma');
INSERT INTO `gestione_asl`.`paziente` (`num_ts`, `nome`, `cognome`, `data_nascita`,
`luogo_nascita`, `residenza`) VALUES (2, 'Giuseppe', 'Verdi', '2002-02-02', 'Le Roncole', 'Via
Giuseppe Verdi 1, Milano');
INSERT INTO `gestione_asl`.`paziente` (`num_ts`, `nome`, `cognome`, `data_nascita`,
`luogo_nascita`, `residenza`) VALUES (10, 'Francesco', 'Totti', '1976-09-27', 'Roma', 'Roma');

COMMIT;


-- -----------------------------------------------------
-- Data for table `gestione_asl`.`recapito`
-- -----------------------------------------------------
START TRANSACTION;
USE `gestione_asl`;
INSERT INTO `gestione_asl`.`recapito` (`paziente`, `campo`) VALUES (1, '555555555');
INSERT INTO `gestione_asl`.`recapito` (`paziente`, `campo`) VALUES (1, '555000555');
INSERT INTO `gestione_asl`.`recapito` (`paziente`, `campo`) VALUES (2, '222222222');
INSERT INTO `gestione_asl`.`recapito` (`paziente`, `campo`) VALUES (10, '333101010');

COMMIT;


-- -----------------------------------------------------
-- Data for table `gestione_asl`.`tipo_esame`
-- -----------------------------------------------------
START TRANSACTION;
USE `gestione_asl`;
INSERT INTO `gestione_asl`.`tipo_esame` (`nome`, `disponibile`) VALUES ('Ecografia', 1);
INSERT INTO `gestione_asl`.`tipo_esame` (`nome`, `disponibile`) VALUES ('Analisi del sangue', 1);
INSERT INTO `gestione_asl`.`tipo_esame` (`nome`, `disponibile`) VALUES ('TAC', 1);
INSERT INTO `gestione_asl`.`tipo_esame` (`nome`, `disponibile`) VALUES ('ECG', 1);
INSERT INTO `gestione_asl`.`tipo_esame` (`nome`, `disponibile`) VALUES ('ecocolordoppler', 0);
INSERT INTO `gestione_asl`.`tipo_esame` (`nome`, `disponibile`) VALUES ('PCR covid19', 1);

COMMIT;


-- -----------------------------------------------------
-- Data for table `gestione_asl`.`esame_eseguito`
-- -----------------------------------------------------
START TRANSACTION;
USE `gestione_asl`;
INSERT INTO `gestione_asl`.`esame_eseguito` (`codice`, `data`, `prenotazione`, `paziente`,
`tipo`, `medico`, `diagnosi`) VALUES (2, '2021-02-03', 2, 2, 'Analisi del sangue', 'fcanhn',
NULL);
INSERT INTO `gestione_asl`.`esame_eseguito` (`codice`, `data`, `prenotazione`, `paziente`,
`tipo`, `medico`, `diagnosi`) VALUES (3, '2021-02-02', 2, 2, 'PCR covid19', 'fcanhn', 'il
paziente risulta positivo e deve isolarsi in quarantena fino a doppio tampone negativo');

COMMIT;


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

```
-- Data for table `gestione_asl`.`reparto`
-- -----------------------------------------------------
START TRANSACTION;
USE `gestione_asl`;
INSERT INTO `gestione_asl`.`reparto` (`id`, `codice`, `ospedale`, `nome`, `telefono`, `primario`)
VALUES (1, 1, 1, 'Virologia', '0612345', 'fcanhn');
INSERT INTO `gestione_asl`.`reparto` (`id`, `codice`, `ospedale`, `nome`, `telefono`, `primario`)
VALUES (2, 1, 2, 'Ematologia', '061122', 'klsbbo');
INSERT INTO `gestione_asl`.`reparto` (`id`, `codice`, `ospedale`, `nome`, `telefono`, `primario`)
VALUES (3, 2, 2, 'Pronto Soccorso', '06118', 'cxoprc');

COMMIT;


-- -----------------------------------------------------
-- Data for table `gestione_asl`.`utenti`
-- -----------------------------------------------------
START TRANSACTION;
USE `gestione_asl`;
INSERT INTO `gestione_asl`.`utenti` (`username`, `password`, `ruolo`) VALUES ('cup',
'0c88028bf3aa6a6a143ed846f2be1ea4', 'cup');
INSERT INTO `gestione_asl`.`utenti` (`username`, `password`, `ruolo`) VALUES ('admin',
'0c88028bf3aa6a6a143ed846f2be1ea4', 'amministratore');
INSERT INTO `gestione_asl`.`utenti` (`username`, `password`, `ruolo`) VALUES ('fauci',
'0c88028bf3aa6a6a143ed846f2be1ea4', 'personale');
INSERT INTO `gestione_asl`.`utenti` (`username`, `password`, `ruolo`) VALUES ('kelso',
'0c88028bf3aa6a6a143ed846f2be1ea4', 'personale');
INSERT INTO `gestione_asl`.`utenti` (`username`, `password`, `ruolo`) VALUES ('cox',
'0c88028bf3aa6a6a143ed846f2be1ea4', 'personale');
INSERT INTO `gestione_asl`.`utenti` (`username`, `password`, `ruolo`) VALUES ('volontario',
'0c88028bf3aa6a6a143ed846f2be1ea4', 'personale');
INSERT INTO `gestione_asl`.`utenti` (`username`, `password`, `ruolo`) VALUES ('dottore',
'0c88028bf3aa6a6a143ed846f2be1ea4', 'personale');

COMMIT;


-- -----------------------------------------------------
-- Data for table `gestione_asl`.`personale`
-- -----------------------------------------------------
START TRANSACTION;
USE `gestione_asl`;
INSERT INTO `gestione_asl`.`personale` (`cf`, `nome`, `cognome`, `domicilio`, `tipo`, `reparto`,
`username`) VALUES ('fcanhn', 'Anthony', 'Fauci', 'USA', 'primario', 1, 'fauci');
INSERT INTO `gestione_asl`.`personale` (`cf`, `nome`, `cognome`, `domicilio`, `tipo`, `reparto`,
`username`) VALUES ('klsbbo', 'Bob', 'Kelso', 'USA', 'primario', 2, 'kelso');
INSERT INTO `gestione_asl`.`personale` (`cf`, `nome`, `cognome`, `domicilio`, `tipo`, `reparto`,
`username`) VALUES ('cxoprc', 'Perry', 'Cox', 'USA', 'primario', 3, 'cox');
INSERT INTO `gestione_asl`.`personale` (`cf`, `nome`, `cognome`, `domicilio`, `tipo`, `reparto`,
`username`) VALUES ('vlnsnt', 'Santino', 'Volontario', 'Roma', 'volontario', 1, 'volontario');
INSERT INTO `gestione_asl`.`personale` (`cf`, `nome`, `cognome`, `domicilio`, `tipo`, `reparto`,
`username`) VALUES ('dottore', 'dottore', 'dottore', 'via dei medici, 1', 'medico', 1,
'dottore');

COMMIT;


-- -----------------------------------------------------
-- Data for table `gestione_asl`.`ospedale`
-- -----------------------------------------------------
START TRANSACTION;
USE `gestione_asl`;
INSERT INTO `gestione_asl`.`ospedale` (`codice`, `nome`, `indirizzo`, `responsabile`) VALUES (1,
'San Giovanni', 'via dell amba aradam', 'fcanhn');
INSERT INTO `gestione_asl`.`ospedale` (`codice`, `nome`, `indirizzo`, `responsabile`) VALUES (2,
'Tor Vergata', 'viale Oxford', 'klsbbo');
```

```
COMMIT;


-- -----------------------------------------------------
-- Data for table `gestione_asl`.`laboratorio`
-- -----------------------------------------------------
START TRANSACTION;
USE `gestione_asl`;
INSERT INTO `gestione_asl`.`laboratorio` (`id`, `codice`, `ospedale`, `nome`, `piano`, `stanza`,
`responsabile`) VALUES (1, 1, 1, 'Tamponi virologia', 3, 20, 'dottore');
INSERT INTO `gestione_asl`.`laboratorio` (`id`, `codice`, `ospedale`, `nome`, `piano`, `stanza`,
`responsabile`) VALUES (2, 1, 2, 'lab. ematologia', 0, 1, 'klsbbo');
INSERT INTO `gestione_asl`.`laboratorio` (`id`, `codice`, `ospedale`, `nome`, `piano`, `stanza`,
`responsabile`) VALUES (3, 2, 2, 'Radiografia', 1, 1, 'cxoprc');

COMMIT;


-- -----------------------------------------------------
-- Data for table `gestione_asl`.`esame_prenotato`
-- -----------------------------------------------------
START TRANSACTION;
USE `gestione_asl`;
INSERT INTO `gestione_asl`.`esame_prenotato` (`codice`, `data`, `ora`, `costo`, `urgente`,
`prenotazione`, `paziente`, `tipo`, `laboratorio`) VALUES (1, '2022-01-01', '11:35', 26.50, 0, 1,
1, 'ECG', 2);
INSERT INTO `gestione_asl`.`esame_prenotato` (`codice`, `data`, `ora`, `costo`, `urgente`,
`prenotazione`, `paziente`, `tipo`, `laboratorio`) VALUES (4, '2021-10-10', '09:45', 123.45, 0,
2, 2, 'TAC', 1);
INSERT INTO `gestione_asl`.`esame_prenotato` (`codice`, `data`, `ora`, `costo`, `urgente`,
`prenotazione`, `paziente`, `tipo`, `laboratorio`) VALUES (5, '2021-11-11', '12:34', 10, 1, 2, 1,
'ECG', 1);

COMMIT;


-- -----------------------------------------------------
-- Data for table `gestione_asl`.`parametro`
-- -----------------------------------------------------
START TRANSACTION;
USE `gestione_asl`;
INSERT INTO `gestione_asl`.`parametro` (`esame`, `nome`, `valore`, `misura`) VALUES (2,
'emoglobina', 0.123456789, '%');
INSERT INTO `gestione_asl`.`parametro` (`esame`, `nome`, `valore`, `misura`) VALUES (2,
'colesterolo', 25.8, 'mg/dL');
INSERT INTO `gestione_asl`.`parametro` (`esame`, `nome`, `valore`, `misura`) VALUES (2,
'piastrine', 175, 'migliaia/mL');
INSERT INTO `gestione_asl`.`parametro` (`esame`, `nome`, `valore`, `misura`) VALUES (3, 'rna
sarscov2', 1, NULL);
INSERT INTO `gestione_asl`.`parametro` (`esame`, `nome`, `valore`, `misura`) VALUES (2, 'globuli
bianchi', 123456789, NULL);

COMMIT;


-- -----------------------------------------------------
-- Data for table `gestione_asl`.`specializzazione`
-- -----------------------------------------------------
START TRANSACTION;
USE `gestione_asl`;
INSERT INTO `gestione_asl`.`specializzazione` (`primario`, `disciplina`) VALUES ('fcanhn',
'Virologia');
INSERT INTO `gestione_asl`.`specializzazione` (`primario`, `disciplina`) VALUES ('klsbbo',
'ematologia');
INSERT INTO `gestione_asl`.`specializzazione` (`primario`, `disciplina`) VALUES ('cxoprc',
```

```
'cardiochirurgia');

COMMIT;


-- -----------------------------------------------------
-- Data for table `gestione_asl`.`associazione`
-- -----------------------------------------------------
START TRANSACTION;
USE `gestione_asl`;
INSERT INTO `gestione_asl`.`associazione` (`volontario`, `nome`) VALUES ('vlnsnt', 'onlus medici
in pensione');

COMMIT;


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

Codice del Front-End

Viene riportato il codice dei file .c del thin-client.

main.c

```
#include "defines.h"

typedef enum{
	AMMINISTRATORE = 1,
	CUP,
	PERSONALE,
	FAILED_LOGIN
} role_t;

struct configuration conf;

static MYSQL *conn; // pointer to connection handler

static role_t attempt_login(MYSQL *conn, char *username, char *password) {
	MYSQL_STMT *stmt;
	MYSQL_BIND param[3]; // IN, IN, OUT

	int role = 0;

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

	// clean memory for param
	memset((void *)param, 0, sizeof(param));

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

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

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

        // binding param
        if(mysql_stmt_bind_param(stmt, param) != 0) {
                print_stmt_error(stmt, "mysql_stmt_bind_param() failed");
                exit(EXIT_FAILURE);
        }

        // execute stmt
        if(mysql_stmt_execute(stmt) != 0) {
                print_stmt_error(stmt, "mysql_stmt_execute() failed");
                exit(EXIT_FAILURE);
        }

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

        // binding out param
        if(mysql_stmt_bind_result(stmt, param)) {
                print_stmt_error(stmt, "mysql_stmt_bind_result() failed\n");
                goto err;
        }

        // retrieve output param
        if(mysql_stmt_fetch(stmt)) {
                print_stmt_error(stmt, "mysql_stmt_fetch() failed\n");
                goto err;
        }

        // deallocate stmt handler
        mysql_stmt_close(stmt);
        return role;

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

int main() {
        role_t role;

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

        // init connection handler
        conn = mysql_init(NULL);
        if (conn == NULL) {
                print_error(conn, "mysql_init() failed\n");
                exit(EXIT_FAILURE);
        }

        // connect to server with db login account
        if(mysql_real_connect(conn, conf.host, conf.db_username, conf.db_password, conf.database,
conf.port, NULL, CLIENT_MULTI_STATEMENTS | CLIENT_MULTI_RESULTS) == NULL) {
                print_error(conn, "mysql_real_connect() failed\n");
                mysql_close(conn);
                exit(EXIT_FAILURE);
        }
```

```
	// input login
	printf("Username: ");
	getInput(128, conf.username, false);
	printf("Password: ");
	getInput(128, conf.password, true);

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

	// change db account and run account specific code
	switch(role) {
		case AMMINISTRATORE:
			run_as_admin(conn);
			break;
		case CUP:
			run_as_cup(conn);
			break;
		case PERSONALE:
			run_as_personale(conn);
			break;
		default:
			fprintf(stderr, "Invalid condition at %s:%d\n", __FILE__, __LINE__);
			abort();
	}

	// disconnect from server
	mysql_close(conn);
	printf("\nA presto!\n\n");
	exit(EXIT_SUCCESS);
}
```

admin.c

```
#include "defines.h"

static void inserisci_tipo_esame(MYSQL *conn) {
	MYSQL_STMT *stmt;
	MYSQL_BIND param[1];

	// input for the routine
	char nome[46];

	if(!setup_prepared_stmt(&stmt, "call inserisci_tipo_esame(?)", conn))
		finish_with_stmt_error(conn, stmt, "Inizializzazione inserisci tipo esame
fallita\n", false);

	// clean memory for param
	memset( (void *) param, 0, sizeof(param));

	// filling parameters
	printf("Nome del nuovo tipo di esame: ");
	getInput(46, nome, false);

	// set up parameters
	param[0].buffer_type = MYSQL_TYPE_STRING;
	param[0].buffer = (void *) nome;
	param[0].buffer_length = strlen(nome);

	// binding parameters
	if(mysql_stmt_bind_param(stmt, param) != 0)
		finish_with_stmt_error(conn, stmt, "Binding inserisci tipo esame fallito\n", true);

	// execute stmt
```

```
	if(mysql_stmt_execute(stmt) != 0)
		print_stmt_error(stmt, "Execute inserisci tipo esame fallito\n");
	else
		printf("\nNuovo tipo esame (%s) inserito correttamente.\n", nome);

	// deallocate stmt handler
	mysql_stmt_close(stmt);
}

static void cambia_stato_tipo_esame(MYSQL *conn) {
	MYSQL_STMT *stmt;
	MYSQL_BIND param[2];

	// input for the routine
	char nome[46];
	my_bool disponibile;

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

	if(!setup_prepared_stmt(&stmt, "call stato_tipo_esame(?,?)", conn))
		finish_with_stmt_error(conn, stmt, "Inizializzazione cambia stato tipo esame
fallita\n", false);

	// clean memory for param
	memset( (void *) param, 0, sizeof(param));

	// filling parameters
	printf("Nome del tipo di esame: ");
	getInput(46, nome, false);

	printf("Stato del tipo di esame:\n");
	printf("\t1) Disponibile\n");
	printf("\t2) NON disponibile\n");
	r = multiChoice("Seleziona stato", options, 2);

	// set true or false
	switch(r) {
		case '1':
			disponibile = true;
			break;
		case '2':
			disponibile = false;
			break;
		default:
			fprintf(stderr, "Invalid condition at %s:%d\n", __FILE__, __LINE__);
			abort();
	}

	// set up parameters
	param[0].buffer_type = MYSQL_TYPE_STRING;
	param[0].buffer = (void *) nome;
	param[0].buffer_length = strlen(nome);

	param[1].buffer_type = MYSQL_TYPE_TINY;
	param[1].buffer = (char *) &disponibile;
	param[1].buffer_length = sizeof(disponibile);

	// binding parameters
	if(mysql_stmt_bind_param(stmt, param) != 0)
		finish_with_stmt_error(conn, stmt, "Binding cambia stato tipo esame fallito\n",
true);

	// execute stmt
	if(mysql_stmt_execute(stmt) != 0)
		print_stmt_error(stmt, "Execute cambia stato tipo esame fallita\n");
```

```
	else
		printf("\nStato del tipo di esame %s correttamente cambiato.\n", nome);

	// deallocate stmt handler
	mysql_stmt_close(stmt);
}

static void report_personale(MYSQL *conn, int opzione) {
	if(opzione != 0 && opzione != 1) {
		printf("Errore: report_personale(MYSQL *conn, int opzione)\t opzione deve essere 0
o 1\n");
		return;
	}

	MYSQL_STMT *stmt;
	MYSQL_BIND param[2];

	char cf[17];

	int num_exams;

	printf("\nInserisci codice fiscale: ");
	getInput(17, cf, false);

	if(!setup_prepared_stmt(&stmt, "call report_esami_eseguiti(?,?)", conn))
		finish_with_stmt_error(conn, stmt, "Inizializzazione report_personale fallita\n",
false);

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

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

	param[1].buffer_type = MYSQL_TYPE_LONG;
	param[1].buffer = &opzione;
	param[1].buffer_length = sizeof(opzione);

	if (mysql_stmt_bind_param(stmt, param) != 0)
		finish_with_stmt_error(conn, stmt, "Binding report_personale fallito\n", true);

	if (mysql_stmt_execute(stmt) != 0)
		finish_with_stmt_error(conn, stmt, "Execute report_paziente fallita\n", true);

	num_exams = dump_result_set(conn, stmt, "\nEsami eseguiti:"); // my DE does not recognize
MYSQL_BIND.is_null_value

	if(mysql_stmt_next_result(stmt) > 0)
		print_stmt_error(stmt, "Errore nello scorrere i result sets di
report_personale()\n");

	if(opzione == 0)
		printf("\nEsami eseguiti nell'ultimo mese: %d\n", num_exams);
	if(opzione == 1)
		printf("\nEsami eseguiti nell'ultimo anno: %d\n", num_exams);

	mysql_stmt_close(stmt);
}

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

	while(true) {
		printf("\033[2J\033[H");
		printf("ADMIN - Gestione Esami\n");
```

```
                printf("*** COSA VUOI FARE ***\n\n");
                printf("1) Aggiungi tipo esame\n");
                printf("2) Cambia stato tipo esame\n");
                printf("3) Indietro\n");

                op = multiChoice("Scegli un'opzione", options, 3);

                switch(op) {
                        case '1':
                                inserisci_tipo_esame(conn);
                                break;
                        case '2':
                                cambia_stato_tipo_esame(conn);
                                break;
                        case '3':
                                return;
                        default:
                                fprintf(stderr, "Invalid condition at %s:%d\n", __FILE__, __LINE__);
                                abort();
                }
                printf("\npremi il tasto INVIO per continuare\n");
                getchar();
        }
}

void run_as_admin(MYSQL *conn) {
        char options[6] = {'1', '2', '3', '4', '5', '6'};
        char op;

        printf("Switching to ADMIN role...\n\n");

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

        if(mysql_change_user(conn, conf.db_username, conf.db_password, conf.database)) {
                print_error(conn, "mysql_change_user() failed");
                exit(EXIT_FAILURE);
        }

        while(true) {
                printf("\033[2J\033[H");
                printf("ADMIN\n");
                printf("*** COSA VUOI FARE ***\n\n");
                printf("1) Gestione Personale\n");
                printf("2) Gestione Ospedali\n");
                printf("3) Gestione Esami\n");
                printf("4) Report mensile\n");
                printf("5) Report annuale\n");
                printf("6) Esci\n");
                op = multiChoice("Scegli un'opzione", options, 6);
                switch(op) {
                        case '1':
                                gestione_personale(conn);
                                break;
                        case '2':
                                gestione_ospedali(conn);
                                break;
                        case '3':
                                gestione_esami(conn);
                                break;
                        case '4':
                                report_personale(conn, 0);
                                break;
                        case '5':
```

```
                                report_personale(conn, 1);
                                break;
                        case '6':
                                return;
                        default:
                                fprintf(stderr, "Invalid condition at %s:%d\n", __FILE__, __LINE__);
                                abort();
                }

                printf("\npremi il tasto INVIO per continuare\n");
                getchar();
        }
}
```

admin_modifica_ospedale.c

```
#include "defines.h"

static void modifica_info_ospedale(MYSQL *conn) {
        MYSQL_STMT *stmt;
        MYSQL_BIND param[3]; // nome, indirizzo, responsabile

        char codice[46];
        int  codice_int;
        int colonna;
        char var_str[46]; // used to send the new value of the attribute to update

        // print hospitals
        stampa_lista_ospedali(conn);

        // select hospital to update
        printf("\nInserisci codice ospedale: ");
        getInput(46, codice, false);
        codice_int = atoi(codice);

        // select which attribute to update
        char options[3] = {'1', '2', '3'};
        char op;
        printf("\nQuale campo modificare?\n");
        printf("1) Nome Ospedale\n");
        printf("2) Indirizzo\n");
        printf("3) Responsabile Ospedale\n");
        op = multiChoice("Scegli un'opzione", options, 3);

        colonna = op - '0';

        switch(op) {
                case '1':
                        printf("\nInserisci nuovo nome ospedale: ");
                        getInput(46, var_str, false);
                        break;
                case '2':
                        printf("\nInserisci nuovo indirizzo ospedale: ");
                        getInput(46, var_str, false);
                        break;
                case '3':
                        printf("\nRicordati che il nuovo responsabile deve essere un membro del
personale di tipo medico o primario, assegnato ad un reparto dell'ospedale che stai
modificando");
                        printf("\nInserisci Codice Fiscale nuovo responsabile: ");
                        getInput(17, var_str, false);
                        break;
                default:
```

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

        // prepare, bind, execute, close
        if(!setup_prepared_stmt(&stmt, "call modifica_ospedale(?,?,?)", conn))
                finish_with_stmt_error(conn, stmt, "Inizializzazione statement modifica ospedale
fallita\n", false);

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

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

        param[1].buffer_type = MYSQL_TYPE_LONG;
        param[1].buffer = &colonna;
        param[1].buffer_length = sizeof(colonna);

        param[2].buffer_type = MYSQL_TYPE_STRING;
        param[2].buffer = var_str;
        param[2].buffer_length = strlen(var_str);

        if(mysql_stmt_bind_param(stmt, param) != 0)
                finish_with_stmt_error(conn, stmt, "Binding statement modifica ospedale fallita\n",
true);

        if(mysql_stmt_execute(stmt) != 0)
                print_stmt_error(stmt, "Execute statement modifica ospedale fallita\n");
        else
                printf("\nModifica avvenuta correttamente.");

        mysql_stmt_close(stmt);
}

static void aggiungi_reparto(MYSQL *conn) {
        MYSQL_STMT *stmt;
        MYSQL_BIND param[5]; // osp., nome, tel., new primario, specializ.

        char ospedale[46];
        int  ospedale_int;
        char nome[46];
        char telefono[46];
        char primario[17];
        char specializzazione[46];

        // get input
        printf("\nInserisci codice ospedale: ");
        getInput(46, ospedale, false);
        ospedale_int = atoi(ospedale);
        printf("\nInserisci nome nuovo reparo: ");
        getInput(46, nome, false);
        printf("\nInserisci telefono nuovo reparto: ");
        getInput(46, telefono, false);
        printf("\nAttenzione: selezionare un medico dallo stesso ospedale che non sia già
primario.");
        printf("\nInserisci codice fiscale del nuovo primario: ");
        getInput(17, primario, false);
        printf("\nInserisci specializzazione del nuovo primario: ");
        getInput(46, specializzazione, false);

        // prepare, bind, execute, close
        if(!setup_prepared_stmt(&stmt, "call inserisci_reparto(?,?,?,?,?)", conn))
                finish_with_stmt_error(conn, stmt, "Inizializzazione statement inserisci reparto
fallita\n", false);
```

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

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

	param[1].buffer_type = MYSQL_TYPE_STRING;
	param[1].buffer = nome;
	param[1].buffer_length = strlen(nome);

	param[2].buffer_type = MYSQL_TYPE_STRING;
	param[2].buffer = telefono;
	param[2].buffer_length = strlen(telefono);

	param[3].buffer_type = MYSQL_TYPE_STRING;
	param[3].buffer = primario;
	param[3].buffer_length = strlen(primario);

	param[4].buffer_type = MYSQL_TYPE_STRING;
	param[4].buffer = specializzazione;
	param[4].buffer_length = strlen(specializzazione);

	if(mysql_stmt_bind_param(stmt, param) != 0)
		finish_with_stmt_error(conn, stmt, "Binding statement inserisci reparto fallita\n",
true);

	if(mysql_stmt_execute(stmt) != 0)
		print_stmt_error(stmt, "Execute statement inserisci reparto fallita\n");
	else
		printf("Reparto aggiunto correttamente\n");

	mysql_stmt_close(stmt);
}

static void modifica_reparto(MYSQL *conn) {
	MYSQL_STMT *stmt;
	MYSQL_BIND param[4]; // id_rep, colonna, var_str, var_specializ

	char reparto[46];
	int  reparto_int;
	int colonna;
	char var_str[46]; // used to send the new value of the attribute to update
	char var_specializzazione[46];
	my_bool var_specializzazione_is_null = true;

	// ask which ward to update
	printf("\nInserisci ID reparto: ");
	getInput(46, reparto, false);
	reparto_int = atoi(reparto);

	// select the attribute to update
	char options[3] = {'1', '2', '3'};
	char op;
	printf("\nQuale campo modificare?\n");
	printf("1) Nome reparto\n");
	printf("2) Telefono\n");
	printf("3) Primario reparto\n");
	op = multiChoice("Scegli un'opzione", options, 3);

	colonna = op - '0';

	switch(op) {
		case '1':
			printf("\nInserisci nuovo nome reparto: ");
			getInput(46, var_str, false);
			break;
```

```
		case '2':
			printf("\nInserisci nuovo telefono reparto: ");
			getInput(46, var_str, false);
			break;
		case '3':
			printf("\nRicordati che il nuovo primario deve essere un membro del
personale di tipo medico, assegnato ad un reparto dell'ospedale che stai modificando");
			printf("\nInserisci Codice Fiscale nuovo primario: ");
			getInput(17, var_str, false);
			var_specializzazione_is_null = false;
			printf("Inserisci specializzazione nuovo primario reparto: ");
			getInput(46, var_specializzazione, false);
			break;
		default:
			fprintf(stderr, "Invalid condition at %s:%d\n", __FILE__, __LINE__);
			abort();
	}

	// prepare, bind, execute, close
	if(!setup_prepared_stmt(&stmt, "call modifica_reparto(?,?,?,?)", conn))
		finish_with_stmt_error(conn, stmt, "Inizializzazione statement modifica reparto
fallita\n", false);

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

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

	param[1].buffer_type = MYSQL_TYPE_LONG;
	param[1].buffer = &colonna;
	param[1].buffer_length = sizeof(colonna);

	param[2].buffer_type = MYSQL_TYPE_STRING;
	param[2].buffer = var_str;
	param[2].buffer_length = strlen(var_str);

	param[3].buffer_type = MYSQL_TYPE_STRING;
	param[3].buffer = var_specializzazione;
	param[3].buffer_length = strlen(var_specializzazione);
	param[3].is_null = &var_specializzazione_is_null;

	if(mysql_stmt_bind_param(stmt, param) != 0)
		finish_with_stmt_error(conn, stmt, "Binding statement modifica reparto fallita\n",
true);

	if(mysql_stmt_execute(stmt) != 0)
		print_stmt_error(stmt, "Execute statement modifica reparto fallita\n");
	else
		printf("\nModifica avvenuta correttamente.");

	mysql_stmt_close(stmt);
}

static void elimina_reparto(MYSQL *conn) {
	MYSQL_STMT *stmt;
	MYSQL_BIND param[2];

	char reparto[46];
	int reparto_int;

	int new_reparto;

	// scan input
	printf("\nInserisci id reparto: ");
	getInput(46, reparto, false);
```

```
	reparto_int = atoi(reparto);

	if(!setup_prepared_stmt(&stmt, "call cancella_reparto(?,?)", conn))
		finish_with_stmt_error(conn, stmt, "Inizialiazzazione cancella reparto fallita\n",
false);

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

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

	param[1].buffer_type = MYSQL_TYPE_LONG;
	param[1].buffer = &new_reparto;
	param[1].buffer_length = sizeof(new_reparto);

	if(mysql_stmt_bind_param(stmt, param) != 0)
		finish_with_stmt_error(conn, stmt, "Binding cancella reparto fallito\n", true);

	if(mysql_stmt_execute(stmt) != 0) {
		print_stmt_error(stmt, "Execute cancella reparto fallita\n");
		goto out;
	}

	// get the new ward id where every personnel member from the deleted ward is moved
	memset(param, 0, sizeof(param));
	param[0].buffer_type = MYSQL_TYPE_LONG;
	param[0].buffer = &new_reparto;
	param[0].buffer_length = sizeof(new_reparto);

	if(mysql_stmt_bind_result(stmt, param))
		finish_with_stmt_error(conn, stmt, "Binding risultato cancella reparto fallito",
true);

	if(mysql_stmt_store_result(stmt))
		finish_with_stmt_error(conn, stmt, "Store result cancella reparto fallita", true);

	while(!mysql_stmt_fetch(stmt)) // 1x1 result set = new_reparto

	if(mysql_stmt_next_result(stmt) > 0)
		finish_with_stmt_error(conn, stmt, "Errore nello scorrere i risultati in cancella
reparto\n", true);

	printf("Reparto eliminato correttamente\nTutti i membri del personale sono stati spostati
nel reparto con ID = %d\n", new_reparto);

 out:
	mysql_stmt_close(stmt);
}

bool aggiungi_laboratorio(MYSQL *conn, bool from_args, char *resp, int osp_code) {
	bool success = false;

	MYSQL_STMT *stmt;
	MYSQL_BIND param[5]; // osp., nome, piano, stanza, new responsabile

	char ospedale[46];
	int  ospedale_int;
	char nome[46];
	char piano[46];
	int piano_int;
	char stanza[46];
	int stanza_int;
	char responsabile[17];

	if(from_args) {
```

```
		ospedale_int = osp_code;
		strcpy(responsabile, resp);
	}

	// get input
	if(!from_args) {
		printf("\nInserisci codice ospedale: ");
		getInput(46, ospedale, false);
		ospedale_int = atoi(ospedale);
	}
	printf("\nInserisci nome nuovo laboratorio: ");
	getInput(46, nome, false);
	printf("Inserisci numero piano nuovo laboratorio: ");
	getInput(46, piano, false);
	piano_int = atoi(piano);
	printf("Inserisci numero stanza nuovo laboratorio: ");
	getInput(46, stanza, false);
	stanza_int = atoi(stanza);
	if(!from_args) {
		printf("\nAttenzione: selezionare un medico dallo stesso ospedale che non sia già
responsabile di un laboratorio.");
		printf("\nInserisci codice fiscale del nuovo responsabile: ");
		getInput(17, responsabile, false);
	}

	// prepare, bind, execute, close
	if(!setup_prepared_stmt(&stmt, "call inserisci_laboratorio(?,?,?,?,?)", conn))
		finish_with_stmt_error(conn, stmt, "Inizializzazione statement inserisci
laboratorio fallita\n", false);

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

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

	param[1].buffer_type = MYSQL_TYPE_STRING;
	param[1].buffer = nome;
	param[1].buffer_length = strlen(nome);

	param[2].buffer_type = MYSQL_TYPE_LONG;
	param[2].buffer = &piano_int;
	param[2].buffer_length = sizeof(piano_int);

	param[3].buffer_type = MYSQL_TYPE_LONG;
	param[3].buffer = &stanza_int;
	param[3].buffer_length = sizeof(stanza_int);

	param[4].buffer_type = MYSQL_TYPE_STRING;
	param[4].buffer = responsabile;
	param[4].buffer_length = strlen(responsabile);

	if(mysql_stmt_bind_param(stmt, param) != 0)
		finish_with_stmt_error(conn, stmt, "Binding statement inserisci laboratorio
fallita\n", true);

	if(mysql_stmt_execute(stmt) != 0)
		print_stmt_error(stmt, "Execute statement inserisci laboratorio fallita\n");
	else {
		printf("Nuovo laboratorio inserito correttamente.\n");
		success = true;
	}

	mysql_stmt_close(stmt);

	return success;
```

```
}

static void modifica_laboratorio(MYSQL *conn) {
	MYSQL_STMT *stmt;
	MYSQL_BIND param[4]; // lab ID, colonna, var_str, var_int

	char laboratorio[46];
	int lab_int;
	int colonna;
	char var_str[46];
	char var[46];
	int  var_int;
	my_bool var_str_is_null = true;
	my_bool var_int_is_null = true;

	// laboratorio(id, codice, ospedale, nome, piano, stanza, respons)
	// var_str is used to send the new value of one of this attributes:      nome,
responsabile
	// var_int is used to send the new value of one of this attributes: piano, stanza

	// get ID laboratorio
	printf("\nInserisci Id laboratorio: ");
	getInput(46, laboratorio, false);
	lab_int = atoi(laboratorio);

	// select which attribute to update
	char options[4] = {'1', '2', '3', '4'};
	char op;
	printf("\nQuale campo modificare?\n");
	printf("1) Nome laboratorio\n");
	printf("2) Piano\n");
	printf("3) Stanza\n");
	printf("4) Responsabile\n");
	op = multiChoice("Scegli un'opzione", options, 4);

	colonna = op - '0';

	switch(op) {
		case '1':
			var_str_is_null = false;
			printf("Nuovo nome del laboratorio: ");
			getInput(46, var_str, false);
			break;
		case '2':
			var_int_is_null = false;
			printf("\nInserisci nuovo numero di piano del laboratorio: ");
			getInput(46, var, false);
			var_int = atoi(var);
			break;
		case '3':
			var_int_is_null = false;
			printf("\nInserisci nuovo numero di stanza del laboratorio: ");
			getInput(46, var, false);
			var_int = atoi(var);
			break;
		case '4':
			var_str_is_null = false;
			printf("\nAttenzione: selezionare un medico dallo stesso ospedale che non
sia già responsabile di un laboratorio.");
			printf("\nInserisci codice fiscale del nuovo responsabile del laboratorio:
");
			getInput(17, var_str, false);
			break;
		default:
			fprintf(stderr, "Invalid condition at %s:%d\n", __FILE__, __LINE__);
			abort();
```

```
	}

	// prepare, bind, execute, close
	if(!setup_prepared_stmt(&stmt, "call modifica_laboratorio(?,?,?,?)", conn))
		finish_with_stmt_error(conn, stmt, "Inizializzazione statement modifica laboratorio
fallita\n", false);

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

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

	param[1].buffer_type = MYSQL_TYPE_LONG;
	param[1].buffer = &colonna;
	param[1].buffer_length = sizeof(colonna);

	param[2].buffer_type = MYSQL_TYPE_STRING;
	param[2].buffer = var_str;
	param[2].buffer_length = strlen(var_str);
	param[2].is_null = &var_str_is_null;

	param[3].buffer_type = MYSQL_TYPE_LONG;
	param[3].buffer = &var_int;
	param[3].buffer_length = sizeof(var_int);
	param[3].is_null = &var_int_is_null;

	if(mysql_stmt_bind_param(stmt, param) != 0)
		finish_with_stmt_error(conn, stmt, "Binding statement modifica laboratorio
fallita\n", true);

	if(mysql_stmt_execute(stmt) != 0)
		print_stmt_error(stmt, "Execute statement modifica laboratorio fallita\n");
	else
		printf("\nModifica avvenuta correttamente.");

	mysql_stmt_close(stmt);
}

static void elimina_laboratorio(MYSQL *conn) {
	MYSQL_STMT *stmt;
	MYSQL_BIND param[1];

	char laboratorio[46];
	int laboratorio_int;

	printf("\nInserisci id laboratorio: ");
	getInput(46, laboratorio, false);
	laboratorio_int = atoi(laboratorio);

	if(!setup_prepared_stmt(&stmt, "call cancella_laboratorio(?)", conn))
		finish_with_stmt_error(conn, stmt, "Inizialiazzazione cancella laboratorio
fallita\n", false);

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

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

	if(mysql_stmt_bind_param(stmt, param) != 0)
		finish_with_stmt_error(conn, stmt, "Binding cancella laboratorio fallito\n", true);

	if(mysql_stmt_execute(stmt) != 0)
		print_stmt_error(stmt, "Execute cancella laboratorio fallita\n");
	else
```

```
                printf("Laboratorio eliminato correttamente.\n");

        mysql_stmt_close(stmt);
}

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

        while(true) {
                printf("\033[2J\033[H");
                printf("ADMIN - Gestione Ospedali - Modifica Ospedale\n");
                printf("*** COSA VUOI FARE ***\n\n");
                printf("1) Modifica info ospedale\n");
                printf("2) Aggiungi reparto\n");
                printf("3) Modifica reparto\n");
                printf("4) Elimina reparto\n");
                printf("5) Aggiungi laboratorio\n");
                printf("6) Modifica laboratorio\n");
                printf("7) Elimina laboratorio\n");
                printf("8) Indietro\n");

                op = multiChoice("Scegli un'opzione", options, 8);

                switch(op) {
                        case '1':
                                modifica_info_ospedale(conn);
                                break;
                        case '2':
                                aggiungi_reparto(conn);
                                break;
                        case '3':
                                modifica_reparto(conn);
                                break;
                        case '4':
                                elimina_reparto(conn);
                                break;
                        case '5':
                                aggiungi_laboratorio(conn, false, NULL, 0);
                                break;
                        case '6':
                                modifica_laboratorio(conn);
                                break;
                        case '7':
                                elimina_laboratorio(conn);
                                break;
                        case '8':
                                return;
                        default:
                                fprintf(stderr, "Invalid condition at %s:%d\n", __FILE__, __LINE__);
                                abort();
                }
                printf("\npremi il tasto INVIO per continuare\n");
                getchar();
        }
}
```

admin_ospedali.c

```
#include "defines.h"

void stampa_lista_ospedali(MYSQL *conn) {
        MYSQL_STMT *stmt;
```

```
	if(!setup_prepared_stmt(&stmt, "call lista_ospedali()", conn))
		finish_with_stmt_error(conn, stmt, "Inizialiazzazione lista ospedali fallita\n",
false);

	if(mysql_stmt_execute(stmt) != 0) {
		print_stmt_error(stmt, "Execute lista ospedali fallita\n");
		goto out;
	}

	dump_result_set(conn, stmt, "\nLista degli ospedali della ASL");
	if(mysql_stmt_next_result(stmt) > 0)
		finish_with_stmt_error(conn, stmt, "Errore nello scorrere i result sets lista
ospedali di stampa_info_ospedale()\n", true);
 out:
	mysql_stmt_close(stmt);
}

static void stampa_info_ospedale(MYSQL *conn) {
	MYSQL_STMT *stmt;
	MYSQL_BIND param[1];

	char ospedale[46];
	int ospedale_int;

	// print hospitals
	stampa_lista_ospedali(conn);

	// select hospital to print
	printf("\nInserisci codice ospedale: ");
	getInput(46, ospedale, false);
	ospedale_int = atoi(ospedale);

	if(!setup_prepared_stmt(&stmt, "call info_ospedale(?)", conn))
		finish_with_stmt_error(conn, stmt, "Inizialiazzazione info ospedale fallita\n",
false);

	// bind, execute
	memset(param, 0, sizeof(param));

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

	if(mysql_stmt_bind_param(stmt, param) != 0)
		finish_with_stmt_error(conn, stmt, "Binding info ospedale fallito\n", true);

	if(mysql_stmt_execute(stmt) != 0) {
		print_stmt_error(stmt, "Execute info ospedale fallita\n");
		goto out;
	}

	// print the 3 result sets returned
	dump_result_set(conn, stmt, "\nInfo degll'ospedale");
	if(mysql_stmt_next_result(stmt) > 0)
		finish_with_stmt_error(conn, stmt, "Errore nello scorrere i result sets info
ospedale di stampa_info_ospedale()\n", true);

	dump_result_set(conn, stmt, "\nLista dei reparti dell'ospedale");
	if(mysql_stmt_next_result(stmt) > 0)
		finish_with_stmt_error(conn, stmt, "Errore nello scorrere i result sets dei reparti
di stampa_info_ospedale()\n", true);

	dump_result_set(conn, stmt, "\nLista dei laboratori dell'ospedale");
	if(mysql_stmt_next_result(stmt) > 0)
		finish_with_stmt_error(conn, stmt, "Errore nello scorrere i result sets dei lab di
```

```
stampa_info_ospedale()\n", true);

 out:
	mysql_stmt_close(stmt);
}

static void aggiungi_ospedale(MYSQL *conn) {
	MYSQL_STMT *stmt;
	MYSQL_BIND param[7]; // nome_osp, indir_osp, medico, nome_rep, tel_rep, specializzaz, OUT
osp_code

	// input for the routine
	char nome_osp[46];
	char indirizzo_osp[46];
	char cf_medico[17];
	char nome_rep[46];
	char tel_rep[46];
	char specializzazione[46];

	int osp_code;

	bool success = false;

	if(!setup_prepared_stmt(&stmt, "call inserisci_ospedale(?,?,?,?,?,?,?)", conn))
		finish_with_stmt_error(conn, stmt, "Inizializzazione inserisci ospedale fallita\n",
false);

	// clean memory for param
	memset( (void *) param, 0, sizeof(param));

	// filling parameters
	printf("\nAttenzione: è necessario inserire il codice fiscale di un membro del personale
di tipo 'medico' già presente che non sia responsabile di un ospedale o un laboratorio.");
	printf("\nIl medico scelto verrà trasferito nel nuovo ospedale e ne sarà responsabile, e
sarà primario del reparto che verrà inserito contestualmente al nuovo ospedale\n");
	printf("\nInserisci Codice Fiscale del responsabile del nuovo ospedale e primario del
nuovo reparto: ");
	getInput(17, cf_medico, false);
	printf("Inserisci specializzazione del nuovo primario: ");
	getInput(46, specializzazione, false);

	printf("Nome del nuovo ospedale: ");
	getInput(46, nome_osp, false);
	printf("Indirizzo del nuovo ospedale: ");
	getInput(46, indirizzo_osp, false);
	printf("Nome del nuovo reparto: ");
	getInput(46, nome_rep, false);
	printf("Telefono del nuovo reparto: ");
	getInput(46, tel_rep, false);

	// set up parameters
	param[0].buffer_type = MYSQL_TYPE_STRING;
	param[0].buffer = (void *) nome_osp;
	param[0].buffer_length = strlen(nome_osp);

	param[1].buffer_type = MYSQL_TYPE_STRING;
	param[1].buffer = (void *) indirizzo_osp;
	param[1].buffer_length = strlen(indirizzo_osp);

	param[2].buffer_type = MYSQL_TYPE_STRING;
	param[2].buffer = (void *) cf_medico;
	param[2].buffer_length = strlen(cf_medico);

	param[3].buffer_type = MYSQL_TYPE_STRING;
	param[3].buffer = (void *) nome_rep;
	param[3].buffer_length = strlen(nome_rep);
```

```
	param[4].buffer_type = MYSQL_TYPE_STRING;
	param[4].buffer = (void *) tel_rep;
	param[4].buffer_length = strlen(tel_rep);

	param[5].buffer_type = MYSQL_TYPE_STRING;
	param[5].buffer = (void *) specializzazione;
	param[5].buffer_length = strlen(specializzazione);

	param[6].buffer_type = MYSQL_TYPE_LONG;
	param[6].buffer = &osp_code;
	param[6].buffer_length = sizeof(osp_code);

	// binding parameters
	if(mysql_stmt_bind_param(stmt, param) != 0)
		finish_with_stmt_error(conn, stmt, "Binding inserisci ospedale fallito\n", true);

	// execute stmt
	if(mysql_stmt_execute(stmt) != 0) {
		print_stmt_error(stmt, "Execute inserisci ospedale fallito\n");
		mysql_stmt_close(stmt);
		return;
	}

	// save the hospital code
	memset(param, 0, sizeof(param));
	param[0].buffer_type = MYSQL_TYPE_LONG;
	param[0].buffer = &osp_code;
	param[0].buffer_length = sizeof(osp_code);

	if(mysql_stmt_bind_result(stmt, param))
		finish_with_stmt_error(conn, stmt, "Binding risultato inserisci ospedale fallito",
true);

	if(mysql_stmt_store_result(stmt))
		finish_with_stmt_error(conn, stmt, "Store result inserisci ospedale fallita",
true);

	while(!mysql_stmt_fetch(stmt)) // 1x1 result set = osp_code

	if(mysql_stmt_next_result(stmt) > 0)
		finish_with_stmt_error(conn, stmt, "Errore nello scorrere i risultati in inserisci
ospedale\n", true);

	// deallocate stmt handler
	mysql_stmt_close(stmt);

	// inserisci un laboratorio in questo ospedale
	success = aggiungi_laboratorio(conn, true, cf_medico, osp_code);

	if(success)
		printf("\nNuovo ospedale inserito correttamente (codice ospedale = %d).\n",
osp_code);
	else
		printf("\nATTENZIONE qualcosa è andato storto ed è stato inserito un ospedale senza
alcun laboratorio\n");
}

static void elimina_ospedale(MYSQL *conn) {
	MYSQL_STMT *stmt;
	MYSQL_BIND param[2];

	char ospedale[46];
	int ospedale_int;

	int new_reparto;
```

```
	// get input
	printf("\nInserisci codice ospedale: ");
	getInput(46, ospedale, false);
	ospedale_int = atoi(ospedale);

	if(!setup_prepared_stmt(&stmt, "call cancella_ospedale(?,?)", conn))
		finish_with_stmt_error(conn, stmt, "Inizialiazzazione cancella ospedale fallita\n",
false);

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

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

	param[1].buffer_type = MYSQL_TYPE_LONG;
	param[1].buffer = &new_reparto;
	param[1].buffer_length = sizeof(new_reparto);

	if(mysql_stmt_bind_param(stmt, param) != 0)
		finish_with_stmt_error(conn, stmt, "Binding cancellla ospedale fallito\n", true);

	if(mysql_stmt_execute(stmt) != 0) {
		print_stmt_error(stmt, "Execute cancella ospedale fallita\n");
		goto out;
	}

	// get the new ward id where every personnel member from the deleted hospital is moved
	memset(param, 0, sizeof(param));
	param[0].buffer_type = MYSQL_TYPE_LONG;
	param[0].buffer = &new_reparto;
	param[0].buffer_length = sizeof(new_reparto);

	if(mysql_stmt_bind_result(stmt, param))
		finish_with_stmt_error(conn, stmt, "Binding risultato cancella ospedale fallito",
true);

	if(mysql_stmt_store_result(stmt))
		finish_with_stmt_error(conn, stmt, "Store result cancella ospedale fallita", true);

	while(!mysql_stmt_fetch(stmt)) // 1x1 result set = new_reparto

	if(mysql_stmt_next_result(stmt) > 0)
		finish_with_stmt_error(conn, stmt, "Errore nello scorrere i risultati in cancella
ospedale\n", true);

	printf("Ospedale eliminato correttamente\nTutti i membri del personale sono stati spostati
nel reparto con ID = %d\n", new_reparto);

 out:
	mysql_stmt_close(stmt);
}

void gestione_ospedali(MYSQL *conn) {
	char options[5] = {'1', '2', '3', '4', '5'};
	char op;

	while(true) {
		printf("\033[2J\033[H");
		printf("ADMIN - Gestione Ospedali\n");
		printf("*** COSA VUOI FARE ***\n\n");
		printf("1) Stampa info ospedale\n");
		printf("2) Aggiungi ospedale\n");
		printf("3) Modifica ospedale\n");
		printf("4) Elimina ospedale\n");
```

```
		printf("5) Indietro\n");

		op = multiChoice("Scegli un'opzione", options, 5);

		switch(op) {
			case '1':
				stampa_info_ospedale(conn);
				break;
			case '2':
				aggiungi_ospedale(conn);
				break;
			case '3':
				gestione_interno_ospedale(conn);
				break;
			case '4':
				elimina_ospedale(conn);
				break;
			case '5':
				return;
			default:
				fprintf(stderr, "Invalid condition at %s:%d\n", __FILE__, __LINE__);
				abort();
		}
		printf("\npremi il tasto INVIO per continuare\n");
		getchar();
	}
}
```

admin_personale.c

```
#include "defines.h"

static void anagrafica_personale(MYSQL *conn, char *cf) {
	MYSQL_STMT *stmt;
	MYSQL_BIND param[1];

	int status;

	if(!setup_prepared_stmt(&stmt, "call anagrafica_personale(?)", conn))
		finish_with_stmt_error(conn, stmt, "Inizializzazione anagrafica personale
fallita\n", false);

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

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

	if(mysql_stmt_bind_param(stmt, param) != 0)
		finish_with_stmt_error(conn, stmt, "Binding anagrafica personale fallita\n", true);

	if(mysql_stmt_execute(stmt) != 0) {
		print_stmt_error(stmt, "Execute anagrafica personale fallita\n");
		goto out;
	}

	dump_result_set(conn, stmt, "\nAnagrafica personale");

	// print the degrees of the ward responsible or the association of the volunteer
	do {
		status = mysql_stmt_next_result(stmt);
		dump_result_set(conn, stmt, "");
```

```
		if(status > 0)
			finish_with_stmt_error(conn, stmt, "Errore nello scorrere i result sets di
anagrafica_personale()\n", true);
	} while (status == 0);

 out:
	mysql_stmt_close(stmt);
}

static void crea_utente(MYSQL *conn) {
	MYSQL_STMT *stmt;
	MYSQL_BIND param[3];

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

	// input for the routine
	char username[46];
	char password[46];
	char ruolo[46];

	// Get the required information
	printf("\nUsername: ");
	getInput(46, username, false);
	printf("password: ");
	getInput(46, password, true);
	printf("Assegna un ruolo possibile:\n");
	printf("\t1) Amministratore\n");
	printf("\t2) CUP\n");
	printf("\t3) Personale\n");
	r = multiChoice("Seleziona ruolo", options, 3);

	// Convert role into enum value
	switch(r) {
		case '1':
			strcpy(ruolo, "amministratore");
			break;
		case '2':
			strcpy(ruolo, "cup");
			break;
		case '3':
			strcpy(ruolo, "personale");
			break;
		default:
			fprintf(stderr, "Invalid condition at %s:%d\n", __FILE__, __LINE__);
			abort();
	}

	// Prepare stored procedure call
	if(!setup_prepared_stmt(&stmt, "call crea_utente(?, ?, ?)", conn))
		finish_with_stmt_error(conn, stmt, "Inizializzazione crea utente fallita\n",
false);

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

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

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

	param[2].buffer_type = MYSQL_TYPE_VAR_STRING;
	param[2].buffer = ruolo;
```

```
	param[2].buffer_length = strlen(ruolo);

	if (mysql_stmt_bind_param(stmt, param) != 0)
		finish_with_stmt_error(conn, stmt, "Binding crea utente fallito\n", true);

	// Run procedure
	if (mysql_stmt_execute(stmt) != 0)
		print_stmt_error(stmt, "Execute crea utente fallita\n");
	else
		printf("Utente aggiunto correttamente...\n");

	mysql_stmt_close(stmt);
}

static void modifica_associazione_volontariato(MYSQL *conn) {
	MYSQL_STMT *stmt;
	MYSQL_BIND param[2];

	char cf[17];
	char associazione[46];

	printf("\nCodice Fiscale del volontario: ");
	getInput(17, cf, false);
	printf("Nome associazione: ");
	getInput(46, associazione, false);

	if(!setup_prepared_stmt(&stmt, "call aggiorna_associazione_vol(?,?)", conn))
		finish_with_stmt_error(conn, stmt, "Inizializzazione modifica associazione
volontario fallita\n", false);

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

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

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

	if(mysql_stmt_bind_param(stmt, param) != 0)
		finish_with_stmt_error(conn, stmt, "Binding modifica associazione volontario
fallita\n", true);

	if(mysql_stmt_execute(stmt) != 0)
		print_stmt_error(stmt, "Execute modifica associazione volontario fallita\n");
	else
		printf("Modifica avvenuta correttamente.\n");

	mysql_stmt_close(stmt);
}

static void inserisci_personale(MYSQL *conn) {
	MYSQL_STMT *stmt;
	MYSQL_BIND param[7]; // cf, nome, cognome, domicilio, tipo, reparto, username

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

	// input for the routine
	char cf[17];
	char nome[46];
	char cognome[46];
	char domicilio[46];
	char tipo[46];
	char reparto[46];
```

```
	int reparto_int;
	char username[46];

	// Get the required information
	printf("\nCodice Fiscale: ");
	getInput(17, cf, false);
	printf("Nome: ");
	getInput(46, nome, false);
	printf("Cognome: ");
	getInput(46, cognome, false);
	printf("Domicilio: ");
	getInput(46, domicilio, false);
	printf("Username: ");
	getInput(46, username, false);
	printf("ID Reparto: ");
	getInput(46, reparto, false);
	reparto_int = atoi(reparto);
	printf("Assegna un tipo possibile:\n"); // to update to 'primario' you have to use
'aggiungi/modifica reparto'
	printf("\t1) Medico\n");
	printf("\t2) Volontario\n");
	r = multiChoice("Seleziona ruolo", options, 2);

	// Convert role into enum value
	switch(r) {
		case '1':
			strcpy(tipo, "medico");
			break;
		case '2':
			strcpy(tipo, "volontario");
			break;
		default:
			fprintf(stderr, "Invalid condition at %s:%d\n", __FILE__, __LINE__);
			abort();
	}

	// Prepare stored procedure call
	if(!setup_prepared_stmt(&stmt, "call inserisci_personale(?,?,?,?,?,?,?)", conn))
		finish_with_stmt_error(conn, stmt, "Inizializzazione inserisci personale
fallita\n", false);

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

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

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

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

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

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

	param[5].buffer_type = MYSQL_TYPE_LONG;
	param[5].buffer = (void *) &reparto_int;
```

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

	if (mysql_stmt_bind_param(stmt, param) != 0)
		finish_with_stmt_error(conn, stmt, "Binding inserisci personale fallito\n", true);

	// Run procedure
	if (mysql_stmt_execute(stmt) != 0)
		print_stmt_error (stmt, "Execute inserisci personale fallita\n");
	else
		printf("Utente aggiunto correttamente...\n");

	mysql_stmt_close(stmt);
}

static void modifica_personale(MYSQL *conn) {
	MYSQL_STMT *stmt;
	MYSQL_BIND param[4]; // cf, colonna, var_str, var_int

	char cf[17];
	int colonna;
	char var_str[46];
	char reparto[46];
	int  var_int;
	my_bool var_str_is_null = true;
	my_bool var_int_is_null = true;

	// personale(cf varchar16, nome varchar45, cognome varchar45, domicilio varchar45, tipo
enum, reparto int, username varchar45)
	// var_str is used to send the new value of one of this attributes: cf, nome, cognome,
domicilio, tipo, username
	// var_int is used to send the new value of one of this attributes: reparto

	// ask which personnel member to update
	printf("\nInserisci codice fiscale: ");
	getInput(17, cf, false);

	// print her/his info
	anagrafica_personale(conn, cf);

	// select which attribute to update
	char options[7] = {'1', '2', '3', '4', '5', '6', '7'};
	char op;
	printf("\nQuale campo modificare?\n");
	printf("1) Codice Fiscale\n");
	printf("2) Nome\n");
	printf("3) Cognome\n");
	printf("4) Domicilio\n");
	printf("5) Tipo\n");
	printf("6) Reparto\n");
	printf("7) Username\n");
	op = multiChoice("Scegli un'opzione", options, 7);

	colonna = op - '0';

	switch(op) {
		case '1':
			var_str_is_null = false;
			printf("Nuovo Codice Fiscale del membro del personale: ");
			getInput(17, var_str, false);
			break;
		case '2':
			var_str_is_null = false;
			printf("\nInserisci nuovo nome: ");
```

```
                        getInput(46, var_str, false);
                        break;
                case '3':
                        var_str_is_null = false;
                        printf("\nInserisci nuovo cognome: ");
                        getInput(46, var_str, false);
                        break;
                case '4':
                        var_str_is_null = false;
                        printf("\nInserisci nuovo domicilio: ");
                        getInput(46, var_str, false);
                        break;
                case '5':
                        var_str_is_null = false;
                        char type[2] = {'1', '2'};
                        char t;
                        printf("\nScegli nuovo tipo\n");
                        printf("1) Medico\t");
                        printf("2) Volontario\n");
                        t = multiChoice("Scegli un'opzione", type, 2);
                        switch(t) {
                                case '1':
                                        strcpy(var_str, "medico");
                                        break;
                                case '2':
                                        strcpy(var_str, "volontario");
                                        break;
                                default:
                                        fprintf(stderr, "Invalid condition at %s:%d\n", __FILE__,
__LINE__);
                                        abort();
                        }
                        break;
                case '6':
                        var_int_is_null = false;
                        printf("\nInserisci ID del nuovo reparto: ");
                        getInput(46, reparto, false);
                        var_int = atoi(reparto);
                        break;
                case '7':
                        var_str_is_null = false;
                        printf("\nInserisci nuovo username: ");
                        getInput(46, var_str, false);
                        break;
                default:
                        fprintf(stderr, "Invalid condition at %s:%d\n", __FILE__, __LINE__);
                        abort();
        }

        // prepare, bind, execute, close
        if(!setup_prepared_stmt(&stmt, "call modifica_personale(?,?,?,?)", conn))
                finish_with_stmt_error(conn, stmt, "Inizializzazione statement modifica personale
fallita\n", false);

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

        param[0].buffer_type = MYSQL_TYPE_STRING;
        param[0].buffer = cf;
        param[0].buffer_length = strlen(cf);

        param[1].buffer_type = MYSQL_TYPE_LONG;
        param[1].buffer = &colonna;
        param[1].buffer_length = sizeof(colonna);

        param[2].buffer_type = MYSQL_TYPE_STRING;
        param[2].buffer = var_str;
```

```
	param[2].buffer_length = strlen(var_str);
	param[2].is_null = &var_str_is_null;

	param[3].buffer_type = MYSQL_TYPE_LONG;
	param[3].buffer = &var_int;
	param[3].buffer_length = sizeof(var_int);
	param[3].is_null = &var_int_is_null;

	if(mysql_stmt_bind_param(stmt, param) != 0)
		finish_with_stmt_error(conn, stmt, "Binding statement modifica personale
fallita\n", true);

	if(mysql_stmt_execute(stmt) != 0)
		print_stmt_error(stmt, "Execute statement modifica personale fallita\n");
	else
		printf("\nModifica avvenuta correttamente.");

	mysql_stmt_close(stmt);
}

static void cancella_personale(MYSQL *conn) {
	MYSQL_STMT *stmt;
	MYSQL_BIND param[1];

	char cf[17];
	printf("\nInserisci Codice Fiscale del membro del personale da cancellare: ");
	getInput(17, cf, false);

	if(!setup_prepared_stmt(&stmt, "call cancella_personale(?)", conn))
		finish_with_stmt_error(conn, stmt, "Inizializzazione cancella_personale fallita\n",
false);

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

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

	if (mysql_stmt_bind_param(stmt, param) != 0)
		finish_with_stmt_error(conn, stmt, "Binding cancella_personale fallito\n", true);

	if (mysql_stmt_execute(stmt) != 0)
		print_stmt_error(stmt, "Execute cancella_paziente fallita\n");
	else
		printf("Cancellazione avvenuta correttamente...\n");

	mysql_stmt_close(stmt);
}

static void aggiungi_specializzazione(MYSQL *conn) {
	MYSQL_STMT *stmt;
	MYSQL_BIND param[2];

	char options[2] = {'1', '2'};
	char r;
	bool continua = true;

	// input for the routine
	char primario[17];
	char specializzazione[46];

	// Prepare stored procedure call
	if(!setup_prepared_stmt(&stmt, "call aggiungi_specializzazione(?, ?)", conn))
		finish_with_stmt_error(conn, stmt, "Inizializzazione aggiungi specializzazione
fallita\n", false);
```

```
	printf("\nCodice Fiscale del primario: ");
	getInput(17, primario, false);

	while(continua) {
		printf("Specializzazione: ");
		getInput(46, specializzazione, false);

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

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

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

		if (mysql_stmt_bind_param(stmt, param) != 0) {
			finish_with_stmt_error(conn, stmt, "Binding aggiungi specializzazione
fallito\n", true);
		}

		// Run procedure
		if (mysql_stmt_execute(stmt) != 0) {
			print_stmt_error (stmt, "Esecuzione aggiungi specializzazione fallita\n");
			goto end;
		} else {
			printf("Specializzazione aggiunta correttamente\n");
		}

		// chiedi se continuare
		printf("Aggiungere un'altra specializzazione a questo primario?\n");
		printf("\t1) Sì\t2) No\n");
		r = multiChoice("Seleziona opzione", options, 2);
		switch(r) {
			case '1':
				continua = true;
				break;
			case '2':
				continua = false;
				break;
			default:
				fprintf(stderr, "Invalid condition at %s:%d\n", __FILE__, __LINE__);
				abort();
		}
	}
end:
	mysql_stmt_close(stmt);
}

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

	char cf[17];

	while(true) {
		printf("\033[2J\033[H");
		printf("ADMIN - Gestione Personale\n");
		printf("*** COSA VUOI FARE ***\n\n");
		printf("1) Aggiungi utente\n");
		printf("2) Aggiungi personale\n");
		printf("3) Modifica personale\n");
		printf("4) Elimina personale\n");
		printf("5) Anagrafica personale\n");
```

```
                printf("6) Aggiungi specializzazione di un primario\n");
                printf("7) Modifica associazione di un volontario\n");
                printf("8) Indietro\n");

                op = multiChoice("Scegli un'opzione", options, 8);

                switch(op) {
                        case '1':
                                crea_utente(conn);
                                break;
                        case '2':
                                inserisci_personale(conn);
                                break;
                        case '3':
                                modifica_personale(conn);
                                break;
                        case '4':
                                cancella_personale(conn);
                                break;
                        case '5':
                                printf("\nInserisci codice fiscale: ");
                                getInput(17, cf, false);
                                anagrafica_personale(conn, cf);
                                break;
                        case '6':
                                aggiungi_specializzazione(conn);
                                break;
                        case '7':
                                modifica_associazione_volontariato(conn);
                                break;
                        case '8':
                                return;
                        default:
                                fprintf(stderr, "Invalid condition at %s:%d\n", __FILE__, __LINE__);
                                abort();
                }
                printf("\npremi il tasto INVIO per continuare\n");
                getchar();
        }
}
```

cup.c

```
#include "defines.h"

static void stampa_recapiti(MYSQL *conn, int num_ts) {
        MYSQL_STMT *stmt;
        MYSQL_BIND param[1];

        // initialize statement
        if(!setup_prepared_stmt(&stmt, "call lista_recapiti(?)", conn))
                finish_with_stmt_error(conn, stmt, "Inizializzazione lista recapiti paziente
fallita", false);

        // set-up parameters
        memset(param, 0, sizeof(param));

        param[0].buffer_type = MYSQL_TYPE_LONG;
        param[0].buffer = (void *) &num_ts;

        // bind param
        if(mysql_stmt_bind_param(stmt, param) != 0)
                finish_with_stmt_error(conn, stmt, "Binding lista recapiti paziente fallita\n",
```

```
true);

	// run procedure
	if(mysql_stmt_execute(stmt) != 0) {
		print_stmt_error(stmt, "Execute lista recapiti paziente fallita\n");
		goto out;
	}

	dump_result_set(conn, stmt, "\nRecapiti del paziente");	// print result set

	if(mysql_stmt_next_result(stmt) > 0)
		finish_with_stmt_error(conn, stmt, "Errore nello scorrere i result sets di
stampa_recapiti()\n", true);

 out:
	mysql_stmt_close(stmt);
}

static int stampa_anagrafica(MYSQL *conn, bool print_contacts) {
	MYSQL_STMT *stmt;
	MYSQL_BIND param[1];

	int num_ts_int;
	char num_ts[46];

	// print patient info
	if(!setup_prepared_stmt(&stmt, "call anagrafica_paziente(?)", conn))
		finish_with_stmt_error(conn, stmt, "Inizializzazione anagrafica paziente fallita",
false);

	printf("\nNumero tessera sanitaria: ");
	getInput(46, num_ts, false);
	num_ts_int = atoi(num_ts);

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

	param[0].buffer_type = MYSQL_TYPE_LONG;
	param[0].buffer = (void *) &num_ts_int;

	if(mysql_stmt_bind_param(stmt, param) != 0)
		finish_with_stmt_error(conn, stmt, "Binding anagrafica paziente fallita\n", true);

	if(mysql_stmt_execute(stmt) != 0) {
		print_stmt_error(stmt, "Execute anagrafica paziente fallita\n");
		mysql_stmt_close(stmt);
		return -1;
	}

	dump_result_set(conn, stmt, "\nAnagrafica paziente");

	if(mysql_stmt_next_result(stmt) > 0)
		finish_with_stmt_error(conn, stmt, "Errore nello scorrere i result sets di
stampa_anagrafica()\n", true);

	mysql_stmt_close(stmt);

	if(print_contacts) // print patient's contacts?
		stampa_recapiti(conn, num_ts_int);

	return num_ts_int;
}

static void inserisci_paziente(MYSQL *conn) {
	MYSQL_STMT *stmt;
	MYSQL_BIND param[7];
```

```
        // input for the routine
        char num_ts[46];
        int num_ts_int;
        char nome[46];
        char cognome[46];
        char luogo_nascita[46];
        char residenza[46];
        char recapito[46];

        MYSQL_TIME data_nascita;
        char anno[5];
        int anno_int;
        char mese[3];
        int mese_int;
        char giorno[3];
        int giorno_int;

        if(!setup_prepared_stmt(&stmt, "call inserisci_paziente(?,?,?,?,?,?,?)", conn))
                finish_with_stmt_error(conn, stmt, "Inizializzazione statement inserisci paziente
fallita\n", false);

        // clean memory for param
        memset( (void *) param, 0, sizeof(param));

        // filling parameters
        printf("\nNumero tessera sanitaria: ");
        getInput(46, num_ts, false);
        num_ts_int = atoi(num_ts);
        printf("Nome: ");
        getInput(46, nome, false);
        printf("Cognome: ");
        getInput(46, cognome, false);

        printf("Giorno di nascita: ");
        getInput(3, giorno, false);
        giorno_int = atoi(giorno);
        data_nascita.day = giorno_int;

        printf("Mese di nascita: ");
        getInput(3, mese, false);
        mese_int = atoi(mese);
        data_nascita.month = mese_int;

        printf("Anno di nascita: ");
        getInput(5, anno, false);
        anno_int = atoi(anno);
        data_nascita.year = anno_int;

        printf("Luogo di nascita: ");
        getInput(46, luogo_nascita, false);
        printf("Residenza: ");
        getInput(46, residenza, false);
        printf("Recapito: ");
        getInput(46, recapito, false);

        // set up parameters
        param[0].buffer_type = MYSQL_TYPE_LONG;
        param[0].buffer = (void *) &num_ts_int;

        param[1].buffer_type = MYSQL_TYPE_STRING;
        param[1].buffer = (void *) nome;
        param[1].buffer_length = strlen(nome);

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

```
	param[3].buffer_type = MYSQL_TYPE_DATE;
	param[3].buffer = (char *)&data_nascita;
	param[3].is_null = 0;
	param[3].length= 0;

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

	param[5].buffer_type = MYSQL_TYPE_VAR_STRING;
	param[5].buffer = residenza;
	param[5].buffer_length = strlen(residenza);

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

	// binding parameters
	if(mysql_stmt_bind_param(stmt, param) != 0)
		finish_with_stmt_error(conn, stmt, "Binding statement inserisci paziente
fallito\n", true);

	// execute stmt
	if(mysql_stmt_execute(stmt) != 0)
		print_stmt_error(stmt, "Execute statement inserisci paziente fallita\n");
	else
		printf("Paziente inserito correttamente\n");

	// deallocate stmt handler
	mysql_stmt_close(stmt);
}

static void modifica_paziente(MYSQL *conn) {
	// print patient info and save her/his tessera sanitaria number
	int num_ts = stampa_anagrafica(conn, false);
	if(num_ts == -1)
		return;

	// update procedure
	MYSQL_STMT *stmt;
	MYSQL_BIND param[5]; // int numTS, int colonna, int new_numTS, varchar str, data

	int colonna;
	int new_ts_int;
	char new_ts[46];
	char var_str[46];
	MYSQL_TIME var_data;
	char giorno[3], mese[3], anno[5];
	my_bool newts_is_null = true;
	my_bool varstr_is_null = true;
	my_bool data_is_null = true;

	if(!setup_prepared_stmt(&stmt, "call modifica_paziente(?,?,?,?,?)", conn))
		finish_with_stmt_error(conn, stmt, "Inizializzazione statement modifica paziente
fallita\n", false);

	// which attribute to update?
	char options[6] = {'1', '2', '3', '4', '5', '6'};
	char op;
	printf("\nQuale campo modificare?\n");
	printf("1) Numero Tessera Sanitaria\n");
	printf("2) Nome\n");
	printf("3) Cognome\n");
	printf("4) Data di nascita\n");
	printf("5) Luogo di nascita\n");
```

```
        printf("6) Residenza\n");
        op = multiChoice("Scegli un'opzione", options, 6);

        colonna = op - '0';

        switch(op) {
                case '1':
                        newts_is_null = false;
                        printf("Nuovo numero Tessera Sanitaria del paziente: ");
                        getInput(46, new_ts, false);
                        new_ts_int = atoi(new_ts);
                        break;
                case '2':
                        varstr_is_null = false;
                        printf("\nInserisci nuovo nome: ");
                        getInput(46, var_str, false);
                        break;
                case '3':
                        varstr_is_null = false;
                        printf("\nInserisci nuovo cognome: ");
                        getInput(46, var_str, false);
                        break;
                case '4':
                        data_is_null = false;
                        printf("Inserisci nuova data di nascita\n\tGiorno : ");
                        getInput(3, giorno, false);
                        printf("\tMese : ");
                        getInput(3, mese, false);
                        printf("\tAnno [formato YYYY] : ");
                        getInput(5, anno, false);
                        var_data.year = atoi(anno);
                        var_data.month = atoi(mese);
                        var_data.day = atoi(giorno);
                        break;
                case '5':
                        varstr_is_null = false;
                        printf("\nInserisci nuovo luogo di nascita: ");
                        getInput(46, var_str, false);
                        break;
                case '6':
                        varstr_is_null = false;
                        printf("\nInserisci nuova residenza: ");
                        getInput(46, var_str, false);
                        break;
                default:
                        fprintf(stderr, "Invalid condition at %s:%d\n", __FILE__, __LINE__);
                        abort();
        }

        // param[2,3,4] initially NULL, the selected column was changed to not NULL in the switch-
case
        memset(param, 0, sizeof(param));

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

        param[1].buffer_type = MYSQL_TYPE_LONG;
        param[1].buffer = &colonna;
        param[1].buffer_length = sizeof(colonna);

        param[2].buffer_type = MYSQL_TYPE_LONG;
        param[2].buffer = &new_ts_int;
        param[2].buffer_length = sizeof(new_ts_int);
        param[2].is_null = &newts_is_null;
```

```
	param[3].buffer_type = MYSQL_TYPE_VAR_STRING;
	param[3].buffer = var_str;
	param[3].buffer_length = strlen(var_str);
	param[3].is_null = &varstr_is_null;

	param[4].buffer_type = MYSQL_TYPE_DATE;
	param[4].buffer = &var_data;
	param[4].buffer_length = sizeof(var_data);
	param[4].is_null = &data_is_null;

	// bind param
	if(mysql_stmt_bind_param(stmt, param) != 0)
		finish_with_stmt_error(conn, stmt, "Binding statement modifica paziente fallita\n",
true);

	// run procedure
	if(mysql_stmt_execute(stmt) != 0){
		if( strcmp( mysql_stmt_sqlstate(stmt), "45004") == 0 ) // deleting a non existing
patient
			printf("\nERROR: il paziente da modificare non esiste!\n");

		print_stmt_error(stmt, "Execute statement modifica paziente fallita\n");
	} else {
		printf("Paziente modificato correttamente\n");
	}

	mysql_stmt_close(stmt);
}

static void cancella_paziente(MYSQL *conn) {
	MYSQL_STMT *stmt;
	MYSQL_BIND param[1];

	// input for the routine
	char num_ts[46];
	int num_ts_int;

	if(!setup_prepared_stmt(&stmt, "call cancella_paziente(?)", conn))
		finish_with_stmt_error(conn, stmt, "Inizializzazione statement cancella paziente
fallita\n", false);

	// clean memory for param
	memset( (void *) param, 0, sizeof(param));

	// filling parameters
	printf("\nNumero tessera sanitaria: ");
	getInput(46, num_ts, false);
	num_ts_int = atoi(num_ts);

	// set up parameters
	param[0].buffer_type = MYSQL_TYPE_LONG;
	param[0].buffer = (void *) &num_ts_int;

	// binding parameters
	if(mysql_stmt_bind_param(stmt, param) != 0)
		finish_with_stmt_error(conn, stmt, "Binding statement cancella paziente fallito\n",
true);

	// execute stmt
	if(mysql_stmt_execute(stmt) != 0) {
		if(strcmp(mysql_stmt_sqlstate(stmt), "45003") == 0) // deleting a non existing
patient
			printf("Il paziente da eliminare non esiste!\n");
		print_stmt_error(stmt, "Execute statement cancella paziente fallita\n");
	} else {
		printf("Paziente con tessera sanitaria n° %d eliminato correttamente.\n",
```

```
num_ts_int);
	}

	// deallocate stmt handler
	mysql_stmt_close(stmt);
}

static void inserisci_recapito(MYSQL *conn) {
	MYSQL_STMT *stmt;
	MYSQL_BIND param[2];

	// inputs for the routine
	char num_ts[46];
	int num_ts_int;
	char recapito[46];

	// get the information
	printf("\nNumero Tessera Sanitaria: ");
	getInput(46, num_ts, false);
	num_ts_int = atoi(num_ts);
	printf("\nRecapito: ");
	getInput(46, recapito, false);

	// prepare stored procedure call
	if(!setup_prepared_stmt(&stmt, "call inserisci_recapito(?,?)", conn))
		finish_with_stmt_error(conn, stmt, "Inizializzazione statement inserisci recapito
fallita\n", false);

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

	param[0].buffer_type = MYSQL_TYPE_LONG;
	param[0].buffer = (void *) &num_ts_int;

	param[1].buffer_type = MYSQL_TYPE_STRING;
	param[1].buffer = (void *) recapito;
	param[1].buffer_length = strlen(recapito);

	// bind param
	if(mysql_stmt_bind_param(stmt, param) != 0)
		finish_with_stmt_error(conn, stmt, "Binding statement inserisci recapito
fallita\n", true);

	// run procedure
	if(mysql_stmt_execute(stmt) != 0)
		print_stmt_error(stmt, "Execute statement inserisci recapito fallita\n");
	else
		printf("Recapito aggiunto correttamente\n");

	mysql_stmt_close(stmt);
}

static void cancella_recapito(MYSQL *conn) {
	int num_ts_int;
	char num_ts[46];

	char recapito[46];

	// ask for tessera sanitaria number
	printf("\nNumero tessera sanitaria: ");
	getInput(46, num_ts, false);
	num_ts_int = atoi(num_ts);

	// print contacts for that patient
	stampa_recapiti(conn, num_ts_int);
```

```
	// ask which to delete
	printf("\nInserire recapito da cancellare: ");
	getInput(46, recapito, false);

	// prepare stmt
	MYSQL_STMT *stmt;
	MYSQL_BIND param[2]; // int numTS, varchar campo

	if(!setup_prepared_stmt(&stmt, "call cancella_recapito(?,?)", conn))
		finish_with_stmt_error(conn, stmt, "Inizializzazione cancella recapito fallita",
false);

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

	param[0].buffer_type = MYSQL_TYPE_LONG;
	param[0].buffer = (void *) &num_ts_int;
	param[0].buffer_length = sizeof(num_ts_int);

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

	// bind
	if(mysql_stmt_bind_param(stmt, param) != 0)
		finish_with_stmt_error(conn, stmt, "Binding cancella recapito fallita\n", true);

	// exec and close
	if(mysql_stmt_execute(stmt) != 0) {
		if( strcmp( mysql_stmt_sqlstate(stmt), "45001") == 0 ) // deleting a non existing
contact
			printf("\nERROR: il contatto da eliminare non esiste!\n");
		if(strcmp(mysql_stmt_sqlstate(stmt), "45002") == 0) // the user can't delete the
only contact of a patient
			printf("\nERROR: impossibile cancellare l'unico contatto di un
paziente!\n");
		print_stmt_error(stmt, "Execute cancella recapito fallita\n");
	} else {
		printf("Contatto eliminato correttamente.\n");
	}

	mysql_stmt_close(stmt);
}

static void prenota_esame(MYSQL *conn) {
	MYSQL_STMT *stmt;
	MYSQL_BIND param[8];

	// inputs
	MYSQL_TIME data_ts;		// param[0]
	char giorno[3], mese[3], anno[5];
	char orario[6];				// param[1]
	char ore[3], minuti[3];
	float costo_f;			// param[2]
	char costo[46];
	my_bool urgente;		// param[3]
	int prenotazione = 0; // param[4]
	my_bool pren_is_null = true;
	int paziente_int;		// param[5]
	char paziente[46];
	char tipo[46];			// param[6]
	int laboratorio_int;  // param[7]
	char laboratorio[46];

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

```
	char continua[2] = {'1', '2'};
	char c;

	bool prenota_ancora = true;

	while(prenota_ancora) {
		if(prenotazione != 0)
			pren_is_null = false;

		/* print reservable exams */
		if(!setup_prepared_stmt(&stmt, "call esami_disponibili()", conn))
			finish_with_stmt_error(conn, stmt, "Inizialiazzazione lista esami
disponibili fallita\n", false);
		if(mysql_stmt_execute(stmt) != 0)
			finish_with_stmt_error(conn, stmt, "Execute lista esami disponibili
fallita\n", true);
		dump_result_set(conn, stmt, "\nLista degli esami disponibili");
		if(mysql_stmt_next_result(stmt) > 0)
			finish_with_stmt_error(conn, stmt, "Errore nello scorrere i result sets
lista esami disponibili\n", true);
		mysql_stmt_close(stmt);

		// select exam type
		printf("\nTipo esame: ");
		getInput(46, tipo, false);

		/* show laboratories */
		if(!setup_prepared_stmt(&stmt, "call lista_laboratori()", conn))
			finish_with_stmt_error(conn, stmt, "Inizializzazione lista laboratori
fallita\n", false);
		if(mysql_stmt_execute(stmt) != 0)
			finish_with_stmt_error(conn, stmt, "Execute lista laboratori fallita\n",
true);
		dump_result_set(conn, stmt, "\nLista dei laboratori");
		if(mysql_stmt_next_result(stmt) > 0)
			finish_with_stmt_error(conn, stmt, "Errore nello scorrere i result sets
lista laboratori\n", true);
		mysql_stmt_close(stmt);

		// select lab
		printf("\nInserisci ID laboratorio: ");
		getInput(46, laboratorio, false);
		laboratorio_int = atoi(laboratorio);

		// inputs
		printf("Data esame\n\tGiorno [formato GG] : ");
		getInput(3, giorno, false);
		printf("\tMese [formato MM] : ");
		getInput(3, mese, false);
		printf("\tAnno [formato YYYY] : ");
		getInput(5, anno, false);
		data_ts.year = atoi(anno);
		data_ts.month = atoi(mese);
		data_ts.day = atoi(giorno);

		printf("Orario esame\n\tOre [formato HH] : ");
		getInput(3, ore, false);
		printf("\tMinuti [formato MM] : ");
		getInput(3, minuti, false);
		// send time as a "HH:MM" string...
		sprintf(orario, "%d:%d", atoi(ore), atoi(minuti));

		printf("Costo esame [formato € xx.xx ] :  € ");
		getInput(46, costo, false);
		costo_f = atof(costo);
```

```
                printf("Numero Tessera Sanitaria del paziente: ");
                getInput(46, paziente, false);
                paziente_int = atoi(paziente);

                printf("\nEsame prescritto con urgenza?\n");
                printf("\t1) URGENTE");
                printf("\t2) non urgente\n");
                u = multiChoice("Seleziona", options, 2);
                switch(u) {
                        case '1':
                                urgente = true;
                                break;
                        case '2':
                                urgente = false;
                                break;
                        default:
                                fprintf(stderr, "Invalid condition at %s:%d\n", __FILE__, __LINE__);
                                abort();
                }

                // prepare
                if(!setup_prepared_stmt(&stmt, "call prenota_esame(?,?,?,?,?,?,?,?)", conn))
                        finish_with_stmt_error(conn, stmt, "Inizializzazione statement prenota
esame fallita\n", false);

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

                param[0].buffer_type = MYSQL_TYPE_DATE;
                param[0].buffer = &data_ts;
                param[0].buffer_length = sizeof(data_ts);

                // send the time as a "HH:MM" string
                param[1].buffer_type = MYSQL_TYPE_VAR_STRING;
                param[1].buffer = orario;
                param[1].buffer_length = strlen(orario);

                param[2].buffer_type = MYSQL_TYPE_FLOAT;
                param[2].buffer = &costo_f;
                param[2].buffer_length = sizeof(costo_f);

                param[3].buffer_type = MYSQL_TYPE_TINY;
                param[3].buffer = (char *) &urgente;
                param[3].buffer_length = sizeof(urgente);

                // for the first iteration param[4] is null
                // then set the reservation number with the value returned by the prepared
statement
                // see the sql routine implementation for more info
                param[4].buffer_type = MYSQL_TYPE_LONG;
                param[4].buffer = &prenotazione;
                param[4].buffer_length = sizeof(prenotazione);
                param[4].is_null = &pren_is_null;

                param[5].buffer_type = MYSQL_TYPE_LONG;
                param[5].buffer = &paziente_int;
                param[5].buffer_length = sizeof(paziente_int);

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

                param[7].buffer_type = MYSQL_TYPE_LONG;
                param[7].buffer = &laboratorio_int;
                param[7].buffer_length = sizeof(laboratorio_int);
```

```
		// bind
		if(mysql_stmt_bind_param(stmt, param) != 0)
			finish_with_stmt_error(conn, stmt, "Binding prenota_esame() fallito\n",
true);

		// execute
		if(mysql_stmt_execute(stmt) != 0) {
			if(mysql_stmt_errno(stmt) == 1062)
				printf("\nUSER ERROR: impossibile prenotare lo stesso tipo di esame
nella stessa data per lo stesso paziente!\n");
			if(mysql_stmt_errno(stmt) == 1644)
				printf("\nUSER ERROR: impossibile prenotare un esame nel
passato!\n");
			print_stmt_error(stmt, "Errore nell'esecuzione dello statement
prenota_esame()\n");
			mysql_stmt_close(stmt);
			return;
		}

		// save the reservation number returned by the prepared statement
		memset(param, 0, sizeof(param));
		param[0].buffer_type = MYSQL_TYPE_LONG;
		param[0].buffer = &prenotazione;
		param[0].buffer_length = sizeof(prenotazione);

		if(mysql_stmt_bind_result(stmt, param))
			finish_with_stmt_error(conn, stmt, "Binding prenota esame fallita", true);

		if(mysql_stmt_store_result(stmt))
			finish_with_stmt_error(conn, stmt, "Store result prenota esame fallita",
true);

		while(!mysql_stmt_fetch(stmt)) // 1x1 result set = prenotazione
			printf("\n\t*** NUMERO DI PRENOTAZIONE ASSEGNATO:  %d  ***\n",
prenotazione);

		// ask to continue
		printf("\nPrenotare altri esami?\n");
		printf("\t1) Sì");
		printf("\t2) No\n");
		c = multiChoice("Seleziona", continua, 2);
		switch(c) {
			case '1':
				prenota_ancora = true;
				break;
			case '2':
				prenota_ancora = false;
				break;
			default:
				fprintf(stderr, "Invalid condition at %s:%d\n", __FILE__, __LINE__);
				abort();
		}

		if(mysql_stmt_next_result(stmt) > 0)
			finish_with_stmt_error(conn, stmt, "Errore nello scorrere i risultati in
prenota_esame()\n", true);

		// close stmt
		mysql_stmt_close(stmt);
	}
}

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

```
        while(true) {
                printf("\033[2J\033[H");
                printf("CUP - Gestione Anagrafica\n");
                printf("*** COSA VUOI FARE ***\n\n");
                printf("1) Stampa anagrafica\n");
                printf("2) Inserisci paziente\n");
                printf("3) Modifica paziente\n");
                printf("4) Cancella paziente\n");
                printf("5) Aggiungi recapito\n");
                printf("6) Cancella recapito\n");
                printf("7) Indietro\n");

                op = multiChoice("Scegli un'opzione", options, 7);

                switch(op) {
                        case '1':
                                stampa_anagrafica(conn, true);
                                break;
                        case '2':
                                inserisci_paziente(conn);
                                break;
                        case '3':
                                modifica_paziente(conn);
                                break;
                        case '4':
                                cancella_paziente(conn);
                                break;
                        case '5':
                                inserisci_recapito(conn);
                                break;
                        case '6':
                                cancella_recapito(conn);
                                break;
                        case '7':
                                return;
                        default:
                                fprintf(stderr, "Invalid condition at %s:%d\n", __FILE__, __LINE__);
                                abort();
                }
                printf("\npremi il tasto INVIO per continuare\n");
                getchar();
        }
}

void run_as_cup(MYSQL *conn) {
        char options[5] = {'1', '2', '3', '4', '5'};
        char op;

        printf("Switching to CUP role...\n\n");

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

        if(mysql_change_user(conn, conf.db_username, conf.db_password, conf.database)) {
                print_error(conn, "mysql_change_user() failed");
                exit(EXIT_FAILURE);
        }

        while(true) {
                printf("\033[2J\033[H");
                printf("CUP\n");
                printf("*** COSA VUOI FARE ***\n\n");
                printf("1) Gestisci anagrafica\n");
                printf("2) Prenota esame\n");
```

```
                printf("3) Report prenotazione\n");
                printf("4) Report storico paziente\n");
                printf("5) Esci\n");

                op = multiChoice("Scegli un'opzione", options, 5);

                switch(op) {
                        case '1':
                                gestisci_anagrafica(conn);
                                break;
                        case '2':
                                prenota_esame(conn);
                                break;
                        case '3':
                                report_prenotazione(conn);
                                break;
                        case '4':
                                report_paziente(conn);
                                break;
                        case '5':
                                return;
                        default:
                                fprintf(stderr, "Invalid condition at %s:%d\n", __FILE__, __LINE__);
                                abort();
                }
                printf("\npremi il tasto INVIO per continuare\n");
                getchar();
        }
}
```

cup_report.c

```
#include "defines.h"

struct exam_list {
        char type[46];
        char date[11];
        char doctor[92];
        char diagnosis[257];
};

struct exam_list_big {
        char type[46];
        char date[11];
        char doctor[92];
        char diagnosis[257];
        char patient[92];
};

struct reservation_list {
        char type[46];
        char patient[92];
        char date[11];
        char time[6];
        char lab[46];
        char hosp[46];
        bool urgent;
        double cost;
};

struct exam_param {
        char name[46];
        double value;
```

```
	char measure[46];
};

static size_t parse_exparam(MYSQL *conn, MYSQL_STMT *stmt, struct exam_param **params) {
	int status;
	size_t row = 0;
	MYSQL_BIND param[3];

	char nome_param[46];
	double valore;
	char misura[46];
	my_bool is_null;

	if(mysql_stmt_store_result(stmt))
		finish_with_stmt_error(conn, stmt, "Store result parse_exparam fallita", true);

	*params = malloc(mysql_stmt_num_rows(stmt) * sizeof(struct exam_param));

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

	param[0].buffer_type = MYSQL_TYPE_VAR_STRING;
	param[0].buffer = nome_param;
	param[0].buffer_length = 46;

	param[1].buffer_type = MYSQL_TYPE_DOUBLE;
	param[1].buffer = &valore;
	param[1].buffer_length = sizeof(valore);

	param[2].buffer_type = MYSQL_TYPE_VAR_STRING;
	param[2].buffer = misura;
	param[2].buffer_length = 46;
	param[2].is_null = &is_null;

	if(mysql_stmt_bind_result(stmt, param))
		finish_with_stmt_error(conn, stmt, "Binding di parse_exparam fallita\n", true);

	// assemble struct exam_param
	while(true) {
		status = mysql_stmt_fetch(stmt);

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

		strcpy((*params)[row].name, nome_param);
		(*params)[row].value = valore;
		if(is_null)
			strcpy((*params)[row].measure, "");
		else
			strcpy((*params)[row].measure, misura);

		row++;
	}

	return row;
}

static size_t parse_exams(MYSQL *conn, MYSQL_STMT *stmt, struct exam_list **list) {
	int status;
	size_t row = 0;
	my_bool is_null;
	MYSQL_BIND param[5];

	char nome_esame[46];
	MYSQL_TIME data;
	char nome_dottore[46];
	char cognome_dottore[46];
```

```
	char diagnosi[257];

	if(mysql_stmt_store_result(stmt))
		finish_with_stmt_error(conn, stmt, "Store result parse_exams fallita", true);

	*list = malloc(mysql_stmt_num_rows(stmt) * sizeof(struct exam_list));

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

	param[0].buffer_type = MYSQL_TYPE_VAR_STRING;
	param[0].buffer = nome_esame;
	param[0].buffer_length = 46;

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

	param[2].buffer_type = MYSQL_TYPE_VAR_STRING;
	param[2].buffer = nome_dottore;
	param[2].buffer_length = 46;

	param[3].buffer_type = MYSQL_TYPE_VAR_STRING;
	param[3].buffer = cognome_dottore;
	param[3].buffer_length = 46;

	param[4].buffer_type = MYSQL_TYPE_STRING;
	param[4].buffer = diagnosi;
	param[4].buffer_length = 257;
	param[4].is_null = &is_null;

	if(mysql_stmt_bind_result(stmt, param))
		finish_with_stmt_error(conn, stmt, "Binding di parse_exams fallita\n", true);

	// assemble struct exam_list
	while(true) {
		status = mysql_stmt_fetch(stmt);

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

		strcpy((*list)[row].type, nome_esame);
		sprintf((*list)[row].date, "%02d/%02d/%4d", data.day, data.month, data.year);
		sprintf((*list)[row].doctor, "%s %s", cognome_dottore, nome_dottore);
		if(is_null) {
			sprintf((*list)[row].diagnosis, "Nessuna diagnosi inserita.");
		} else {
			sprintf((*list)[row].diagnosis, "%s", diagnosi);
		}

		row++;
	}
	return row;
}

static size_t parse_exams_with_patient_info(MYSQL *conn, MYSQL_STMT *stmt, struct exam_list_big
**list) {
	int status;
	size_t row = 0;
	my_bool is_null;
	MYSQL_BIND param[7];

	char nome_esame[46];
	MYSQL_TIME data;
	char nome_dottore[46];
	char cognome_dottore[46];
	char diagnosi[257];
```

```
	char nome_paz[46];
	char cognome_paz[46];

	if(mysql_stmt_store_result(stmt))
		finish_with_stmt_error(conn, stmt, "Store result parse_exams_with_patient_info
fallita", true);

	*list = malloc(mysql_stmt_num_rows(stmt) * sizeof(struct exam_list_big));

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

	param[0].buffer_type = MYSQL_TYPE_VAR_STRING;
	param[0].buffer = nome_esame;
	param[0].buffer_length = 46;

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

	param[2].buffer_type = MYSQL_TYPE_VAR_STRING;
	param[2].buffer = nome_dottore;
	param[2].buffer_length = 46;

	param[3].buffer_type = MYSQL_TYPE_VAR_STRING;
	param[3].buffer = cognome_dottore;
	param[3].buffer_length = 46;

	param[4].buffer_type = MYSQL_TYPE_STRING;
	param[4].buffer = diagnosi;
	param[4].buffer_length = 257;
	param[4].is_null = &is_null;

	param[5].buffer_type = MYSQL_TYPE_VAR_STRING;
	param[5].buffer = nome_paz;
	param[5].buffer_length = 46;

	param[6].buffer_type = MYSQL_TYPE_VAR_STRING;
	param[6].buffer = cognome_paz;
	param[6].buffer_length = 46;

	if(mysql_stmt_bind_result(stmt, param))
		finish_with_stmt_error(conn, stmt, "Binding di parse_exams fallita\n", true);

	// assemble struct exam_list
	while(true) {
		status = mysql_stmt_fetch(stmt);

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

		strcpy((*list)[row].type, nome_esame);
		sprintf((*list)[row].date, "%02d/%02d/%4d", data.day, data.month, data.year);
		sprintf((*list)[row].doctor, "%s %s", cognome_dottore, nome_dottore);
		if(is_null)
			sprintf((*list)[row].diagnosis, "Nessuna diagnosi inserita.");
		else
			sprintf((*list)[row].diagnosis, "%s", diagnosi);
		sprintf((*list)[row].patient, "%s %s", cognome_paz, nome_paz);

		row++;
	}

	return row;
}

static size_t parse_reservations(MYSQL *conn, MYSQL_STMT *stmt, struct reservation_list **list) {
```

```
	int status;
	size_t row = 0;
	MYSQL_BIND param[9];

	char tipo[46];
	char nome[46];
	char cognome[46];
	MYSQL_TIME data;
	MYSQL_TIME ora;
	char laboratorio[46];
	char ospedale[46];
	my_bool urgente;
	double costo;

	if(mysql_stmt_store_result(stmt))
		finish_with_stmt_error(conn, stmt, "Store result parse_reservations fallita",
true);

	*list = malloc(mysql_stmt_num_rows(stmt) * sizeof(struct reservation_list));

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

	param[0].buffer_type = MYSQL_TYPE_VAR_STRING;
	param[0].buffer = tipo;
	param[0].buffer_length = 46;

	param[1].buffer_type = MYSQL_TYPE_VAR_STRING;
	param[1].buffer = nome;
	param[1].buffer_length = 46;

	param[2].buffer_type = MYSQL_TYPE_VAR_STRING;
	param[2].buffer = cognome;
	param[2].buffer_length = 46;

	param[3].buffer_type = MYSQL_TYPE_DATE;
	param[3].buffer = &data;
	param[3].buffer_length = sizeof(data);

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

	param[5].buffer_type = MYSQL_TYPE_VAR_STRING;
	param[5].buffer = laboratorio;
	param[5].buffer_length = 46;

	param[6].buffer_type = MYSQL_TYPE_VAR_STRING;
	param[6].buffer = ospedale;
	param[6].buffer_length = 46;

	param[7].buffer_type = MYSQL_TYPE_TINY;
	param[7].buffer = (char *) &urgente;
	param[7].buffer_length = sizeof(urgente);

	param[8].buffer_type = MYSQL_TYPE_DOUBLE;
	param[8].buffer = &costo;
	param[8].buffer_length = sizeof(costo);


	if(mysql_stmt_bind_result(stmt, param))
		finish_with_stmt_error(conn, stmt, "Binding di parse_exams fallita\n", true);

	// assemble struct reservation_list
	while(true) {
		status = mysql_stmt_fetch(stmt);
```

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

                strcpy((*list)[row].type, tipo);
                sprintf((*list)[row].patient, "%s %s", cognome, nome);
                sprintf((*list)[row].date, "%02d/%02d/%4d", data.day, data.month, data.year);
                sprintf((*list)[row].time, "%02d:%02d", ora.hour, ora.minute);
                strcpy((*list)[row].lab, laboratorio);
                strcpy((*list)[row].hosp, ospedale);
                if(urgente) {
                        (*list)[row].urgent = true;
                } else {
                        (*list)[row].urgent = false;
                }
                (*list)[row].cost = costo;

                row++;
        }

        return row;
}

void report_paziente(MYSQL* conn) {
        MYSQL_STMT *stmt;
        MYSQL_BIND param[1];

        int status;
        bool first = true;
        struct exam_list *exams;
        size_t i = 0;
        size_t tabelle = 0;

        char num_ts[46];
        int num_ts_int;

        // preapre stmt
        if(!setup_prepared_stmt(&stmt, "call report_storico_paz(?)", conn))
                finish_with_stmt_error(conn, stmt, "Inizializzazione report storico paziente
fallita\n", false);

        // prepare params
        memset(param, 0, sizeof(param));

        param[0].buffer_type = MYSQL_TYPE_LONG;
        param[0].buffer = (void *) &num_ts_int;

        // fill param
        printf("\nNumero tessera sanitaria: ");
        getInput(46, num_ts, false);
        num_ts_int = atoi(num_ts);

        // bind
        if(mysql_stmt_bind_param(stmt, param) != 0)
                finish_with_stmt_error(conn, stmt, "Binding report storico paziente fallita\n",
true);

        // run procedure
        if(mysql_stmt_execute(stmt) != 0) {
                print_stmt_error(stmt, "Execute report storico paziente fallita\n");
                goto out;
        }

        // multiple reuslt sets
        // first table: examType | date | doctor | diagnosys
        // next: parameter | value
```

```
	do {
		// skip OUT variables even if there are not
		if(conn->server_status & SERVER_PS_OUT_PARAMS)
			goto next;

		if(first) {
			tabelle = parse_exams(conn, stmt, &exams);
			first = false;
		} else {
			if(i >= tabelle)
				goto next;

			printf("\n+-----------------------------------------------------------------
------------------");
			printf("\n| Esame:\t%s\n| Eseguito il:\t%s\n| Eseguito da:\t%s\n|
Diagnosi:\t%s\n|", exams[i].type, exams[i].date, exams[i].doctor, exams[i].diagnosis);
			printf("\n|  # Valori:\n|  #\n");

			// dump_result_set() makes float colunmn header 331 wide... printing
without a table layout
			struct exam_param *params;
			size_t righe;
			righe = parse_exparam(conn, stmt, &params);

			for(unsigned int j = 0; j < righe; j++) {
				printf("|  #  - %s : %.02f %s\n", params[j].name, params[j].value,
params[j].measure);
			}
			printf("|\n+-----------------------------------------------------------------
--------------------\n");
			i++;
		}

	next: // -1 = no more results; 0 = more results, keep looking; >0 = error
		status = mysql_stmt_next_result(stmt);
		if(status > 0)
			finish_with_stmt_error(conn, stmt, "Errore nello scorrere i risultati di
report storico paziente\n", true);
	} while (status == 0);

 out:
	mysql_stmt_close(stmt);
}

void report_prenotazione(MYSQL* conn) {
	MYSQL_STMT *stmt;
	MYSQL_BIND param[1];

	int status;
	int giro = 1;
	struct reservation_list *reservations;
	struct exam_list_big *exams;
	size_t i = 0;
	size_t done_exams = 0;

	char num_prenotazione[46];
	int num_prenotazione_int;

	// preapre stmt
	if(!setup_prepared_stmt(&stmt, "call report_prenotazione(?)", conn))
		finish_with_stmt_error(conn, stmt, "Inizializzazione report storico paziente
fallita\n", false);

	// prepare params
	memset(param, 0, sizeof(param));
```

```
	param[0].buffer_type = MYSQL_TYPE_LONG;
	param[0].buffer = (void *) &num_prenotazione_int;

	// fill param
	printf("\nNumero prenotazione: ");
	getInput(46, num_prenotazione, false);
	num_prenotazione_int = atoi(num_prenotazione);

	// bind
	if(mysql_stmt_bind_param(stmt, param) != 0)
		finish_with_stmt_error(conn, stmt, "Binding report storico paziente fallita\n",
true);

	// run procedure
	if(mysql_stmt_execute(stmt) != 0) {
		print_stmt_error(stmt, "Execute report storico paziente fallita\n");
		goto out;
	}

	// multiple reuslt sets
	// first table: type | patient (n+s)| date | time | lab name | H name | cost | urgency
	// secont table: examType | date | doctor | diagnosys | name | surname
	// next: parameter | value

	do {
		// skip OUT variables even if there are not
		if(conn->server_status & SERVER_PS_OUT_PARAMS)
			goto next;

		if(giro == 1) {
			size_t n_reservations = parse_reservations(conn, stmt, &reservations);
			printf("### ESAMI PRENOTATI ANCORA DA ESEGUIRE (%zd)\n", n_reservations);

			// tipo, paz, data, ora, lab, osp, urg, costo
			for(unsigned int r = 0; r < n_reservations; r++) {
				printf("\n %d) Esame: %s\n", r+1, reservations[r].type);
				printf("    Paziente: %s\n", reservations[r].patient);
				printf("    %s ore: %s\n", reservations[r].date,
reservations[r].time);
				printf("    Presso: %s, %s\n", reservations[r].lab,
reservations[r].hosp);
				printf("    Costo: € %.02f\n", reservations[r].cost);
				if(reservations[r].urgent)
					printf("    *** URGENTE ***\n");
			}
		}
		if(giro == 2) {
			done_exams = parse_exams_with_patient_info(conn, stmt, &exams);
			printf("\n\n### ESAMI ESEGUITI (%zd)\n", done_exams);
		}
		if(giro > 2) {
			if(i >= done_exams) {
				goto next;
			}

			printf("\n+-----------------------------------------------------------------
-------------------");
			printf("\n| Esame:\t%s\n| Eseguito il:\t%s\n| Eseguito da:\t%s\n|
Diagnosi:\t%s\n|", exams[i].type, exams[i].date, exams[i].doctor, exams[i].diagnosis);
			printf(" Paziente: %s\n|", exams[i].patient);
			printf("\n|  # Valori:\n|  #\n");

			// dump_result_set() makes float colunmn header 331 wide... printing
without a table layout
			struct exam_param *params;
			size_t righe;
```

```
                        righe = parse_exparam(conn, stmt, &params);

                        for(unsigned int j = 0; j < righe; j++) {
                                printf("|  #  - %s: %.02f %s\n", params[j].name, params[j].value,
params[j].measure);
                        }
                        printf("|\n+-----------------------------------------------------------------
-------------------\n");
                        i++;
                }

        next: // -1 = no more results; 0 = more results, keep looking; >0 = error
                status = mysql_stmt_next_result(stmt);
                giro++;
                if(status > 0)
                        finish_with_stmt_error(conn, stmt, "Errore nello scorrere i risultati di
report storico paziente\n", true);
        } while (status == 0);

 out:
        mysql_stmt_close(stmt);
}
```

defines.h

```
#pragma once

#include <stdlib.h>
#include <stdio.h>
#include <my_global.h>
#include <my_sys.h>
#include <stdbool.h>
#include <mysql.h>


struct configuration {
        char *host;
        char *db_username;
        char *db_password;
        unsigned int port;
        char *database;

        char username[128];
        char password[128];
};

extern struct configuration conf;

// parse.c
extern int parse_config(char *path, struct configuration *conf);

// inout.c
extern char *getInput(unsigned int lung, char *stringa, bool hide);
extern bool yesOrNo(char *domanda, char yes, char no, bool predef, bool insensitive);
extern char multiChoice(char *domanda, char choices[], int num);

// utils.c
extern void print_error (MYSQL *conn, char *message);
extern void print_stmt_error (MYSQL_STMT *stmt, char *message);
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 bool setup_prepared_stmt(MYSQL_STMT **stmt, char *statement, MYSQL *conn);
```

```
extern int dump_result_set(MYSQL *conn, MYSQL_STMT *stmt, char *title);

// cup.c
extern void run_as_cup(MYSQL *conn);

// cup_report.c
extern void report_paziente(MYSQL *conn);
extern void report_prenotazione(MYSQL *conn);

// admin.c
extern void run_as_admin(MYSQL *conn);

// admin_personale.c
extern void gestione_personale(MYSQL *conn);

// admin_ospedali.c
extern void stampa_lista_ospedali(MYSQL *conn);
extern void gestione_ospedali(MYSQL *conn);

// admin_modifica_ospedale.c
extern void gestione_interno_ospedale(MYSQL *conn);
extern bool aggiungi_laboratorio(MYSQL *conn, bool from_args, char *resp, int osp_code);

// personale.c
extern void run_as_personale(MYSQL *conn);
```

inout.c

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

#include "defines.h"

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

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

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

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

		// Cattura i segnali che altrimenti potrebbero far terminare il programma,
lasciando l'utente senza output sulla shell
		sigemptyset(&sa.sa_mask);
```

```
		sa.sa_flags = SA_INTERRUPT; // Per non resettare le system call
		sa.sa_handler = handler;
		(void) sigaction(SIGALRM, &sa, &savealrm);
		(void) sigaction(SIGINT, &sa, &saveint);
		(void) sigaction(SIGHUP, &sa, &savehup);
		(void) sigaction(SIGQUIT, &sa, &savequit);
		(void) sigaction(SIGTERM, &sa, &saveterm);
		(void) sigaction(SIGTSTP, &sa, &savetstp);
		(void) sigaction(SIGTTIN, &sa, &savettin);
		(void) sigaction(SIGTTOU, &sa, &savettou);

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

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

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

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

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

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

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

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

```
		(void) sigaction(SIGTTIN, &savettin, NULL);
		(void) sigaction(SIGTTOU, &savettou, NULL);

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

	return stringa;
}

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


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

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

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

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

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

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

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

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

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

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

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

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

parse.c

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

#include "defines.h"

#define BUFF_SIZE 4096

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

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

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

/**
 * JSON token description.
 * type        type (object, array, string etc.)
 * start       start position in JSON data string
 * end         end position in JSON data string
 */
```

```
typedef struct {
	jsmntype_t type;
	int start;
	int end;
	int size;
#ifdef JSMN_PARENT_LINKS
	int parent;
#endif
} jsmntok_t;

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

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

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

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

	start = parser->pos;

	for (; parser->pos < len && js[parser->pos] != '\0'; parser->pos++) {
		switch (js[parser->pos]) {
#ifndef JSMN_STRICT
			/* In strict mode primitive must be followed by "," or "}" or "]" */
			case ':':
#endif
			case '\t' : case '\r' : case '\n' : case ' ' :
			case ','  : case ']'  : case '}' :
```

```
                        goto found;
                }
                if (js[parser->pos] < 32 || js[parser->pos] >= 127) {
                        parser->pos = start;
                        return JSMN_ERROR_INVAL;
                }
        }
#ifdef JSMN_STRICT
        /* In strict mode primitive must be followed by a comma/object/array */
        parser->pos = start;
        return JSMN_ERROR_PART;
#endif

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

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

        int start = parser->pos;

        parser->pos++;

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

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

                /* Backslash: Quoted symbol expected */
                if (c == '\\' && parser->pos + 1 < len) {
                        int i;
```

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

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

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

                c = js[parser->pos];
                switch (c) {
                        case '{': case '[':
                                count++;
                                if (tokens == NULL) {
                                        break;
                                }
                                token = jsmn_alloc_token(parser, tokens, num_tokens);
                                if (token == NULL)
                                        return JSMN_ERROR_NOMEM;
                                if (parser->toksuper != -1) {
                                        tokens[parser->toksuper].size++;
#ifdef JSMN_PARENT_LINKS
                                        token->parent = parser->toksuper;
#endif
                                }
```

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

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

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

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

        return count;
}

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

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

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

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

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

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

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

	config[fsize] = 0;
	return fsize;
}

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

	load_file(path);

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

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

	/* Loop over all keys of the root object */
	for (i = 1; i < r; i++) {
		if (jsoneq(config, &t[i], "host") == 0) {
```

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

personale.c

```
#include "defines.h"

static void aggiorna_diagnosi(MYSQL *conn) {
        MYSQL_STMT *stmt;
        MYSQL_BIND param[3];

        char codice[46];
        int codice_int;
        char diagnosi[257];

        if(!setup_prepared_stmt(&stmt, "call aggiorna_diagnosi(?,?,?)", conn))
                finish_with_stmt_error(conn, stmt, "Inizializzazione aggiorna_diagnosi fallita\n",
false);

        printf("\nCodice esame: ");
        getInput(46, codice, false);
        codice_int = atoi(codice);

        printf("\nInserisci diagnosi: \n");
        getInput(257, diagnosi, false);

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

        param[0].buffer_type = MYSQL_TYPE_LONG;
        param[0].buffer = (void *) &codice_int;

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

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

        if(mysql_stmt_bind_param(stmt, param) != 0)
                finish_with_stmt_error(conn, stmt, "Binding aggiorna_diagnosi fallito\n", true);
```

```
	if(mysql_stmt_execute(stmt) != 0)
		print_stmt_error(stmt, "Esecuzione aggiorna_diagnosi fallita\n");
	else
		printf("Diagnosi aggiornata\n");

	mysql_stmt_close(stmt);
}

static void inserisci_risultati(MYSQL *conn) {
	MYSQL_STMT *stmt;
	MYSQL_BIND param[3], bind_risultati[4];

	char codice[46];
	int codice_int;
	char diagnosi[257];

	char nome_param[46];
	char valore[46];
	double valore_double;
	char misura[46];

	char options[2] = {'1', '2'};
	char op;
	bool continua = true;

	if(!setup_prepared_stmt(&stmt, "call esegui_esame(?,?,?)", conn))
		finish_with_stmt_error(conn, stmt, "Inizializzazione esegui_esame fallita\n",
false);

	printf("\nCodice esame: ");
	getInput(46, codice, false);
	codice_int = atoi(codice);

	printf("\nInserisci diagnosi: \n");
	getInput(257, diagnosi, false);

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

	param[0].buffer_type = MYSQL_TYPE_LONG;
	param[0].buffer = (void *) &codice_int;

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

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

	if(mysql_stmt_bind_param(stmt, param) != 0)
		finish_with_stmt_error(conn, stmt, "Binding esegui_esame fallito\n", true);

	if(mysql_stmt_execute(stmt) != 0) {
		print_stmt_error(stmt, "Esecuzione esegui_esame fallita\n");
		mysql_stmt_close(stmt);
		return;
	}

	mysql_stmt_close(stmt);

	// insert exam results
	printf("Inserisci parametri:\n");

	if(!setup_prepared_stmt(&stmt, "call inserisci_risultati(?,?,?,?)", conn))
		finish_with_stmt_error(conn, stmt, "Inizializzazione inserisci_risultati
fallita\n", false);
```

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

	bind_risultati[0].buffer_type = MYSQL_TYPE_LONG;
	bind_risultati[0].buffer = (void *) &codice_int;

	bind_risultati[1].buffer_type = MYSQL_TYPE_VAR_STRING;
	bind_risultati[1].buffer = nome_param;
	bind_risultati[1].buffer_length = strlen(nome_param);

	bind_risultati[2].buffer_type = MYSQL_TYPE_DOUBLE;
	bind_risultati[2].buffer = &valore_double;
	bind_risultati[2].buffer_length = sizeof(valore_double);

	bind_risultati[3].buffer_type = MYSQL_TYPE_VAR_STRING;
	bind_risultati[3].buffer = misura;

	while(continua) {
		// chiedi input
		printf("\n\tNome parametro: ");
		getInput(46, nome_param, false);

		printf("\tValore parametro: ");
		getInput(46, valore, false);
		valore_double = atof(valore);

		printf("\tUnità di misura parametro: ");
		getInput(46, misura, false);

		// bind, execute inserisci_risultato
		bind_risultati[1].buffer_length = strlen(nome_param);
		bind_risultati[3].buffer_length = strlen(misura);

		if(mysql_stmt_bind_param(stmt, bind_risultati) != 0)
			finish_with_stmt_error(conn, stmt, "Binding inserisci_risultati fallito\n",
true);

		if(mysql_stmt_execute(stmt) != 0)
			finish_with_stmt_error(conn, stmt, "Esecuzione inserisci_risultati
fallita\n", true);

		printf("\nInserire altri parametri?\t1) SI\t2)NO\n");
		op = multiChoice("Scegli un'opzione", options, 2);
		switch(op) {
			case '1':
				continua = true;
				break;
			case '2':
				continua = false;
				break;
			default:
				fprintf(stderr, "Invalid condition at %s:%d\n", __FILE__, __LINE__);
				abort();
		}
	}

	// close statement
	mysql_stmt_close(stmt);
}

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

	printf("Switching to PERSONNEL role...\n\n");
```

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

	if(mysql_change_user(conn, conf.db_username, conf.db_password, conf.database)) {
		print_error(conn, "mysql_change_user() failed");
		exit(EXIT_FAILURE);
	}

	while(true) {
		printf("\033[2J\033[H");
		printf("PERSONALE\n");
		printf("*** COSA VUOI FARE ***\n\n");
		printf("1) Inserisci risultati esame\n");
		printf("2) Aggiorna diagnosi esame\n");
		printf("3) Esci\n");

		op = multiChoice("Scegli un'opzione", options, 3);

		switch(op) {
			case '1':
				inserisci_risultati(conn);
				break;
			case '2':
				aggiorna_diagnosi(conn);
				break;
			case '3':
				return;
			default:
				fprintf(stderr, "Invalid condition at %s:%d\n", __FILE__, __LINE__);
				abort();
		}

		printf("\npremi il tasto INVIO per continuare\n");
		getchar();
	}
}
```

utils.c

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

#include "defines.h"

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

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

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

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

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

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

        mysql_stmt_attr_set(*stmt, STMT_ATTR_UPDATE_MAX_LENGTH, &update_length);

        return true;
}

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

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

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

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

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

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

	mysql_field_seek (res_set, 0);

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

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

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

	print_dashes(res_set);
}

int dump_result_set(MYSQL *conn, MYSQL_STMT *stmt, char *title)
{
	int rows = 0;
	int i;
	int status;
	int num_fields;       /* number of columns in result */
	MYSQL_FIELD *fields;  /* for result set metadata */
	MYSQL_BIND *rs_bind;  /* for output buffers */
	MYSQL_RES *rs_metadata;
	MYSQL_TIME *date;
	size_t attr_size;

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

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

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

		if((rs_metadata = mysql_stmt_result_metadata(stmt)) == NULL) {
			finish_with_stmt_error(conn, stmt, "Unable to retrieve result metadata\n",
true);
		}
```

```
		dump_result_set_header(rs_metadata);

		fields = mysql_fetch_fields(rs_metadata);

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

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

			// Properly size the parameter buffer
			switch(fields[i].type) {
				case MYSQL_TYPE_DATE:
				case MYSQL_TYPE_TIMESTAMP:
				case MYSQL_TYPE_DATETIME:
				case MYSQL_TYPE_TIME:
					attr_size = sizeof(MYSQL_TIME);
					break;
				case MYSQL_TYPE_FLOAT:
					attr_size = sizeof(float);
					break;
				case MYSQL_TYPE_DOUBLE:
					attr_size = sizeof(double);
					break;
				case MYSQL_TYPE_TINY:
					attr_size = sizeof(signed char);
					break;
				case MYSQL_TYPE_SHORT:
				case MYSQL_TYPE_YEAR:
					attr_size = sizeof(short int);
					break;
				case MYSQL_TYPE_LONG:
				case MYSQL_TYPE_INT24:
					attr_size = sizeof(int);
					break;
				case MYSQL_TYPE_LONGLONG:
					attr_size = sizeof(int);
					break;
				default:
					attr_size = fields[i].max_length;
					break;
			}

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

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

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

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

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

                    rows++;

                    putchar('|');

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

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

                            switch (rs_bind[i].buffer_type) {

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

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

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

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

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

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

                                    default:
                                        printf("ERROR: Unhandled type (%d)\n",
rs_bind[i].buffer_type);
                                        abort();
                            }
                    }
                    putchar('\n');
                    print_dashes(rs_metadata);
            }

            mysql_free_result(rs_metadata); /* free metadata */

            /* free output buffers */
```

```
                for (i = 0; i < num_fields; i++) {
                        free(rs_bind[i].buffer);
                }
                free(rs_bind);
        }
        return rows;
}
```