Basi di Dati e Conoscenza

Progetto A.A. 2020/2021

SISTEMA DI PRENOTAZIONE DI ESAMI IN UNA ASL

0267020

Andrea Pepe

Indice

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
12-16	Esame	Esame disponibile	Possibile ambiguità tra esame medico disponibile, da intendersi come entità astratta, ed esame medico effettivamente eseguito (concreto).
21-40-42	Amministrazione	Amministratori del sistema	È possibile pensare che si faccia riferimento all'amministrazione dell'ospedale, intesa come il responsabile. Inoltre, meglio essere coerenti con la terminologia trovata in precedenza alle righe 14-15.
20-25-38	Responsabile	Primario	Rendere esplicito il concetto che un primario sia un responsabile di un laboratorio e/o di un ospedale.
27	Prenotazione di un esame	Esame prenotato	Le informazioni che si vogliono memorizzare sono relative allo specifico esame e non alla prenotazione di tale esame.
31	Giorno	Data	È preferibile rimanere conformi al termine "data" utilizzato alla linea precedente (30).
36	Personale	Personale del reparto	Rischio di confondere il personale del reparto con il personale del CUP.

Specifiche disambiguate

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 disponibili 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 disponibile. 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 primario. Anche la gestione degli ospedali è in capo agli amministratori del sistema.

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 primario.

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 esame in date diverse. Si noti inoltre che lo stesso esame non può essere ripetuto nella stessa data 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 del reparto, 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 primari degli ospedali e dei laboratori vanno individuati all'interno del personale medico, che può essere gestito unicamente dagli amministratori del sistema.

Per motivi di storicizzazione, gli amministratori del sistema 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
Paziente	Persona che puo' prenotare esami da svolgere presso le ASL		Esame
Esame disponibile	Tipologia di esame medico che la ASL è in grado di svolgere	Esame	Esame
Esame	Esame effettivo che coinvolge un paziente ed il personale dei reparti che lo svolge		Paziente, Personale del reparto, Laboratorio
Ospedale	Struttura medica della ASL, suddivisa in reparti e contenente laboratori		Laboratorio, Reparto, Primario
Laboratorio	Laboratorio di un ospedale, in cui vengono svolti gli esami medici		Ospedale, Primario, Esame

Reparto	Reparto di un ospedale, dove lavorano i membri del personale del reparto		Ospedale, Personale del reparto
Personale del reparto	Personale di un reparto di un ospedale, formato da medici e volontari	Personale	Esame, Medico, Volontario
Medico	Membro del personale del reparto, tra cui identificare i primari	Personale medico	Personale del reparto
Primario	Medico responsabile di un ospedale e/o di un laboratorio	Responsabile	Ospedale, Laboratorio
Volontario	Membro del personale del reparto, facente parte di una associazione di volontariato		Personale del reparto

Raggruppamento dei requisiti in insiemi omogenei

Frasi di carattere generale
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.

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. In fase di prenotazione, è possibile prenotare con un unico codice di prenotazione un numero arbitrario di esami. 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 nella stessa data dallo stesso paziente. 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 agli esami disponibili
Gli esami medici disponibili 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 disponibile.

Frasi relative agli esami
In fase di prenotazione, è possibile prenotare con un unico codice di prenotazione un numero arbitrario di esami. Ciascun esame è associato ad un insieme di valori numerici, riportanti i risultati dei parametri legati allo specifico esame disponibile. 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 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 nella stessa data dallo stesso paziente. Per motivi di storicizzazione, gli amministratori del sistema 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 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 primario. Anche la gestione degli ospedali è in capo agli amministratori del sistema. 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. I primari degli ospedali e dei laboratori vanno individuati all'interno del personale medico, che può essere gestito unicamente dagli amministratori del sistema.

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 primario. Per ogni esame prenotato da parte di un paziente si vuole memorizzare […] il laboratorio presso cui è eseguito.

I primari degli ospedali e dei laboratori vanno individuati all'interno del personale medico, che può essere gestito unicamente dagli amministratori del sistema.

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 del reparto

Il personale del reparto è identificato attraverso il codice fiscale; sono noti inoltre il nome, il cognome e l'indirizzo di domicilio.

Per motivi di storicizzazione, gli amministratori del sistema possono generare dei report che mostrano ciascun membro del personale quanti esami (e quali esami) ha svolto, su base mensile e/o annuale.

Frasi relative ai medici

I primari degli ospedali e dei laboratori vanno individuati all'interno del personale medico, che può essere gestito unicamente dagli amministratori del sistema.

Frasi relative ai primari

Tra il personale del reparto, nel caso dei medici primari del reparto è noto l'elenco delle specializzazioni.

I primari degli ospedali e dei laboratori vanno individuati all'interno del personale medico, che può essere gestito unicamente dagli amministratori del sistema.

Frasi relative ai volontari

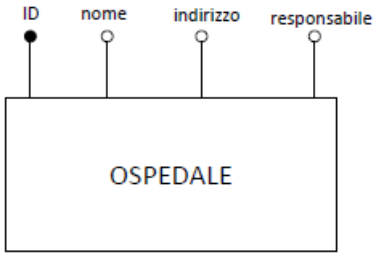
[…] mentre per il personale volontario è noto il nome dell'associazione di appartenenza, se disponibile.

3. Progettazione concettuale

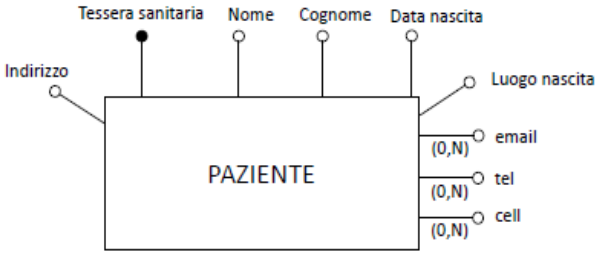
Costruzione dello schema E-R

Nella costruzione dello schema E-R si è cercato di adottare una strategia di tipo bottom-up, iniziando dall'individuazione delle principali entità.

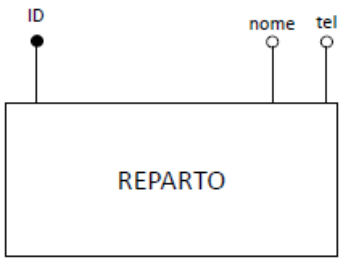
…Gli ospedali della ASL sono identificati da un codice numerico e sono caratterizzati da un nome, un indirizzo e dal nome di un primario. Anche la gestione degli ospedali è in capo agli amministratori del sistema…



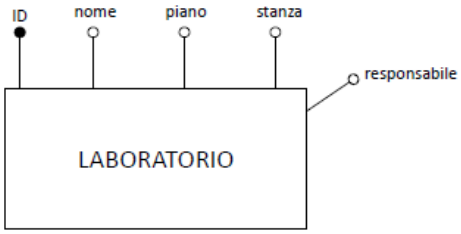
…Ciascun paziente è identificato da un codice di tessera sanitaria ed è caratterizzato da un nome, un cognome, la data ed il luogo di nascita, un indirizzo di residenza ed un insieme arbitrario di recapiti (email, telefono, cellulare) …



…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…



…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 primario…

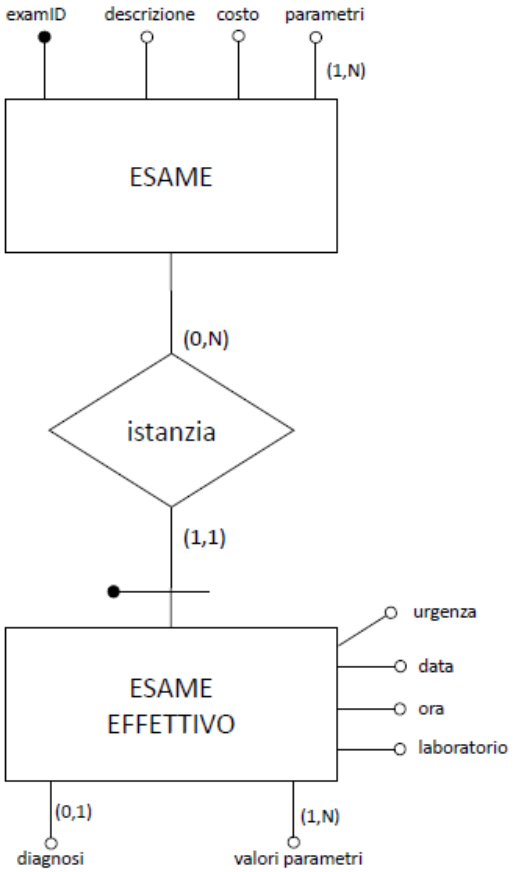


…Gli esami medici disponibili che possono essere eseguiti sono identificati da un codice numerico e sono caratterizzati dalla descrizione di esame medico (ad esempio Radiografia, ecc.). […] Ciascun esame è associato ad un insieme di valori numerici, riportanti i risultati dei parametri legati allo specifico esame disponibile…

…Ciascun esame è associato ad un insieme di valori numerici, riportanti i risultati dei parametri legati allo specifico esame disponibile. 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…

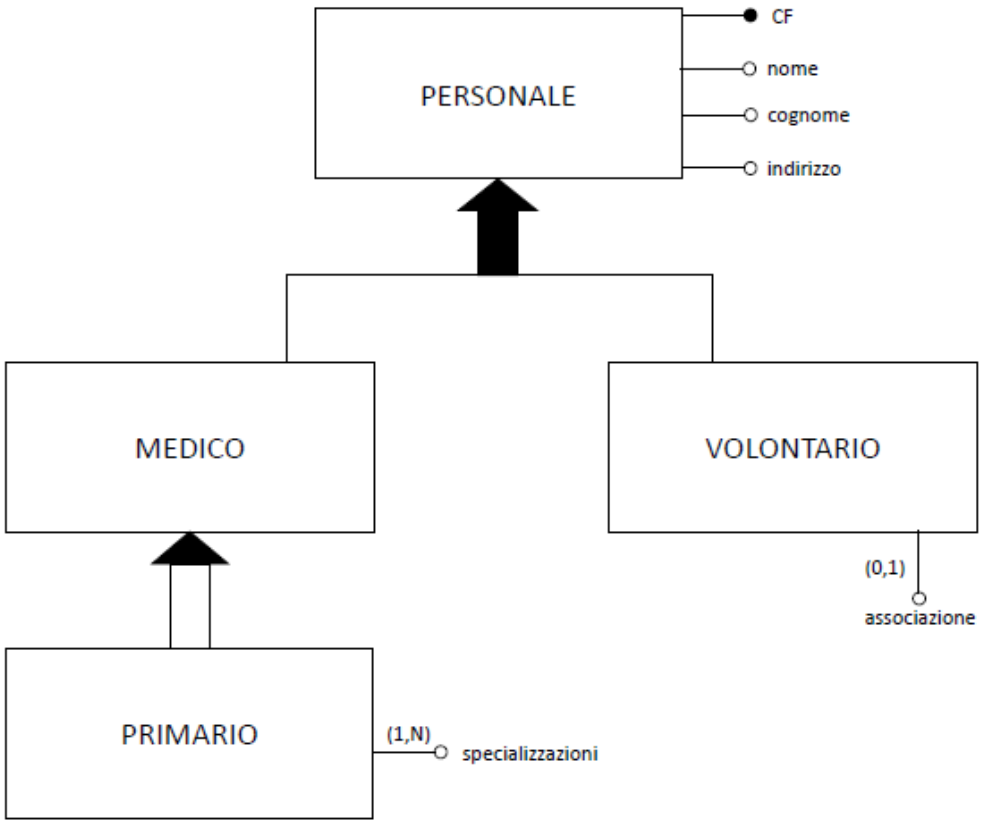
È stato individuato un costrutto di tipo "instance-of" per quanto riguarda le entità *ESAME* ed *ESAME EFFETTIVO*:



…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 del reparto, 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…

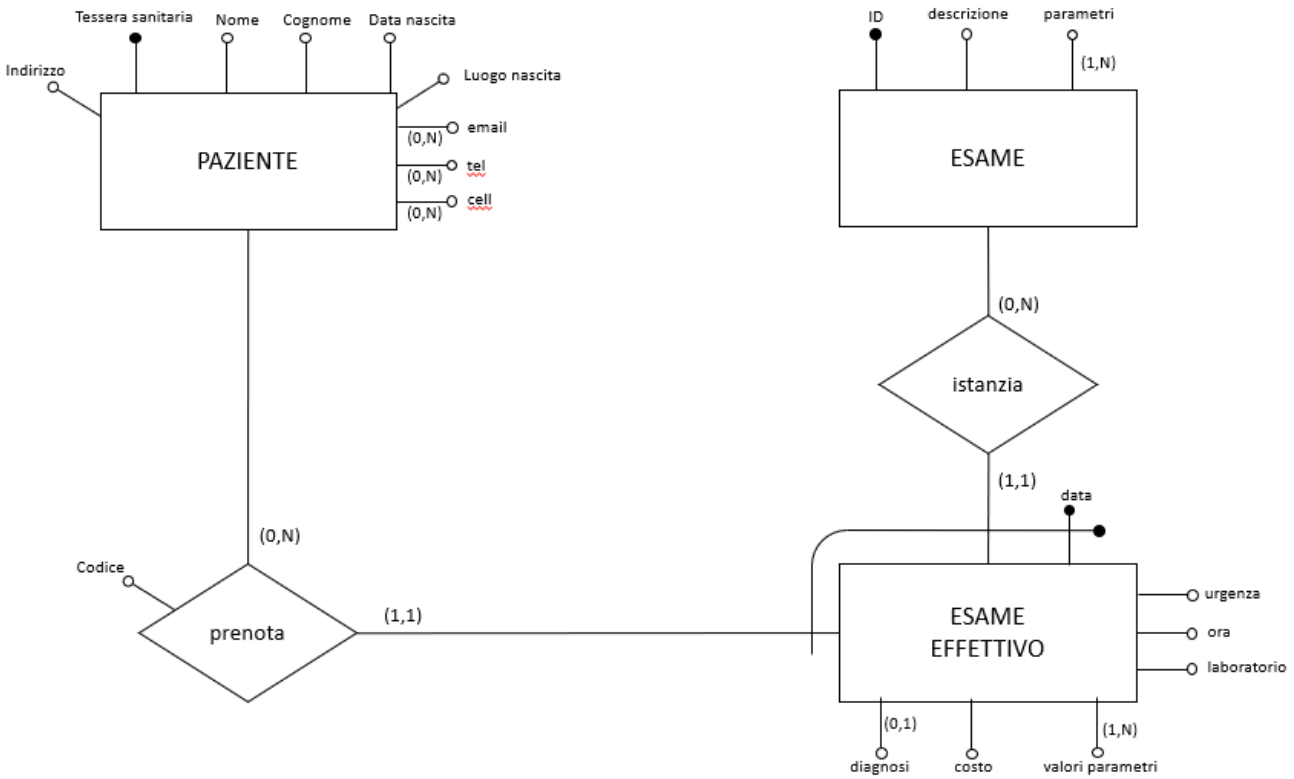
…I primari degli ospedali e dei laboratori vanno individuati all'interno del personale medico…



Si è proseguito nella costruzione dello schema E-R individuando, tramite un'analisi delle specifiche, le associazioni cui partecipano le diverse entità precedentemente rappresentate.
Integrando le diverse entità ed associazioni così ottenute, sono stati costruiti degli schemi più complessi ai precedenti e più vicini alla definizione dello schema E-R finale.

Per l'entità *ESAME EFFETTIVO* è stato individuato un identificatore esterno che comprende l'attributo "data" e le associazioni "istanzia" e "prenota", di modo che ogni paziente possa effettuare più prenotazioni dello stesso esame in date diverse e che lo stesso esame non possa essere ripetuto nella stessa data dallo stesso paziente, così come richiesto dalle specifiche, che sono di seguito riportate per completezza:

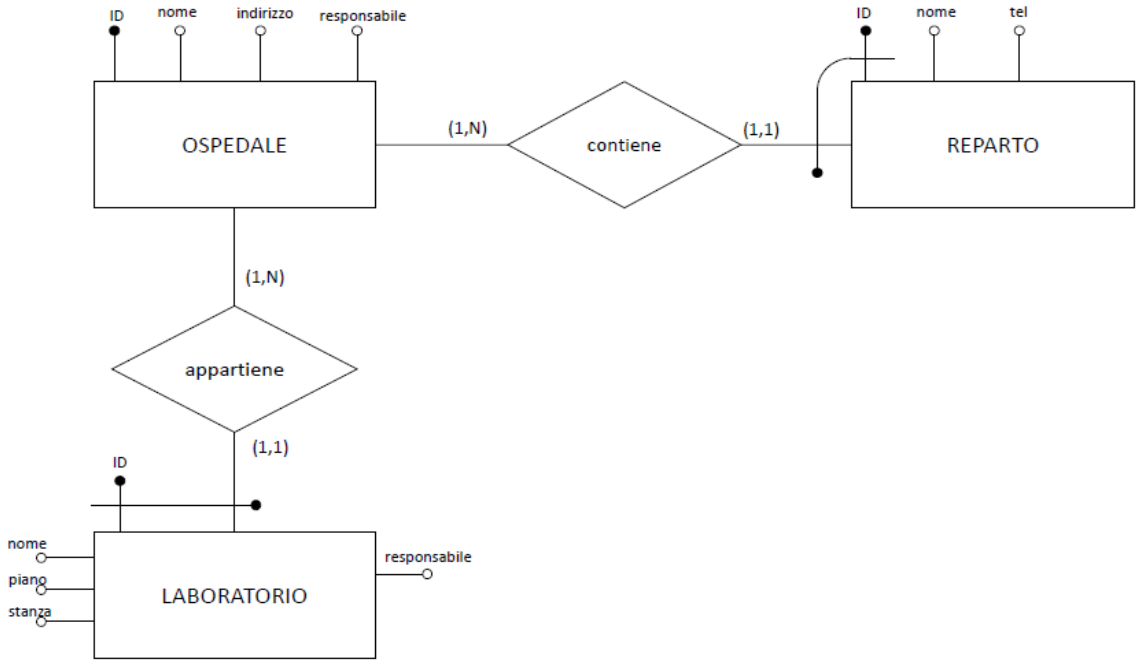
…Si tenga presente che ogni paziente può effettuare più prenotazioni dello stesso esame in date diverse. Si noti inoltre che lo stesso esame non può essere ripetuto nella stessa data dallo stesso paziente…



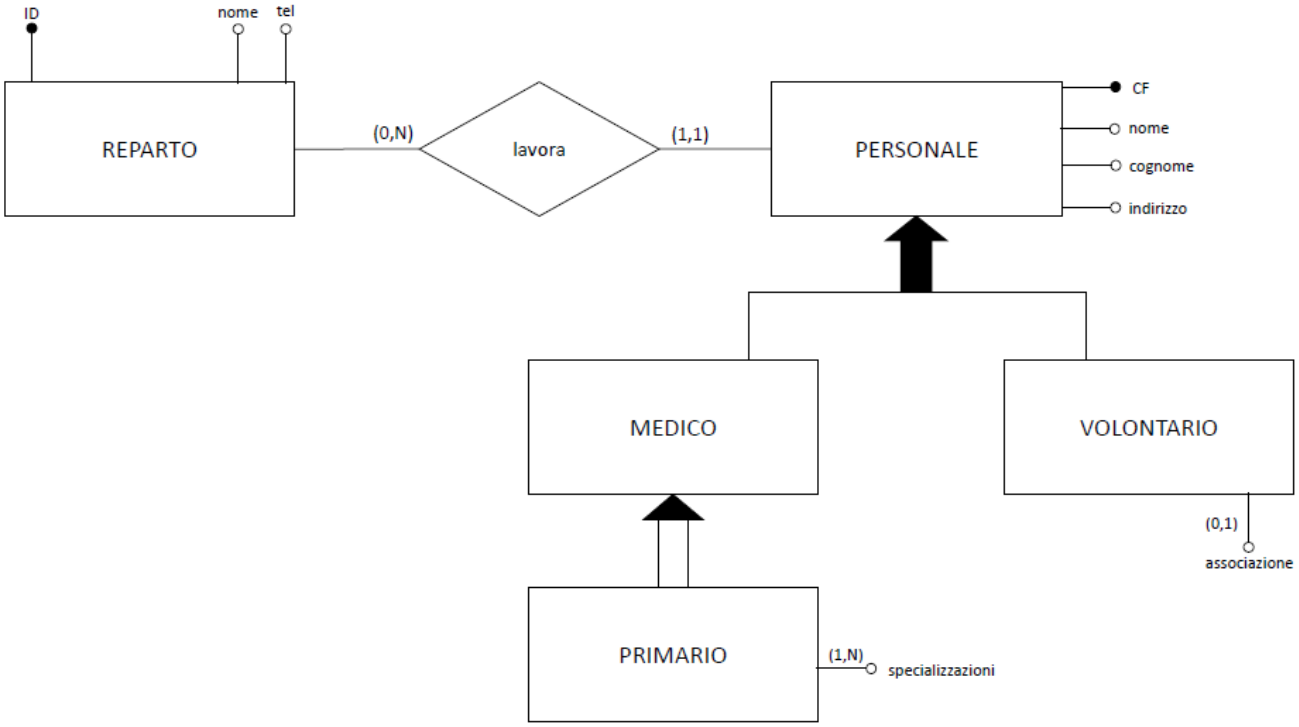
…Ogni ospedale è suddiviso in reparti identificati da un codice numerico univoco all'interno dell'ospedale di appartenenza…

…I laboratori che eseguono gli esami sono identificati da un codice univoco all'interno di un ospedale della ASL…

Essendo sia *REPARTO* che *LABORATORIO* identificati tramite un codice univoco all'interno dell'ospedale, hanno un identificatore esterno con le relazioni che coinvolgono *OSPEDALE*, seguendo il costrutto "part-of".



Unendo REPARTO con PERSONALE tramite la relazione “lavora”, si è ragionevolmente assunto che in un reparto possano lavorare un numero arbitrario di membri del personale del reparto e che un particolare membro del personale del reparto lavori unicamente in uno specifico reparto.



Integrazione finale

Nel passo di integrazione finale. Si sono sostituiti gli attributi “responsabile” delle entità *OSPEDALE* e *LABORATORIO*, nati da una prima analisi approssimativa delle specifiche, con due distinte associazioni (“responsabile ospedale” e “responsabile laboratorio”) con l’entità *PRIMARIO*.

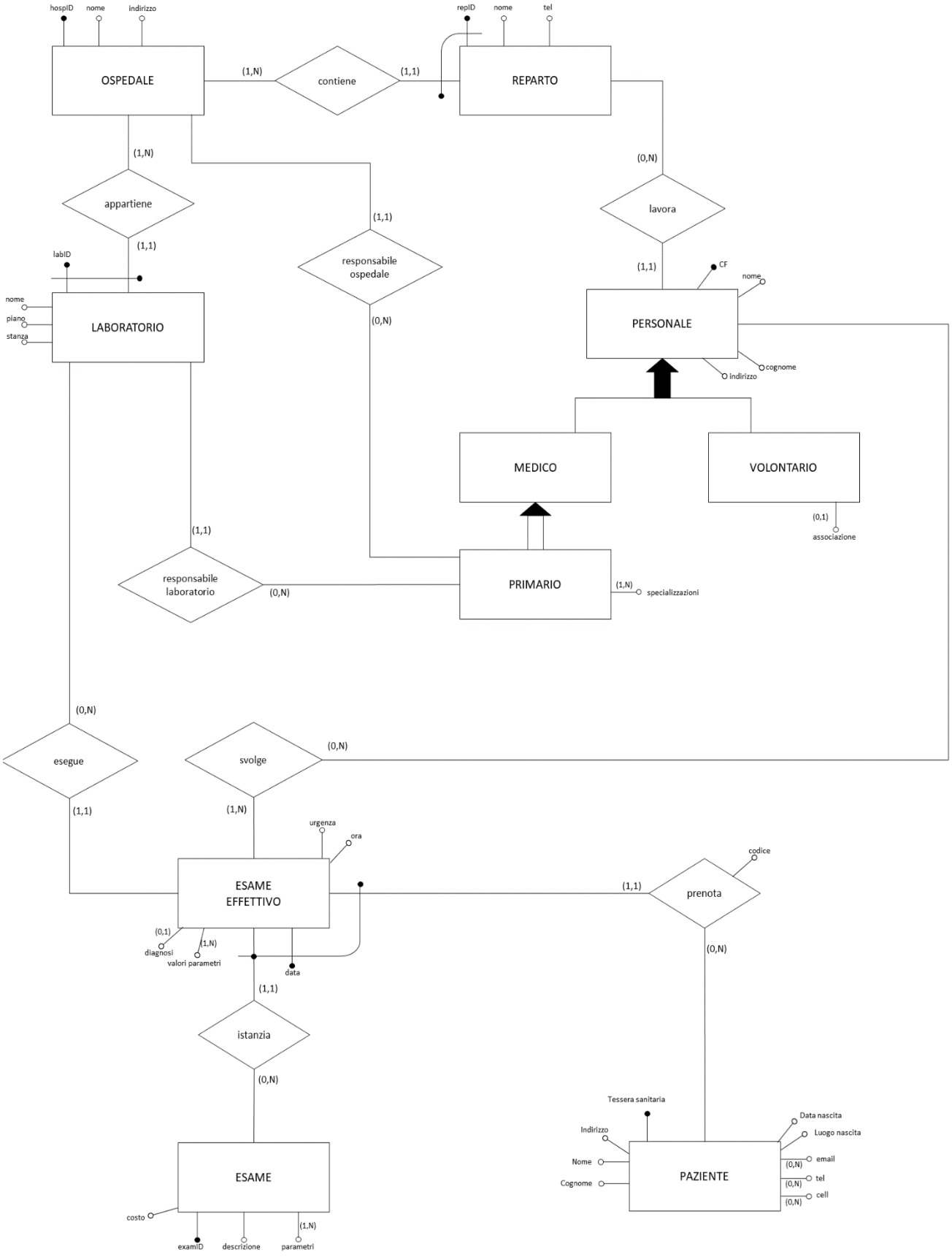
Analogamente, l’attributo “laboratorio” dell’entità *ESAME EFFETTIVO* è stato sostituito con l’associazione “esegue” tra le entità *ESAME EFFETTIVO* e *LABORATORIO*.

Inoltre, interpretando le specifiche, è stata introdotta l’associazione “svolge” tra le entità *PERSONALE* ed *ESAME EFFETTIVO*, per poter tener traccia degli esami svolti da ciascun membro del personale.

Per quanto riguarda conflitti sui nomi, si è preferito, in previsione futura, rinominare gli identificatori delle entità *OSPEDALE*, *LABORATORIO*, *REPARTO* ed *ESAME* rispettivamente come “hospID”, “labID”, “repID” ed “examID”.

Invece, per quanto riguarda conflitti strutturali, è stato necessario ridefinire le posizioni di alcune entità e di alcune relazioni così da poterne guadagnare in chiarezza e leggibilità dello schema E-R finale; inoltre è stato necessario anche spostare alcuni attributi e disporli in maniera tale da agevolare la connessione tra le entità e le associazioni aggiunte in questa fase di integrazione finale.

Il tutto ha portato alla realizzazione del seguente schema E-R:


hospID
nome
indirizzo
OSPEDALE
(1,N)
contiene
(1,1)
repID
nome
tel
REPARTO
(1,N)
appartiene
(1,1)
labID
nome
piano
stanza
LABORATORIO
(1,1)
responsabile ospedale
(0,N)
(0,N)
lavora
(1,1)
CF
nome
PERSONALE
cognome
indirizzo
MEDICO
VOLONTARIO
(0,1)
associazione
(1,1)
responsabile laboratorio
(0,N)
PRIMARIO
(1,N)
specializzazioni
(0,N)
esegue
(1,1)
svolge
(0,N)
(1,N)
urgenza
ora
ESAME EFFETTIVO
(0,1)
diagnosi
(1,N)
valori parametri
data
(1,1)
prenota
codice
(0,N)
(1,1)
istanzia
(0,N)
ESAME
costo
examID
descrizione
(1,N)
parametri
Tessera sanitaria
Indirizzo
Nome
Cognome
PAZIENTE
Data nascita
Luogo nascita
(0,N)
email
(0,N)
tel
(0,N)
cell

Regole aziendali

1) Ogni primario deve essere responsabile di almeno uno tra ospedali e laboratori;
2) Ogni esame effettivo deve avere tanti valori numerici quanti sono i parametri dell'entità esame;
3) Esami prenotati da pazienti differenti devono avere codici di prenotazione differenti.

Dizionario dei dati

Entità	Descrizione	Attributi	Identificatori
Ospedale	Ospedale di una ASL	hospID, nome, inidirizzo	hospID
Reparto	Reparto di un ospedale	repID, nome, tel	repID, Ospedale
Laboratorio	Laboratorio di un ospedale	labID, nome, piano, stanza	labID, Ospedale
Personale	Membro del personale di un reparto	CF, nome, cognome, indirizzo	CF
Medico	Particolare membro del personale di un reparto	CF, nome, cognome, indirizzo	CF
Volontario	Particolare membro del personale di un reparto, facente parte di un'associazione di volontariato	CF, nome, cognome, indirizzo, associazione*	CF
Primario	Medico che è responsabile di almeno un ospedale e/o di un laboratorio	CF, nome, cognome, indirizzo, specializzazioni	CF
Paziente	Persona che può prenotare e sottoporsi ad esami presso le strutture della ASL	tessera sanitaria, nome, cognome, data nascita, luogo nascita, indirizzo, email*, tel*, cell*	tessera sanitaria
Esame	Tipologia di esame disponibile presso le strutture della ASL	examID, descrizione, parametri, costo	examID
Esame effettivo	Esame di un tipo tra quelli disponibili presso la ASL, prenotato da un paziente ed eseguito in un laboratorio dal personale di un determinato reparto	data, ora, urgenza, valori parametri, diagnosi*	data, Esame, Paziente

Relationship	Descrizione	Componenti	Attributi
Contiene	Un ospedale contiene uno o più reparti	Ospedale, Reparto	
Appartiene	Un laboratorio appartiene ad uno ed un solo ospedale	Laboratorio, Ospedale	
Lavora	Un membro del personale lavora in un unico reparto	Personale, Reparto	
Responsabile ospedale	Ogni ospedale ha un primario come responsabile	Ospedale, Primario	
Responsabile laboratorio	Ogni laboratorio ha un primario come responsabile	Laboratorio, Primario	
Prenota	Un paziente prenota uno o più esami (concreti)	Paziente, Esame effettivo	codice
Istanzia	Un esame effettivo è un'istanza della tipologia di esame (astratto)	Esame, Esame effettivo	
Esegue	Un laboratorio esegue un esame effettivo	Laboratorio, Esame effettivo	
Svolge	Un membro del personale di un reparto svolge esami medici	Personale, Esame effettivo	

4. Progettazione logica

Volume dei dati

Concetto nello schema	Tipo[1]	Volume atteso
Paziente	E	1.500.000
Esame effettivo	E	4.500.000
Esame	E	500
Ospedale	E	15
Reparto	E	100
Laboratorio	E	300
Personale	E	10.000
Medico	E	2.000
Volontario	E	8.000
Primario	E	315
Prenota	R	4.500.000
Istanzia	R	4.500.000
Esegue	R	4.500.000
Svolge	R	4.500.000
Responsabile laboratorio	R	300
Responsabile ospedale	R	15
Lavora	R	10.000
Contiene	R	100
Appartiene	R	300

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

Tavola delle operazioni

Cod.	Descrizione	Frequenza attesa
1	Inserire un nuovo paziente	500 al giorno
2	Prenotare un esame	15.000 al giorno
3	Designare un responsabile di laboratorio	5 al mese
4	Generare dei report che mostrino quanti e quali esami ha svolto ciascun membro del personale in un anno e/o in un mese	10 al mese
5	Generare dei report che mostrino i risultati di un insieme di esami associati ad una prenotazione	10 a settimana
6	Generare dei report che mostrino lo storico di tutti gli esami svolti da un paziente dal momento della sua registrazione nel sistema	10 a settimana
7	Trovare tutti i dati di un membro del personale	10 a settimana
8	Visualizzare quali strutture mediche sono sotto la responsabilità di un dato primario	1 a settimana

Costo delle operazioni

Tavola degli accessi: operazione 1

Concetto	Costrutto	Accessi	Tipo
Paziente	Entity	1	S

La frequenza è di 500 volte al giorno; abbiamo 1 accesso in scrittura, che va considerato con un costo di 2 (doppio di quello in lettura). Dunque il costo per l'operazione 1 è:

500 * 2 = 1000 accessi/giorno

Tavola degli accessi: operazione 2

Concetto	Costrutto	Accessi	Tipo
Esame effettivo	Entity	1	S
Istanzia	Relationship	1	S

Prenota	Relationship	1	S
Esame	Entity	1	L
Paziente	Entity	1	L

La frequenza è di 1500 volte al giorno; abbiamo 2 accessi in lettura e 3 accessi in scrittura (che valgono il doppio di quelli in lettura). Dunque il costo per l'operazione 2 è:

1500 * (2 + 3*2) = 1500 * 8 = 12000 accessi/giorno

Tavola degli accessi: operazione 3

Concetto	Costrutto	Accessi	Tipo
Laboratorio	Entity	1	L
Primario	Entity	1	L
Responsabile laboratorio	Relationship	1	S

La frequenza è di 5 volte al mese; abbiamo 2 accessi in lettura e 1 accesso in scrittura. Dunque il costo per l'operazione 3 è:

5 * (2 + 1*2) = 10 accessi/mese

Tavola degli accessi: operazione 4

Concetto	Costrutto	Accessi	Tipo
Esame effettivo	Entity	450	L
Personale	Entity	1	L
Svolge	Relationship	450	L

La frequenza è di 10 volte al mese; abbiamo 450 + 1 + 450 = 901 accessi in lettura. Dunque il costo per l'operazione 4 è:

10 * 901 = 9010 accessi/mese

Si è considerato di generare un report che riguardasse un singolo membro del personale alla volta, desumendo dal volume dei dati che ogni membro del personale svolga in media 450 esami all'anno e che normalmente un esame viene svolto da un solo membro.

Tavola degli accessi: operazione 5

Concetto	Costrutto	Accessi	Tipo
Prenota	R	2	L
Esame effettivo	E	2	L

La frequenza è di 1 volta a settimana; abbiamo 2 + 2 = 4 accessi in lettura. Dunque il costo per l'operazione 5 è:

1 * 4 = 4 accessi/settimana

Tavola degli accessi: operazione 6

Concetto	Costrutto	Accessi	Tipo
Paziente	E	1	L
Esame effettivo	E	3	L
Prenota	R	3	L

La frequenza è di 10 volte a settimana; abbiamo 1 + 3 + 3 = 7 accessi in lettura. Dunque il costo per l'operazione 6 è:

10 * 7 = 70 accessi/settimana

Tavola degli accessi: operazione 7

Concetto	Costrutto	Accessi	Tipo
Personale	E	1	L

La frequenza è di 10 volte a settimana; abbiamo 1 accesso in lettura. Dunque il costo per l'operazione 7 è:

10 * 1 = 10 accessi/settimana

Tavola degli accessi: operazione 8

Concetto	Costrutto	Accessi	Tipo
Primario	E	1	L
Rsponsabile ospedale	R	1	L
Responsabile laboratorio	R	1	L

Ospedale	E	1	L
Laboratorio	E	1	L

La frequenza è di 1 volta a settimana; abbiamo 5 accessi in lettura. Dunque il costo per l'operazione 8 è:

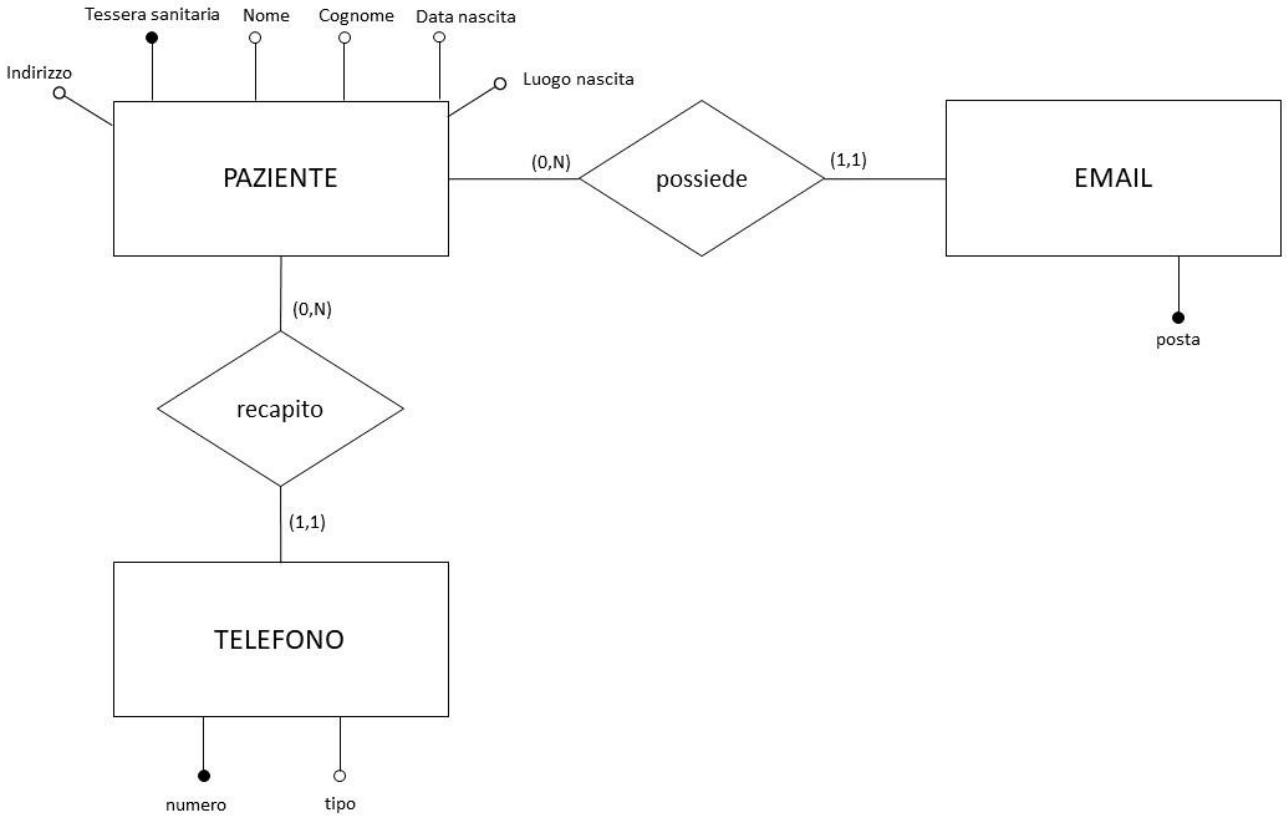
1 * 5 = 5 accessi/settimana

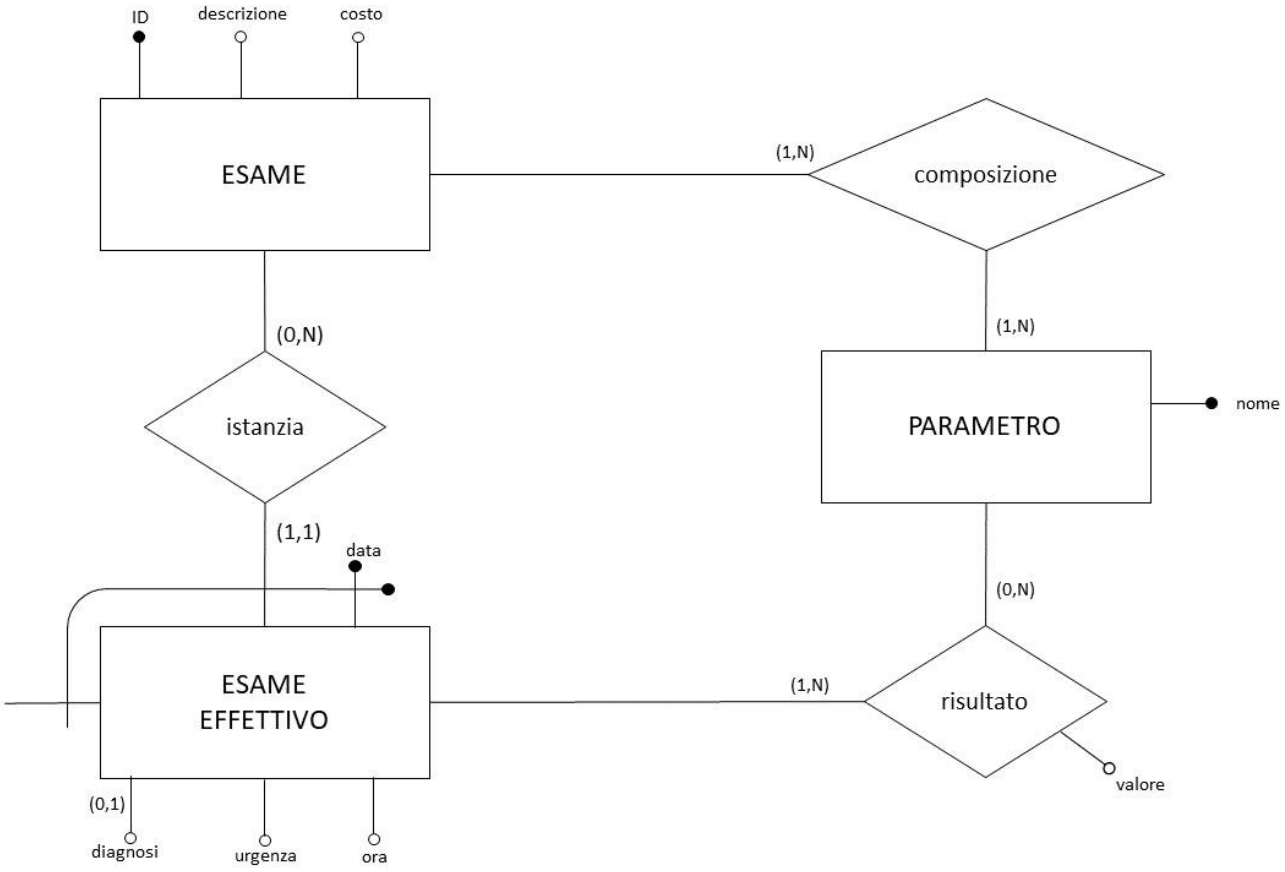
Ristrutturazione dello schema E-R

Lo schema E-R è stato ristrutturato innanzitutto rimuovendo gli attributi multivalore e traformandoli in entità. In particolare, tale operazione ha riguardato:

- gli attributi "email", "tel" e "cell" di *PAZIENTE;*
- l'attributo "specializzazioni" di *PRIMARIO*;
- l'attributo "parametri" di *ESAME* e l'attributo "valore parametri" di *ESAME EFFETTIVO*.

Tali modifiche si sono tradotte nei seguenti schemi parziali:





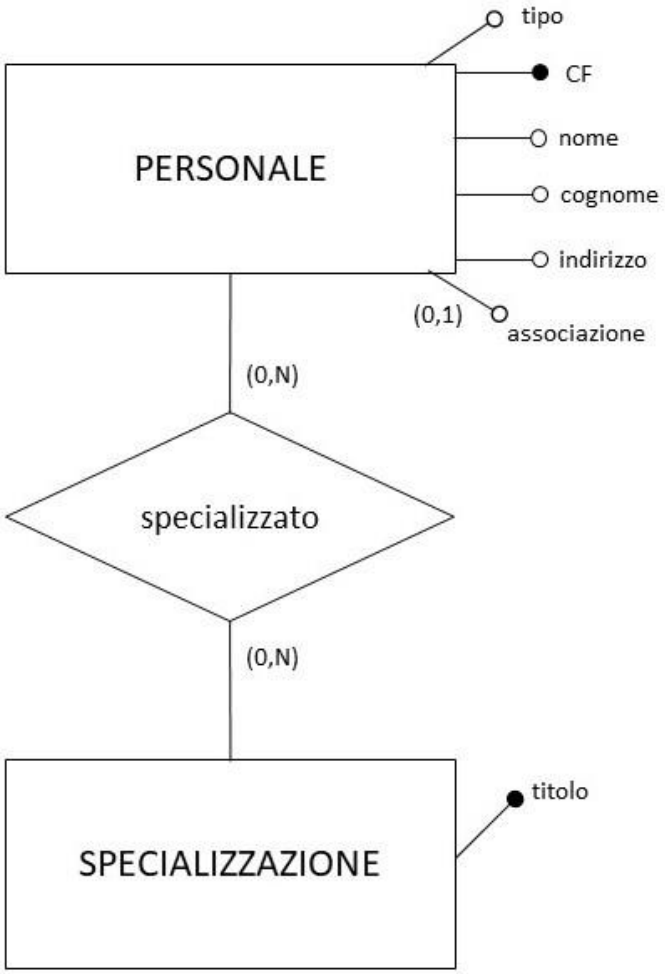
Successivamente, si è proseguito nella ristrutturazione dello schema E-R rimuovendo le generalizzazioni. Si è dapprima deciso di accorpare l'entità figlia *PRIMARIO* nell'entità padre *MEDICO,* introducendo l'attributo "tipo" per distinguere un semplice medico da un primario. Inoltre, l'entità *MEDICO* parteciperà all'associazione "specializzato" al posto di *PRIMARIO* con cardinalità (0,N) anziché (1,N); parteciperà anche alle associazioni "responsabile ospedale" e "responsabile laboratorio" al posto di *PRIMARIO*. Tale decisione è stata dettata dal fatto che le operazioni che coinvolgono specificatamente un primario, come ad esempio l'operazione 3, hanno bassa frequenza e un basso numero di accessi.

In seguito, si è deciso di accorpare le entità figlie *MEDICO* e *VOLONTARIO* nell'entità padre *PERSONALE*, in quanto gli accessi alle entità figlie sono quasi sempre contestuali agli accessi all'entità padre (operazioni 4 e7) tranne che per l'operazione 8, la quale però viene eseguita con una frequenza molto bassa. Dunque l'attributo "associazione" dell'entità *VOLONTARIO* diventa attributo opzionale dell'entità *PERSONALE,* il che comporta un piccolo spreco di memoria dovuto a valori

nulli dell'attributo, ma bisogna anche considerare che i medici costituiscono circa il 20% del personale, dunque l'introduzione di valori nulli è non significativa (2000 valori nulli; se per una stringa vengono riservati 32 byte, si ha uno spreco di 64 Kbyte, che è uno spreco di memoria accettabile a fronte di un guadagno in prestazioni); analogamente, l'attributo "tipo" dell'entità *MEDICO*, diventa un attributo di *PERSONALE* e distinguerà tra volontario, medico e primario. Inoltre, l'entità *PERSONALE* parteciperà all'associazione "specializzato" al posto di *MEDICO* con cardinalità (0,N); parteciperà anche alle associazioni "responsabile ospedale" e "responsabile laboratorio" al posto di *MEDICO*.

Bisogna ovviamente aggiungere i seguenti vincoli:

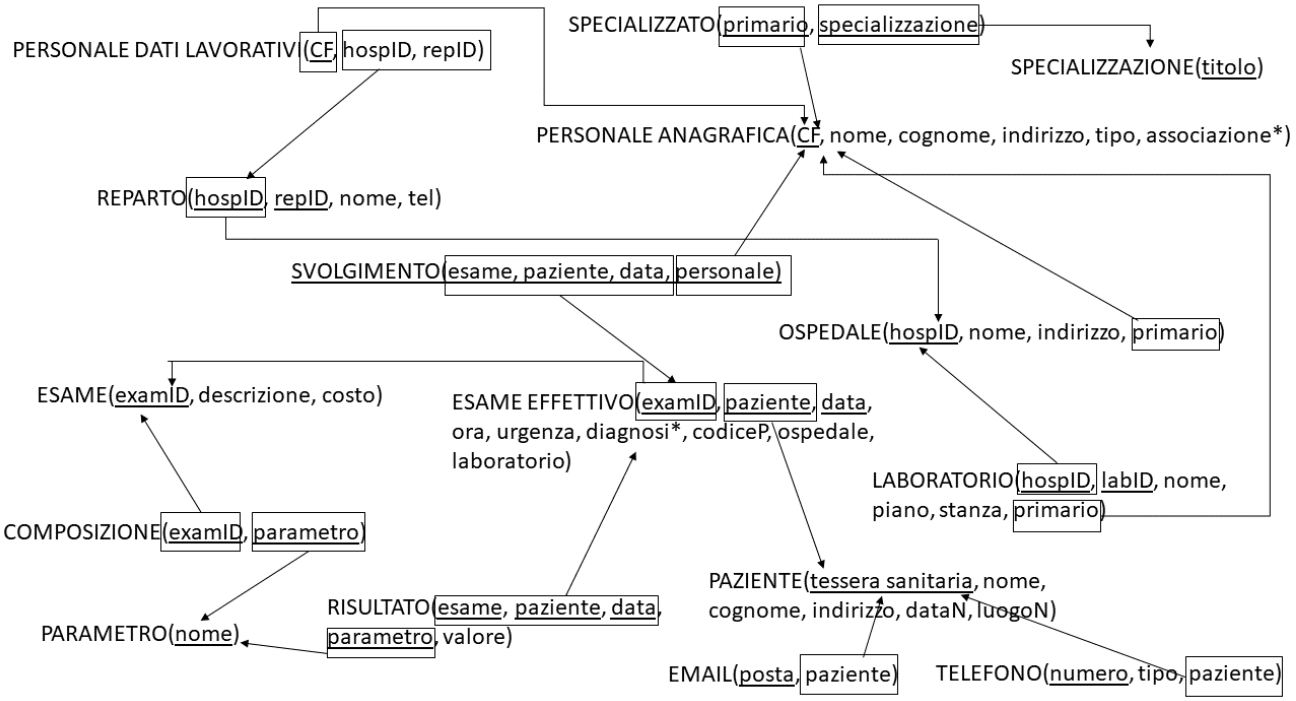
- Solo un membro del personale di tipo primario può essere responsabile di un ospedale e/o di un laboratorio;
- Solo per un membro del personale di tipo primario bisogna indicare l'elenco delle specializzazioni;
- Solo per un membro del personale di tipo volontario è possibile indicare l'associazione di appartenenza.



Inoltre, al fine di evitare anomalie di inserimento, l'entità *PERSONALE* è stata partizionata verticalmente in due entità *PERSONALE ANAGGRAFICA* e *PERSONALE DATI LAVORATIVI*; la prima ha gli attributi CF, nome, cognome, indirizzo, tipo, associazione; la seconda ha invece gli attributi repID e hospID ed è debole rispetto alla prima.

Trasformazione di attributi e identificatori

Traduzione di entità e associazioni



Normalizzazione del modello relazionale

- La relazione OSPEDALE è in forma normale di Boyce e Codd (in seguito BCNF), in quanto presenta solo dipendenze funzionali banali tra la chiave "hospID" e gli altri attributi "nome", "cognome" e "primario";
- La relazione REPARTO è in BCNF, in quanto anch'essa presenta solo dipendenze funzionali banali tra la chiave composta "repID" "hospID" e gli attributi "nome" e "tel";
- La relazione LABORATORIO è in BCNF, poiché anche in questo caso sussistono solo dipendenze funzionali banali tra la chiave "labID" "hospID" e gli altri attributi "nome", "piano", "stanza" e "primario";
- La relazione PERSONALE ANAGRAFICA è anch'essa in BCNF, in quanto anche in questo caso la chiave "CF" è primo membro di una dipendenza funzionale verso tutti gli altri attributi della relazione;

- La relazione PERSONALE DATI LAVORATIVI è anch'essa in BCNF, in quanto anche in questo caso la chiave "CF" è primo membro di una dipendenza funzionale verso gli attributi "hospID" e "repID";
- La relazione SPECIALIZZAZIONE ha come attributo la sola chiave primaria "titolo" ed è ovviamente in BCNF;
- La relazione SPECIALIZZATO ha anch'essa come unici attributi "primario" e "specializzazione" che costituiscono la chiave primaria, ed è quindi in BCNF;
- La relazione SVOLGIMENTO è in BCNF poiché ha come unici attributi "esame" e "personale" che ne costituiscono la chiave;
- La relazione ESAME è in BCNF, in quanto le uniche dipendenze funzionali presenti sono quelle banali tra la chiave primaria "examID" e gli attributi "descrizione" e "costo";
- La relazione PARAMETRO ha come unico attributo la sola chiave primaria "nome" ed è quindi in BCNF;
- La relazione COMPOSIZIONE ha come attributi "examID" e "parametro" che formano la sua chiave primaria, dunque è in BCNF;
- La relazione ESAME EFFETTIVO ha le seguenti dipendenze funzionali:

  $$\text{paziente examID data} \rightarrow A$$

  dove $A \in$ {"ora", "urgenza", "diagnosi", "laboratorio", "ospedale", "codiceP"}.
  Tuttavia, gli attributi "paziente", "examID" e "data" formano la chiave primaria per la relazione ESAME EFFETTIVO, la quale risulta dunque essere in BCNF;
- La relazione RISULTATO ha solo la dipendenza funzionale banale tra la chiave formata dagli attributi "esame", "paziente", "data", "parametro" e l'attributo "valore", dunque è in BCNF;
- La relazione PAZIENTE ha anch'essa solo dipendenze funzionali tra la chiave "tessera sanitaria" e gli attributi "nome", "cognome", "indirizzo", "dataN", "luogoN"; quindi è in BCNF;
- La relazione TELEFONO ha solo dipendenze funzionali banali tra la chiave "numero" e gli attributi "tipo" e "paziente", ed è quindi in BCNF;
- La relazione EMAIL ha solo dipendenze funzionali banali tra la chiave "posta" e l'attributo "paziente", ed è dunque anch'essa in BCNF.

Ovviamente, relazioni che sono in BCNF soddisfano anche la 1NF, la 2NF e la 3NF. In particolare: soddisfano la 1NF poiché ogni relazione ha una chiave primaria e non ha attributi composti né attributi che si ripetono; soddisfano la 2NF poiché nessuna relazione, in particolare quelle che hanno una chiave formata da più attributi, presenta una dipendenza parziale; infine, soddisfano la 3NF in quanto nessuna relazione presenta una dipendenza transitiva.

5. Progettazione fisica

Utenti e privilegi

All'interno dell'applicazione sono stati previsti 3 tipi di utenti: Login, PersonaleCUP, Amministratore, con diversi privilegi di accesso alle tabelle e privilegi di esecuzioni di operazioni e stored procedures. Tali privilegi sono stati assegnati cercando di rimanere il più fedele possibile al POLP (Principle Of Least Principles) e contemporaneamente di garantire a ciascun utente il necessario per poter operare al meglio sulla base di dati.

Utente Login: tale utente è l'utente con il quale si accede preliminarmente alla base di dati e ha l'unico privilegio di tipo EXECUTE sulla stored procedure 'login' che permette appunto di effettuare una lettura nella tabella 'Users' per poter consentire l'accesso come uno degli altri due tipi di utenti previsti o rifiutare l'accesso in caso di credenziali errate.

Utente PersonaleCUP: questo tipo di utente sarà utilizzato dai membri del personale del CUP e ha privilegi di tipo EXECUTE sulle seguenti stored procedures:

- 'inserisci_paziente': permette di inserire un nuovo paziente nel sistema; tale privilegio è stato assegnato in quanto la gestione dei pazienti è responsabilità dei membri del personale del CUP;
- 'aggiungi_email', 'aggiungi_recapito_tel': tali operazioni riguardano sempre la gestione dei pazienti e dei loro dati;
- 'prenota_esame': permette di prenotare un esame effettivo a nome di un paziente in una precisa data; le prenotazioni sono gestite dai membri del personale del CUP;
- 'scrivi_diagnosi', 'inserisci_risultato_esame': permettono di aggiungere informazioni riguardanti gli esami svolti, precedentemente prenotati;
- 'search_by_codiceP': permette di cercare tutti gli esami che hanno lo stesso codice di prenotazione;
- 'report_paziente': consente di visualizzare un report sullo storico degli esami svolti da un paziente dal suo inserimento nel sistema;
- 'assegna_personale_ad_esame': consente di registrare quali membri del personale medico hanno svolto o dovranno svolgere un dato esame; tale privilegio è stato assegnato all'utente PersonaleCUP in quanto è lui ad avere conoscenza degli esami prenotati;

- ‘list_esami_disponibili’: permette di visualizzare quali tipologie di esami medici sono disponibili per una prenotazione;
- ‘list_ospedali’, ‘list_laboratori’, ‘list_reparti’: consentono di visualizzare le strutture mediche gestite dalla ASL, per poter assegnare lo svolgimento degli esami ad una di esse;
- ‘select_personale_by_hosp’, ‘select_peronale_by_rep’: permettono di visualizzare i membri del personale medico che afferiscono ad un certo ospedale o reparto;
- ‘list_patients’, ‘search_patient’: permettono rispettivamente di listare tutti i pazienti registrati nel sistema e di cercare un determinato paziente tramite la sua tessera sanitaria;
- ‘search_recapiti’: consente di visualizzare i recapiti (email e/o numeri di telefono) associati ad un determinato paziente;
- ‘report_risultati_prenotazione’: permette di visualizzare i risultati di un insieme di esami associati ad un codice di prenotazione;
- ‘cancella_prenotazione’: permette di cancellare la prenotazione di un esame se esso non è ancora stato eseguito.

Utente Amministratore: questo tipo di utente sarà utilizzato dagli amministratori del sistema e ha privilegi di tipo EXECUTE sulle seguenti stored procedures:

- ‘registration’: permette di creare un nuovo utente con username e password;
- ‘crea_tipologia_esame’, ‘crea_parametro’, ‘add_parametro_ad_esame’: consentono la gestione degli esami disponibili presso la ASL, unicamente a capo degli amministratori;
- ‘add_personale_anagrafica’, ‘add_personale_lavorativo’, ‘aggiungi_tipo_specializzazione’, ‘assegna_specializzazione’: consentono di inserire nuovi membri del personale medico, e i dati loro associati;
- ‘inserisci_ospedale’, ‘inserisci_laboratorio’, ‘inserisci_reparto’: consentono la gestione delle strutture mediche e la designazione dei primari di quest’ultime, unicamente a capo degli amministratori;
- ‘select_personale_by_hosp’, ‘select_peronale_by_rep’: consente di visualizzare i membri del personale afferenti ad un dato ospedale o reparto;
- ‘report_personale_mese’, ‘report_personale_anno’: consentono di generare dei report che, dato un membro del personale medico, indichino quanti e quali esami ha eseguito nell’ultimo mese o anno.

Strutture di memorizzazione

Tabella <Esame_effettivo>		
Attributo	**Tipo di dato**	**Attributi[2]**
examID	INT	PK, NN
paziente	VARCHAR(45)	PK, NN
data	DATE	PK, NN
ora	TIME	NN
urgenza	ENUM(‘si’, ‘no)	NN
diagnosi	VARCHAR(256)	
codiceP	INT	NN
laboratorio	INT	NN
ospedale	INT	NN

Tabella <Paziente>		
Attributo	**Tipo di dato**	**Attributi**
tessera_sanitaria	VARCHAR(45)	PK, NN
nome	VARCHAR(45)	NN
cognome	VARCHAR(45)	NN
indirizzo	VARCHAR(100)	NN
dataN	DATE	NN
luogoN	VARCHAR(45)	NN

Tabella <Esame>		
Attributo	**Tipo di dato**	**Attributi**
examID	INT	PK, NN
descrizione	VARCHAR(45)	NN

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

costo	FLOAT	NN

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

Tabella <Composizione>		
Attributo	**Tipo di dato**	**Attributi**
examID	INT	PK, NN
parametro	VARCHAR(45)	PK, NN

Tabella <Risultato>		
Attributo	**Tipo di dato**	**Attributi**
esame	INT	PK, NN
paziente	VARCHAR(45)	PK, NN
data	DATE	PK, NN
parametro	VARCHAR(45)	PK, NN
valore	FLOAT	NN

Tabella <Telefono>		
Attributo	**Tipo di dato**	**Attributi**
numero	VARCHAR(45)	PK, NN
tipo	ENUM(‘cellulare’, ‘fisso’)	NN
paziente	VARCHAR(45)	NN

Tabella <Email>		
Attributo	**Tipo di dato**	**Attributi**
posta	VARCHAR(45)	PK, NN
paziente	VARCHAR(45)	NN

Tabella <Ospedale>		
Attributo	**Tipo di dato**	**Attributi**
hospID	INT	PK, NN, AI
nome	VARCHAR(45)	NN
indirizzo	VARCHAR(100)	NN
primario	VARCHAR(45)	NN

Tabella <Laboratorio>		
Attributo	**Tipo di dato**	**Attributi**
hospID	INT	PK, NN
labID	INT	PK, NN
nome	VARCHAR(45)	NN
piano	INT	NN
stanza	VARCHAR(45)	NN
primario	VARCHAR(45)	NN

Tabella <Reparto>		
Attributo	**Tipo di dato**	**Attributi**
hospID	INT	PK, NN
repID	INT	PK, NN
nome	VARCHAR(45)	NN
tel	VARCHAR(45)	NN

Tabella <Personale_anagrafica>		
Attributo	**Tipo di dato**	**Attributi**
CF	VARCHAR(45)	PK, NN
nome	VARCHAR(45)	NN
cognome	VARCHAR(45)	NN

indirizzo	VARCHAR(100)	NN
tipo	ENUM('medico', 'primario', 'volontario')	NN
associazione	VARCHAR(45)	

Tabella <Personale_dati_lavorativi>		
Attributo	**Tipo di dato**	**Attributi**
personale	VARCHAR(45)	PK, NN
ospedale	INT	NN
reparto	INT	NN

Tabella <Svolgimento>		
Attributo	**Tipo di dato**	**Attributi**
esame	INT	PK, NN
paziente	VARCHAR(45)	PK, NN
data	VARCHAR(45)	PK, NN
personale	VARCHAR(45)	NN

Tabella <Specializzazione>		
Attributo	**Tipo di dato**	**Attributi**
titolo	VARCHAR(100)	PK, NN

Tabella <Specializzato>		
Attributo	**Tipo di dato**	**Attributi**
primario	VARCHAR(45)	PK, NN
specializzazione	VARCHAR(100)	PK, NN

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

password	CHAR(32)	NN
ruolo	ENUM('personaleCUP', 'amministratore')	NN

Indici

Sono stati inseriti indici in ogni tabella di tipo PRIMARY e di tipo INDEX in particolare per implementare le Foreign Key. In questo modo si ottimizzano enormemente le prestazioni delle queries, specialmente quelle con una struttura più articolata e che toccano più dati; inoltre si permette anche al DBMS di controllare più rapidamente che i vincoli di chiave e di Foreign Key vengano rispettati.

Tabella <Esame_effettivo>	
Indice <PRIMARY>	**Tipo[3]:**
examID, paziente, data	PR

Tabella <Esame_effettivo>	
Indice <fk_Esame_effettivo_Esame1_idx>	**Tipo:**
examID	IDX

Tabella <Esame_effettivo>	
Indice <fk_Esame_effettivo_Paziente1_idx>	**Tipo:**
paziente	IDX

Tabella <Esame_effettivo>	
Indice <fk_Esame_effettivo_Laboratorio1_idx>	**Tipo:**
ospedale, laboratorio	IDX

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

Tabella <Paziente>	
Indice <PRIMARY>	**Tipo:**
tessera_sanitaria	PR

Tabella <Esame>	
Indice <PRIMARY>	**Tipo:**
examID	PR

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

Tabella <Composizione>	
Indice <PRIMARY>	**Tipo:**
examID, parametron	PR

Tabella <Composizione>	
Indice <fk_Parametro_has_Esame_Esame1_idx>	**Tipo:**
examID	IDX

Tabella <Composizione>	
Indice <fk_Parametro_has_Esame_Esame1_idx >	**Tipo:**
parametro	IDX

Tabella <Risultato>	
Indice <PRIMARY>	**Tipo:**
esame, paziente, data, parametro	PR

Tabella <Risultato>	
Indice <fk_Esame_effettivo_has_Parametro_Parametro1_idx >	**Tipo:**
parametro	IDX

Tabella <Risultato>	
Indice <fk_Esame_effettivo_has_Parametro_Esame_effettivo1_idx>	**Tipo:**
esame, paziente, data	IDX

Tabella <Telefono>	
Indice <PRIMARY>	**Tipo:**
numero	PR

Tabella <Telefono>	
Indice <fk_Telefono_Paziente_idx>	**Tipo:**
paziente	IDX

Tabella <Email>	
Indice <PRIMARY>	**Tipo:**
posta	PR

Tabella <Email>	
Indice <fk_Email_Paziente1_idx>	**Tipo:**
paziente	IDX

Tabella <Ospedale>	
Indice <PRIMARY>	**Tipo:**

hospID	PR

Tabella <Ospedale>	
Indice <fk_Ospedale_Personale1_idx>	**Tipo:**
primario	IDX

Tabella <Laboratorio>	
Indice <PRIMARY>	**Tipo:**
hospID, labID	PR

Tabella <Laboratorio>	
Indice <fk_Laboratorio_Ospedale1_idx>	**Tipo:**
hospID	IDX

Tabella <Laboratorio>	
Indice <fk_Laboratorio_Personale1_idx>	**Tipo:**
primario	IDX

Tabella <Reparto>	
Indice <PRIMARY>	**Tipo:**
hospID, repID	PR

Tabella <Reparto>	
Indice <fk_Reparto_Ospedale1_idx>	**Tipo:**
hospID	IDX

Tabella <Personale_anagrafica>	
Indice <PRIMARY>	**Tipo:**
CF	PR

Tabella <Personale_dati_lavorativi>	
Indice <PRIMARY>	**Tipo:**
personale	PR

Tabella <Personale_dati_lavorativi>	
Indice <fk_Personale_dati_lavorativi_Reparto1_idx>	**Tipo:**
ospedale, reparto	IDX

Tabella <Svolgimento>	
Indice <PRIMARY>	**Tipo:**
esame, paziente, data	PR

Tabella <Svolgimento>	
Indice <fk_Svolgimento_Personale_anagrafica1_idx>	**Tipo:**
personale	IDX

Tabella <Specializzazione>	
Indice <PRIMARY>	**Tipo:**
titolo	PR

Tabella <Specializzato>	
Indice <PRIMARY>	**Tipo:**
primario, specializzazione	PR

Tabella <Specializzato>	
Indice <fk_Specializzazione_has_Personale_Personale1_idx>	**Tipo:**
primario	IDX

Tabella <Specializzato>	
Indice <fk_Specializzazione_has_Personale_Specializzazione1_idx>	**Tipo:**
specializzazione	IDX

Tabella <Users>	
Indice <PRIMARY>	**Tipo:**
username	PR

Trigger

```
DELIMITER $$


USE `ASL_db`$$

DROP TRIGGER IF EXISTS `ASL_db`.`Esame_effettivo_BEFORE_INSERT` $$

USE `ASL_db`$$

CREATE DEFINER = CURRENT_USER TRIGGER
`ASL_db`.`Esame_effettivo_BEFORE_INSERT` BEFORE INSERT ON `Esame_effettivo` FOR
EACH ROW

BEGIN

	if NEW.data < CURDATE() then

		signal sqlstate '45007' set message_text = "A reservation in the past is not allowed!";

	end if;

END$$

USE `ASL_db`$$

DROP TRIGGER IF EXISTS `ASL_db`.`Esame_effettivo_BEFORE_INSERT_1` $$

USE `ASL_db`$$

CREATE DEFINER = CURRENT_USER TRIGGER
`ASL_db`.`Esame_effettivo_BEFORE_INSERT_1` BEFORE INSERT ON `Esame_effettivo` FOR
EACH ROW

BEGIN

	declare var_paziente varchar(45);


   select paziente

   from Esame_effettivo

   where codiceP = NEW.codiceP
```

```
    into var_paziente;


    if var_paziente is not null and var_paziente <> NEW.paziente then

            signal sqlstate '45012' set message_text="Il codice di prenotazione è già stato
utilizzato per un altro paziente";

        end if;

END$$


USE `ASL_db`$$

DROP TRIGGER IF EXISTS `ASL_db`.`Esame_effettivo_BEFORE_DELETE` $$

USE `ASL_db`$$

CREATE DEFINER = CURRENT_USER TRIGGER
`ASL_db`.`Esame_effettivo_BEFORE_DELETE` BEFORE DELETE ON `Esame_effettivo` FOR
EACH ROW

BEGIN

        if OLD.data < CURDATE() then

            signal sqlstate '45011' set message_text = "L'esame è già stato eseguito e non può
essere rimosso";

        end if;

END$$




USE `ASL_db`$$

DROP TRIGGER IF EXISTS `ASL_db`.`Svolgimento_BEFORE_INSERT` $$

USE `ASL_db`$$

CREATE DEFINER = CURRENT_USER TRIGGER `ASL_db`.`Svolgimento_BEFORE_INSERT`
BEFORE INSERT ON `Svolgimento` FOR EACH ROW
```

```
BEGIN
	if NEW.data < current_date() then
		signal sqlstate '45008' set message_text = "Exam already executed";
	end if;
END$$



USE `ASL_db`$$
DROP TRIGGER IF EXISTS `ASL_db`.`Specializzato_BEFORE_INSERT` $$
USE `ASL_db`$$
CREATE DEFINER = CURRENT_USER TRIGGER
`ASL_db`.`Specializzato_BEFORE_INSERT` BEFORE INSERT ON `Specializzato` FOR EACH
ROW
BEGIN
	declare var_tipo enum('medico', 'primario', 'volontario');

   select tipo
   from Personale_anagrafica as P
   where P.CF = NEW.primario
   into var_tipo;

   if var_tipo <> 'primario' then
		signal sqlstate '45005' set message_text = "Il membro del personale indicato non è un
primario.";
	end if;
END$$
```

```
USE `ASL_db`$$

DROP TRIGGER IF EXISTS `ASL_db`.`Risultato_BEFORE_INSERT` $$

USE `ASL_db`$$

CREATE DEFINER = CURRENT_USER TRIGGER `ASL_db`.`Risultato_BEFORE_INSERT`
BEFORE INSERT ON `Risultato` FOR EACH ROW

BEGIN

        declare counter int;


    select count(*)

    from Composizione C

    where NEW.esame = C.examID and NEW.parametro = C.parametro

    into counter;


    if counter <> 1 then

                signal sqlstate '45010' set message_text = "Parametro non previsto per tale esame.";

        end if;


END$$

DELIMITER ;
```

Eventi

Non sono stati previsti eventi.

Viste

Non sono state previste delle viste.

Stored Procedures e transazioni

```
-- -----------------------------------------------------
-- procedure inserisci_ospedale
-- -----------------------------------------------------

USE `ASL_db`;
DROP procedure IF EXISTS `ASL_db`.`inserisci_ospedale`;

DELIMITER $$
USE `ASL_db`$$
CREATE PROCEDURE `inserisci_ospedale` (IN var_nome varchar(45), IN var_indirizzo varchar(100), IN var_primario VARCHAR(45), OUT var_hospID int)
BEGIN
	declare var_tipo enum('medico', 'primario', 'volontario');

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

	set transaction isolation level READ COMMITTED;
  start transaction;

    select P.tipo
    from Personale_anagrafica P
    where P.CF = var_primario
    into var_tipo;
```

```
	if var_tipo = 'volontario' then
			signal sqlstate '45006' set message_text = "Un volontario non può essere primario";
		elseif var_tipo = 'medico' then
			update Personale_anagrafica
		set tipo = 'primario'
		where CF = var_primario;
		end if;
	-- if var_tipo = 'primario' do nothing

		insert into Ospedale (nome, indirizzo, primario)
		values (var_nome, var_indirizzo, var_primario);

		set var_hospID = last_insert_id();
	commit;
END$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure inserisci_reparto
-- -----------------------------------------------------

USE `ASL_db`;
DROP procedure IF EXISTS `ASL_db`.`inserisci_reparto`;

DELIMITER $$
USE `ASL_db`$$
CREATE PROCEDURE `inserisci_reparto` (IN var_hospID int, IN var_repID int, IN var_nome varchar(45), IN var_tel varchar(45))
```

```
BEGIN
        insert into Reparto (hospID, repID, nome, tel)
    values (var_hospID, var_repID, var_nome, var_tel);

END$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure inserisci_laboratorio
-- -----------------------------------------------------

USE `ASL_db`;
DROP procedure IF EXISTS `ASL_db`.`inserisci_laboratorio`;

DELIMITER $$
USE `ASL_db`$$
CREATE PROCEDURE `inserisci_laboratorio` (IN var_ospedale int, IN var_laboratorio int, IN var_nome varchar(45), IN var_piano int, IN var_stanza varchar(45),IN var_primario varchar(45))
BEGIN
        declare var_tipo enum('medico', 'primario', 'volontario');

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

    set transaction isolation level read committed;
```

```
   start transaction;

      select tipo
      from Personale_anagrafica
      where CF = var_primario
      into var_tipo;

      if var_tipo = 'volontario' then
                  signal sqlstate '45006' set message_text = "Un volontario non può essere un primario.";
            elseif var_tipo = 'medico' then
                  update Personale_anagrafica
         set tipo = 'primario'
         where CF = var_primario;
            end if;

      -- if var_tipo = 'primario' do nothing;

            insert into Laboratorio(hospID, labID,nome, piano, stanza, primario)
      values (var_ospedale, var_laboratorio, var_nome, var_piano, var_stanza, var_primario);

         commit;
END$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure inserisci_paziente
-- -----------------------------------------------------
```

```
USE `ASL_db`;
DROP procedure IF EXISTS `ASL_db`.`inserisci_paziente`;

DELIMITER $$
USE `ASL_db`$$
CREATE PROCEDURE `inserisci_paziente` (IN var_tessera_sanitaria varchar(45), IN var_nome varchar(45), IN var_cognome varchar(45), IN var_indirizzo varchar(100), IN var_dataN date, IN var_luogoN varchar(45))
BEGIN
	insert into Paziente (tessera_sanitaria, nome, cognome, indirizzo, dataN, luogoN)
  values (var_tessera_sanitaria, var_nome, var_cognome, var_indirizzo, var_dataN, var_luogoN);
END$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure crea_parametro
-- -----------------------------------------------------

USE `ASL_db`;
DROP procedure IF EXISTS `ASL_db`.`crea_parametro`;

DELIMITER $$
USE `ASL_db`$$
CREATE PROCEDURE `crea_parametro` (IN var_nome varchar(45))
BEGIN
	insert into Parametro (nome) values (var_nome);
END$$
```

```
DELIMITER ;

-- -----------------------------------------------------
-- procedure crea_tipologia_esame
-- -----------------------------------------------------

USE `ASL_db`;
DROP procedure IF EXISTS `ASL_db`.`crea_tipologia_esame`;

DELIMITER $$
USE `ASL_db`$$
CREATE PROCEDURE `crea_tipologia_esame` (IN var_descrizione varchar(100), IN var_costo float, OUT var_examID int)
BEGIN
	insert into Esame(descrizione, costo) values (var_descrizione, var_costo);
   set var_examID = last_insert_id();
END$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure login
-- -----------------------------------------------------

USE `ASL_db`;
DROP procedure IF EXISTS `ASL_db`.`login`;

DELIMITER $$
USE `ASL_db`$$
```

```
CREATE PROCEDURE `login` (IN var_username VARCHAR(45), IN var_password CHAR(32), OUT var_role INT)
BEGIN
	declare var_role_name ENUM('amministratore','personaleCUP');

   SELECT ruolo
   FROM Users
   WHERE username = var_username AND password = md5(var_password)
   INTO var_role_name;

   -- protocol between DBMS and client
   IF var_role_name = 'amministratore' then
		set var_role = 1;
	ELSEIF var_role_name = 'personaleCUP' then
		set var_role = 2;
	ELSE
		set var_role = 3;
	END IF;
END$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure registration
-- -----------------------------------------------------

USE `ASL_db`;
DROP procedure IF EXISTS `ASL_db`.`registration`;
```

```
DELIMITER $$

USE `ASL_db`$$

CREATE PROCEDURE `registration` (IN var_username VARCHAR(45), IN var_password VARCHAR(32), IN var_ruolo ENUM('amministratore', 'personaleCUP'))

BEGIN

        INSERT INTO Users(username, password, ruolo)

    VALUES (var_username, md5(var_password), var_ruolo);

END$$


DELIMITER ;


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

-- procedure aggiungi_email

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


USE `ASL_db`;

DROP procedure IF EXISTS `ASL_db`.`aggiungi_email`;


DELIMITER $$

USE `ASL_db`$$

CREATE PROCEDURE `aggiungi_email` (IN var_posta VARCHAR(45), IN var_paziente VARCHAR(45))

BEGIN

        INSERT INTO Email(posta, paziente)

    VALUES (var_posta, var_paziente);


END$$


DELIMITER ;
```

```
-- -----------------------------------------------------
-- procedure aggiungi_recapito_tel
-- -----------------------------------------------------

USE `ASL_db`;
DROP procedure IF EXISTS `ASL_db`.`aggiungi_recapito_tel`;

DELIMITER $$
USE `ASL_db`$$
CREATE PROCEDURE `aggiungi_recapito_tel` (IN var_numero varchar(45), IN var_tipo ENUM('cellulare', 'fisso'), IN var_paziente VARCHAR(45))
BEGIN
	insert into Telefono (numero, tipo, paziente)
   values (var_numero, var_tipo, var_paziente);
END$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure add_personale_anagrafica
-- -----------------------------------------------------

USE `ASL_db`;
DROP procedure IF EXISTS `ASL_db`.`add_personale_anagrafica`;

DELIMITER $$
USE `ASL_db`$$
CREATE PROCEDURE `add_personale_anagrafica` (IN var_CF VARCHAR(45), IN var_nome VARCHAR(45), IN var_cognome VARCHAR(45), IN var_indirizzo VARCHAR(100), IN var_tipo ENUM('medico','primario','volontario'), IN var_associazione VARCHAR(45))
```

```
BEGIN
        insert into Personale_anagrafica(CF, nome, cognome, indirizzo, tipo, associazione)
    values (var_CF, var_nome, var_cognome, var_indirizzo, var_tipo, var_associazione);
END$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure add_personale_lavorativo
-- -----------------------------------------------------

USE `ASL_db`;
DROP procedure IF EXISTS `ASL_db`.`add_personale_lavorativo`;

DELIMITER $$
USE `ASL_db`$$
CREATE PROCEDURE `add_personale_lavorativo` (IN var_CF VARCHAR(45), IN var_ospedale INT, IN var_reparto INT)
BEGIN
        insert into Personale_dati_lavorativi (personale, ospedale, reparto)
    values (var_CF, var_ospedale, var_reparto);
END$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure aggiungi_tipo_specializzazione
-- -----------------------------------------------------
```

```
USE `ASL_db`;
DROP procedure IF EXISTS `ASL_db`.`aggiungi_tipo_specializzazione`;

DELIMITER $$
USE `ASL_db`$$
CREATE PROCEDURE `aggiungi_tipo_specializzazione` (IN var_titolo VARCHAR(100))
BEGIN
        insert into Specializzazione (titolo)
   values (var_titolo);
END$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure assegna_specializzazione
-- -----------------------------------------------------

USE `ASL_db`;
DROP procedure IF EXISTS `ASL_db`.`assegna_specializzazione`;

DELIMITER $$
USE `ASL_db`$$
CREATE PROCEDURE `assegna_specializzazione` (IN var_primario VARCHAR(45), IN var_specializzazione VARCHAR(100))
BEGIN
        insert into Specializzato (primario, specializzazione)
   values (var_primario, var_specializzazione);
END$$
```

```
DELIMITER ;

-- -----------------------------------------------------
-- procedure add_parametro_ad_esame
-- -----------------------------------------------------

USE `ASL_db`;
DROP procedure IF EXISTS `ASL_db`.`add_parametro_ad_esame`;

DELIMITER $$
USE `ASL_db`$$
CREATE PROCEDURE `add_parametro_ad_esame` (IN var_esame int, IN var_parametro varchar(45))
BEGIN
	insert into Composizione (examID, parametro)
   values (var_esame, var_parametro);
END$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure select_personale_by_hosp
-- -----------------------------------------------------

USE `ASL_db`;
DROP procedure IF EXISTS `ASL_db`.`select_personale_by_hosp`;

DELIMITER $$
USE `ASL_db`$$
```

```
CREATE PROCEDURE `select_personale_by_hosp` (IN var_ospedale int)
BEGIN

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

        set transaction isolation level READ COMMITTED;
   start transaction;

                select P2.ospedale as ospedale, P2.reparto as reparto, CF, nome, cognome, indirizzo, tipo as ruolo, associazione
       from Personale_anagrafica as P1 join Personale_dati_lavorativi as P2 on P1.CF = P2.personale
                where P2.ospedale = var_ospedale;

   commit;

END$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure select_personale_by_rep
-- -----------------------------------------------------

USE `ASL_db`;
DROP procedure IF EXISTS `ASL_db`.`select_personale_by_rep`;
```

```
DELIMITER $$
USE `ASL_db`$$
CREATE PROCEDURE `select_personale_by_rep` (IN var_ospedale int, IN var_reparto int)
BEGIN

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

        set transaction isolation level READ COMMITTED;
   start transaction;

                select P2.ospedale as ospedale, P2.reparto as reparto, CF, nome, cognome, indirizzo, tipo as ruolo, associazione
       from Personale_anagrafica as P1 join Personale_dati_lavorativi as P2 on P1.CF = P2.personale
                where P2.ospedale = var_ospedale and P2.reparto = var_reparto;

   commit;

END$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure prenota_esame
-- -----------------------------------------------------
```

```
USE `ASL_db`;

DROP procedure IF EXISTS `ASL_db`.`prenota_esame`;


DELIMITER $$

USE `ASL_db`$$

CREATE PROCEDURE `prenota_esame` (IN var_esame int, IN var_paziente varchar(45), IN var_data date, IN var_ora time, IN var_urgenza enum('si', 'no'), IN var_diagnosi varchar(256), IN var_codiceP int, IN var_lab int, IN var_hosp int)

BEGIN


    insert into Esame_effettivo(examID, paziente, data, ora, urgenza, diagnosi, codiceP, laboratorio, ospedale)

  values (var_esame, var_paziente, var_data, var_ora, var_urgenza, var_diagnosi, var_codiceP, var_lab, var_hosp);


END$$


DELIMITER ;


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

-- procedure scrivi_diagnosi

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


USE `ASL_db`;

DROP procedure IF EXISTS `ASL_db`.`scrivi_diagnosi`;


DELIMITER $$

USE `ASL_db`$$

CREATE PROCEDURE `scrivi_diagnosi` (IN var_esame int, IN var_paziente varchar(45), IN var_data date, IN var_diagnosi varchar(256))
```

```
BEGIN

        update Esame_effettivo
    set diagnosi = var_diagnosi
    where examId = var_esame and paziente = var_paziente and data = var_data;

END$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure search_by_codiceP
-- -----------------------------------------------------

USE `ASL_db`;
DROP procedure IF EXISTS `ASL_db`.`search_by_codiceP`;

DELIMITER $$
USE `ASL_db`$$
CREATE PROCEDURE `search_by_codiceP` (IN var_codiceP int)
BEGIN

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

    set transaction isolation level read committed;
```

```
    start transaction;

        select paziente, descrizione as esame, data, ora, urgenza, diagnosi, ospedale, laboratorio
        from Esame_effettivo E1 join Esame E2 on E1.examID = E2.examID
        where E1.codiceP = var_codiceP;

    commit;
END$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure inserisci_risultato_esame
-- -----------------------------------------------------

USE `ASL_db`;
DROP procedure IF EXISTS `ASL_db`.`inserisci_risultato_esame`;

DELIMITER $$
USE `ASL_db`$$
CREATE PROCEDURE `inserisci_risultato_esame` (IN var_esame int, IN var_paziente varchar(45), IN var_data date, IN var_parametro varchar(45), IN var_valore float)
BEGIN
        insert into Risultato(esame, paziente, data, parametro, valore)
    values (var_esame, var_paziente, var_data, var_parametro, var_valore);
END$$

DELIMITER ;
```

```
-- -----------------------------------------------------
-- procedure report_paziente
-- -----------------------------------------------------

USE `ASL_db`;
DROP procedure IF EXISTS `ASL_db`.`report_paziente`;

DELIMITER $$
USE `ASL_db`$$
CREATE PROCEDURE `report_paziente` (IN var_paziente varchar(45))
BEGIN

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

   set transaction isolation level READ COMMITTED;
   start transaction;

                select codiceP as 'P. code', descrizione as esame, data, urgenza, diagnosi, laboratorio, ospedale, costo
      from Esame_effettivo E join Esame E1 on E.examID = E1.examID
      where E.paziente = var_paziente and E.data <= current_date();

      select codiceP as 'P. code', descrizione as esame, data, urgenza, diagnosi, laboratorio, ospedale, costo
      from Esame_effettivo E join Esame E1 on E.examID = E1.examID
      where E.paziente = var_paziente and E.data > current_date();
```

```
    commit;

END$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure assegna_personale_ad_esame
-- -----------------------------------------------------

USE `ASL_db`;
DROP procedure IF EXISTS `ASL_db`.`assegna_personale_ad_esame`;

DELIMITER $$
USE `ASL_db`$$
CREATE PROCEDURE `assegna_personale_ad_esame` (IN var_esame int, IN var_paziente varchar(45), IN var_data date, IN var_personale varchar(45))
BEGIN

        insert into Svolgimento(esame, paziente, data, personale)
    values (var_esame, var_paziente, var_data, var_personale);

END$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure report_personale_mese
```

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

USE `ASL_db`;
DROP procedure IF EXISTS `ASL_db`.`report_personale_mese`;

DELIMITER $$
USE `ASL_db`$$
CREATE PROCEDURE `report_personale_mese` (IN var_personale varchar(45), OUT numero_esami int)
BEGIN

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

   -- consecutive reads and range lock needed
   set transaction isolation level SERIALIZABLE;
   start transaction;

	 -- number of executed exams
      select count(*)
      from Svolgimento S
      where S.personale = var_personale and S.data <= current_date() and S.data >= DATE_SUB(current_date(), interval 1 MONTH)
      into numero_esami;

      -- info about executed exams
```

```
    select E1.descrizione as esame, E.paziente as paziente, E.data as data, E.urgenza as urgenza, E.ospedale as ospedale, E.laboratorio as laboratorio

    from Svolgimento S join Personale_anagrafica P on S.personale = P.CF

                    join Esame_effettivo E on S.esame = E.examID and S.data = E.data and S.paziente = E.paziente

        join Esame E1 on E.examID = E1.examID

                where S.personale = var_personale and S.data <= current_date() and S.data >= DATE_SUB(current_date(), interval 1 MONTH);




  commit;

END$$


DELIMITER ;


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

-- procedure report_personale_anno

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


USE `ASL_db`;

DROP procedure IF EXISTS `ASL_db`.`report_personale_anno`;


DELIMITER $$

USE `ASL_db`$$

CREATE PROCEDURE `report_personale_anno` (IN var_personale varchar(45), OUT numero_esami int)

BEGIN


    declare var_qta_esami int;

  declare exit handler for sqlexception
```

```
begin
        rollback;
    resignal;
      end;

set transaction isolation level SERIALIZABLE;
start transaction;

    -- number of executed exams
    select count(*)
    from Svolgimento S
    where S.personale = var_personale and S.data <= current_date() and S.data >= DATE_SUB(current_date(), interval 1 YEAR)
    into numero_esami;

    -- info about executed exams
    select E1.descrizione as esame, E.paziente as paziente, E.data as data, E.urgenza as urgenza, E.ospedale as ospedale, E.laboratorio as laboratorio
    from Svolgimento S join Personale_anagrafica P on S.personale = P.CF
                    join Esame_effettivo E on S.esame = E.examID and S.data = E.data and S.paziente = E.paziente
      join Esame E1 on E.examID = E1.examID
            where S.personale = var_personale and S.data <= current_date() and S.data >= DATE_SUB(current_date(), interval 1 YEAR);


commit;
END$$

DELIMITER ;
```

```
-- -----------------------------------------------------
-- procedure list_esami_disponibili
-- -----------------------------------------------------

USE `ASL_db`;
DROP procedure IF EXISTS `ASL_db`.`list_esami_disponibili`;

DELIMITER $$
USE `ASL_db`$$
CREATE PROCEDURE `list_esami_disponibili` ()
BEGIN
	declare exit handler for sqlexception
  begin
		rollback;
    resignal;
	end;

  set transaction isolation level READ COMMITTED;
  start transaction;
		select examID as ID, descrizione, costo
		from Esame;
	commit;
END$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure list_ospedali
```

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

USE `ASL_db`;
DROP procedure IF EXISTS `ASL_db`.`list_ospedali`;

DELIMITER $$
USE `ASL_db`$$
CREATE PROCEDURE `list_ospedali` ()
BEGIN
	declare exit handler for sqlexception
  begin
		rollback;
    resignal;
	end;

  set transaction isolation level READ COMMITTED;
  start transaction;
		select hospID as ID, O.nome as nome, O.indirizzo as indirizzo, P.nome as 'nome primario', P.cognome as 'cognome primario'
		from Ospedale O join Personale_anagrafica P on O.primario = P.CF
    order by hospID asc;
	commit;
END$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure list_laboratori
-- -----------------------------------------------------
```

```
USE `ASL_db`;
DROP procedure IF EXISTS `ASL_db`.`list_laboratori`;

DELIMITER $$
USE `ASL_db`$$
CREATE PROCEDURE `list_laboratori` ()
BEGIN
	declare exit handler for sqlexception
   begin
		rollback;
      resignal;
	end;

   set transaction isolation level READ COMMITTED;
   start transaction;
		select hospID as ospedale, labID as ID, Laboratorio.nome as nome, piano, stanza, P.nome as 'nome primario', P.cognome as 'cognome primario'
		from Laboratorio join Personale_anagrafica P on primario = P.CF
      order by hospID asc;
	commit;
END$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure list_reparti
-- -----------------------------------------------------
```

```
USE `ASL_db`;
DROP procedure IF EXISTS `ASL_db`.`list_reparti`;

DELIMITER $$
USE `ASL_db`$$
CREATE PROCEDURE `list_reparti` (IN var_ospedale int)
BEGIN
        declare exit handler for sqlexception
   begin
                rollback;
      resignal;
        end;

   set transaction isolation level READ COMMITTED;
   start transaction;
                select hospID as ospedale, repID as ID, nome, tel as telefono
                from Reparto
      where hospID = var_ospedale
      order by hospID asc;
        commit;
END$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure list_patients
-- -----------------------------------------------------

USE `ASL_db`;
```

```
DROP procedure IF EXISTS `ASL_db`.`list_patients`;

DELIMITER $$
USE `ASL_db`$$
CREATE PROCEDURE `list_patients` ()
BEGIN
        declare exit handler for sqlexception
   begin
                rollback;
      resignal;
        end;

   set transaction isolation level READ COMMITTED;

      select tessera_sanitaria as 'tessera sanitaria', nome, cognome, indirizzo, dataN as 'D.O.B.', luogoN as 'luogo nascita'
      from Paziente
      order by tessera_sanitaria;

   commit;
END$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure search_patient
-- -----------------------------------------------------

USE `ASL_db`;
```

```
DROP procedure IF EXISTS `ASL_db`.`search_patient`;

DELIMITER $$
USE `ASL_db`$$
CREATE PROCEDURE `search_patient` (IN var_tessera_sanitaria varchar(45))
BEGIN

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

  set transaction isolation level READ COMMITTED;
  start transaction;

		select tessera_sanitaria as 'tessera sanitaria', nome, cognome, indirizzo, dataN as 'D.O.B.', luogoN as 'luogo nascita'
    from Paziente
    where tessera_sanitaria = var_tessera_sanitaria;

  commit;
END$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure search_recapiti
-- -----------------------------------------------------
```

```
USE `ASL_db`;
DROP procedure IF EXISTS `ASL_db`.`search_recapiti`;

DELIMITER $$
USE `ASL_db`$$
CREATE PROCEDURE `search_recapiti` (IN var_paziente varchar(45))
BEGIN

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

   set transaction isolation level READ COMMITTED;
   start transaction;

      select paziente, numero as recapito
      from Telefono
      where paziente = var_paziente
      union
      select paziente, posta as recapito
      from Email
      where paziente = var_paziente;
   commit;

END$$
```

```
DELIMITER ;

-- -----------------------------------------------------
-- procedure search_strutture_primario
-- -----------------------------------------------------

USE `ASL_db`;
DROP procedure IF EXISTS `ASL_db`.`search_strutture_primario`;

DELIMITER $$
USE `ASL_db`$$
CREATE PROCEDURE `search_strutture_primario` (IN var_primario varchar(45))
BEGIN

	declare var_ruolo enum('medico', 'primario', 'volontario');
	declare exit handler for sqlexception
  begin
		rollback;
    resignal;
	end;

  set transaction isolation level READ COMMITTED;
  start transaction;

		select tipo
    from Personale_anagrafica
    where CF = var_primario
    into var_ruolo;
```

```
        if var_ruolo <> 'primario' then
                        signal sqlstate '45009' set message_text = "Il membro del personale non è un primario";
                end if;


        select hospID as ospedale, nome, indirizzo
        from Ospedale
        where primario = var_primario;

        select hospID as ospedale, labID as laboratorio, nome, piano, stanza
        from Laboratorio
        where primario = var_primario;

    commit;
END$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure report_risultati_prenotazione
-- -----------------------------------------------------

USE `ASL_db`;
DROP procedure IF EXISTS `ASL_db`.`report_risultati_prenotazione`;

DELIMITER $$
USE `ASL_db`$$
CREATE PROCEDURE `report_risultati_prenotazione` (IN var_codiceP int)
```

```
BEGIN

        declare done int default false;
    declare var_examID int;
    declare var_paziente varchar(45);
    declare var_data date;

        declare cur cursor for
                select examID, paziente, data
       from Esame_effettivo
       where codiceP = var_codiceP;

        declare continue handler for not found
                set done = true;

    -- both handler must be declared after cursor
    declare exit handler for sqlexception
        begin
                rollback;
       resignal;
        end;

    set transaction isolation level READ COMMITTED;
    start transaction;
                open cur;

                read_loop: loop
                        fetch cur into var_examID, var_paziente, var_data;
                        if done then
```

```
                leave read_loop;
            end if;

            select E.descrizione as esame, R.paziente as paziente, R.data as data, C.parametro as parametro, R.valore as valore
            from Esame E join Risultato R on E.examID = R.esame
                join Composizione C on R.esame = C.examID and R.parametro = C.parametro
            where R.esame = var_examID and R.paziente = var_paziente and R.data = var_data;

        end loop;
        close cur;

    commit;
END$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure cancella_prenotazione
-- -----------------------------------------------------

USE `ASL_db`;
DROP procedure IF EXISTS `ASL_db`.`cancella_prenotazione`;

DELIMITER $$
USE `ASL_db`$$
CREATE PROCEDURE `cancella_prenotazione` (IN var_esame int, IN var_paziente varchar(45), IN var_data date)
BEGIN
```

```
        delete from Esame_effettivo
    where examID = var_esame and paziente = var_paziente and data = var_data;

END$$

DELIMITER ;
```

Sono state inserite delle transazioni nelle seguenti stored procedures:

- ‘inserisci_ospedale’ e ‘inserisci_laboratorio’: transazione inserita per verificare se il primario indicato è effettivamente un primario o un medico; nel secondo caso, il medico viene promosso a primario; invece, se il primario indicato risultasse essere un volontario, la transazione andrà in rollback e sarà abortita. Livello di isolamento READ COMMITTED, in quanto si vogliono leggere dati di membri del personale effettivamente inseriti nella base di dati;
- ‘select_personale_by_hosp’ e ‘select_personale_by_rep’: livello di isolamento READ COMMITTED per essere coerenti nelle letture di membri del personale effettivamente presenti nella base di dati;
- ‘select_by_codiceP’: livello di isolamento READ COMMITTED per leggere esami effettivamente prenotati;
- ‘report_paziente’: livello di isolamento READ COMMITTED in quanto, anche se si fanno due SELECT sulla stessa tabella, i loro risultati sono 2 insiemi di record certamente disgiunti e dunque non si può soffrire di anomalie del tipo unrepeatable read;
- ‘report_personale_mese’ e ‘report_personale_anno’: livello di isolamento SERIALIZABLE, in quanto si hanno due SELECT consecutive sulla stessa tabella ed inoltre la prima delle due conta il numero di righe risultato della seconda; dunque, per essere coerenti e non essere soggetti ad anomalie di tipo phantom read è necessario tale livello di isolamento;
- ‘list_esami_disponibili’, ‘list_ospedali’, ‘list_laboratori’, ‘list_reparti’, ‘list_patients’: livello di isolamento READ COMMITTED per garantire la lettura di dati coerenti;
- ‘search_patient’, ‘search_recapiti’, ‘search_strutture_primario’: livello di isolamento READ COMIITTED per garantire la lettura di dati coerenti;
- ‘report_risultati_prenotazione’: livello di isolamento READ COMMITTED, in quanto, nonostante il numero di SELECT sia molteplice, non si esegue mai la stessa SELECT sulla stessa tabella (le condizioni nella clausola WHERE cambiano ad ogni iterazione).

Appendice: Implementazione

Codice SQL per instanziare il database

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

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

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



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

-- Schema ASL_db

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

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

USE `ASL_db` ;



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

-- Table `ASL_db`.`Paziente`

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

DROP TABLE IF EXISTS `ASL_db`.`Paziente` ;


CREATE TABLE IF NOT EXISTS `ASL_db`.`Paziente` (

  `tessera_sanitaria` VARCHAR(45) NOT NULL,

  `nome` VARCHAR(45) NOT NULL,

  `cognome` VARCHAR(45) NOT NULL,

  `indirizzo` VARCHAR(100) NOT NULL,

  `dataN` DATE NOT NULL,
```

```
  `luogoN` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`tessera_sanitaria`))
ENGINE = InnoDB;



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

CREATE TABLE IF NOT EXISTS `ASL_db`.`Telefono` (
  `numero` VARCHAR(45) NOT NULL,
  `tipo` ENUM('cellulare', 'fisso') NOT NULL,
  `paziente` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`numero`),
  INDEX `fk_Telefono_Paziente_idx` (`paziente` ASC) VISIBLE,
  CONSTRAINT `fk_Telefono_Paziente`
    FOREIGN KEY (`paziente`)
    REFERENCES `ASL_db`.`Paziente` (`tessera_sanitaria`)
    ON DELETE CASCADE
    ON UPDATE CASCADE)
ENGINE = InnoDB;



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

```
-- Table `ASL_db`.`Email`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `ASL_db`.`Email` ;

CREATE TABLE IF NOT EXISTS `ASL_db`.`Email` (
  `posta` VARCHAR(45) NOT NULL,
  `paziente` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`posta`),
  INDEX `fk_Email_Paziente1_idx` (`paziente` ASC) VISIBLE,
  CONSTRAINT `fk_Email_Paziente1`
    FOREIGN KEY (`paziente`)
    REFERENCES `ASL_db`.`Paziente` (`tessera_sanitaria`)
    ON DELETE CASCADE
    ON UPDATE CASCADE)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `ASL_db`.`Esame`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `ASL_db`.`Esame` ;

CREATE TABLE IF NOT EXISTS `ASL_db`.`Esame` (
  `examID` INT NOT NULL AUTO_INCREMENT,
  `descrizione` VARCHAR(100) NOT NULL,
```

```
  `costo` FLOAT(8) NOT NULL,
  PRIMARY KEY (`examID`))
ENGINE = InnoDB;



-- -----------------------------------------------------
-- Table `ASL_db`.`Parametro`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `ASL_db`.`Parametro` ;

CREATE TABLE IF NOT EXISTS `ASL_db`.`Parametro` (
  `nome` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`nome`))
ENGINE = InnoDB;



-- -----------------------------------------------------
-- Table `ASL_db`.`Personale_anagrafica`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `ASL_db`.`Personale_anagrafica` ;

CREATE TABLE IF NOT EXISTS `ASL_db`.`Personale_anagrafica` (
  `CF` VARCHAR(45) NOT NULL,
  `nome` VARCHAR(45) NOT NULL,
  `cognome` VARCHAR(45) NOT NULL,
```

```
  `indirizzo` VARCHAR(100) NOT NULL,
  `tipo` ENUM('medico', 'primario', 'volontario') NOT NULL,
  `associazione` VARCHAR(45) NULL DEFAULT NULL,
  PRIMARY KEY (`CF`))
ENGINE = InnoDB;



-- -----------------------------------------------------
-- Table `ASL_db`.`Ospedale`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `ASL_db`.`Ospedale` ;

CREATE TABLE IF NOT EXISTS `ASL_db`.`Ospedale` (
  `hospID` INT NOT NULL AUTO_INCREMENT,
  `nome` VARCHAR(45) NOT NULL,
  `indirizzo` VARCHAR(100) NOT NULL,
  `primario` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`hospID`),
  INDEX `fk_Ospedale_Personale1_idx` (`primario` ASC) VISIBLE,
  CONSTRAINT `fk_Ospedale_Personale1`
    FOREIGN KEY (`primario`)
    REFERENCES `ASL_db`.`Personale_anagrafica` (`CF`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

```
-- -----------------------------------------------------
-- Table `ASL_db`.`Laboratorio`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `ASL_db`.`Laboratorio` ;

CREATE TABLE IF NOT EXISTS `ASL_db`.`Laboratorio` (
  `hospID` INT NOT NULL,
  `labID` INT NOT NULL,
  `nome` VARCHAR(45) NOT NULL,
  `piano` INT NOT NULL,
  `stanza` VARCHAR(45) NOT NULL,
  `primario` VARCHAR(45) NOT NULL,
  INDEX `fk_Laboratorio_Ospedale1_idx` (`hospID` ASC) VISIBLE,
  INDEX `fk_Laboratorio_Personale1_idx` (`primario` ASC) VISIBLE,
  PRIMARY KEY (`hospID`, `labID`),
  CONSTRAINT `fk_Laboratorio_Ospedale1`
    FOREIGN KEY (`hospID`)
    REFERENCES `ASL_db`.`Ospedale` (`hospID`)
    ON DELETE CASCADE
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Laboratorio_Personale1`
    FOREIGN KEY (`primario`)
    REFERENCES `ASL_db`.`Personale_anagrafica` (`CF`)
```

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



-- -----------------------------------------------------
-- Table `ASL_db`.`Esame_effettivo`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `ASL_db`.`Esame_effettivo` ;

CREATE TABLE IF NOT EXISTS `ASL_db`.`Esame_effettivo` (
  `examID` INT NOT NULL,
  `paziente` VARCHAR(45) NOT NULL,
  `data` DATE NOT NULL,
  `ora` TIME NOT NULL,
  `urgenza` ENUM('si', 'no') NOT NULL,
  `diagnosi` VARCHAR(256) NULL DEFAULT NULL,
  `codiceP` INT NOT NULL,
  `laboratorio` INT NOT NULL,
  `ospedale` INT NOT NULL,
  INDEX `fk_Esame_effettivo_Esame1_idx` (`examID` ASC) VISIBLE,
  INDEX `fk_Esame_effettivo_Paziente1_idx` (`paziente` ASC) INVISIBLE,
  PRIMARY KEY (`examID`, `paziente`, `data`),
  INDEX `fk_Esame_effettivo_Laboratorio1_idx` (`ospedale` ASC, `laboratorio` ASC) VISIBLE,
  CONSTRAINT `fk_Esame_effettivo_Esame1`
```

```
    FOREIGN KEY (`examID`)
    REFERENCES `ASL_db`.`Esame` (`examID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Esame_effettivo_Paziente1`
    FOREIGN KEY (`paziente`)
    REFERENCES `ASL_db`.`Paziente` (`tessera_sanitaria`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Esame_effettivo_Laboratorio1`
    FOREIGN KEY (`ospedale` , `laboratorio`)
    REFERENCES `ASL_db`.`Laboratorio` (`hospID` , `labID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;



-- -----------------------------------------------------
-- Table `ASL_db`.`Composizione`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `ASL_db`.`Composizione` ;

CREATE TABLE IF NOT EXISTS `ASL_db`.`Composizione` (
  `examID` INT NOT NULL,
  `parametro` VARCHAR(45) NOT NULL,
```

```
  PRIMARY KEY (`examID`, `parametro`),
  INDEX `fk_Parametro_has_Esame_Esame1_idx` (`examID` ASC) INVISIBLE,
  INDEX `fk_Parametro_has_Esame_Parametro1_idx` (`parametro` ASC) VISIBLE,
  CONSTRAINT `fk_Parametro_has_Esame_Parametro1`
    FOREIGN KEY (`parametro`)
    REFERENCES `ASL_db`.`Parametro` (`nome`)
    ON DELETE NO ACTION
    ON UPDATE CASCADE,
  CONSTRAINT `fk_Parametro_has_Esame_Esame1`
    FOREIGN KEY (`examID`)
    REFERENCES `ASL_db`.`Esame` (`examID`)
    ON DELETE CASCADE
    ON UPDATE NO ACTION)
ENGINE = InnoDB;



-- -----------------------------------------------------
-- Table `ASL_db`.`Reparto`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `ASL_db`.`Reparto` ;

CREATE TABLE IF NOT EXISTS `ASL_db`.`Reparto` (
  `hospID` INT NOT NULL,
  `repID` INT NOT NULL,
  `nome` VARCHAR(45) NOT NULL,
```

```
  `tel` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`hospID`, `repID`),
  INDEX `fk_Reparto_Ospedale1_idx` (`hospID` ASC) VISIBLE,
  CONSTRAINT `fk_Reparto_Ospedale1`
    FOREIGN KEY (`hospID`)
    REFERENCES `ASL_db`.`Ospedale` (`hospID`)
    ON DELETE CASCADE
    ON UPDATE NO ACTION)
ENGINE = InnoDB;



-- -----------------------------------------------------
-- Table `ASL_db`.`Svolgimento`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `ASL_db`.`Svolgimento` ;

CREATE TABLE IF NOT EXISTS `ASL_db`.`Svolgimento` (
  `esame` INT NOT NULL,
  `paziente` VARCHAR(45) NOT NULL,
  `data` DATE NOT NULL,
  `personale` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`esame`, `paziente`, `data`),
  INDEX `fk_Svolgimento_Personale_anagrafica1_idx` (`personale` ASC) VISIBLE,
  CONSTRAINT `fk_Svolgimento_Esame_effettivo1`
    FOREIGN KEY (`esame` , `paziente` , `data`)
```

```
    REFERENCES `ASL_db`.`Esame_effettivo` (`examID` , `paziente` , `data`)
    ON DELETE CASCADE
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Svolgimento_Personale_anagrafica1`
    FOREIGN KEY (`personale`)
    REFERENCES `ASL_db`.`Personale_anagrafica` (`CF`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;



-- -----------------------------------------------------
-- Table `ASL_db`.`Specializzazione`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `ASL_db`.`Specializzazione` ;

CREATE TABLE IF NOT EXISTS `ASL_db`.`Specializzazione` (
  `titolo` VARCHAR(100) NOT NULL,
  PRIMARY KEY (`titolo`))
ENGINE = InnoDB;



-- -----------------------------------------------------
-- Table `ASL_db`.`Specializzato`
-- -----------------------------------------------------
```

```
DROP TABLE IF EXISTS `ASL_db`.`Specializzato` ;


CREATE TABLE IF NOT EXISTS `ASL_db`.`Specializzato` (
  `primario` VARCHAR(45) NOT NULL,
  `specializzazione` VARCHAR(100) NOT NULL,
  PRIMARY KEY (`primario`, `specializzazione`),
  INDEX `fk_Specializzazione_has_Personale_Personale1_idx` (`primario` ASC) VISIBLE,
  INDEX `fk_Specializzazione_has_Personale_Specializzazione1_idx` (`specializzazione` ASC)
VISIBLE,
  CONSTRAINT `fk_Specializzazione_has_Personale_Specializzazione1`
    FOREIGN KEY (`specializzazione`)
    REFERENCES `ASL_db`.`Specializzazione` (`titolo`)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  CONSTRAINT `fk_Specializzazione_has_Personale_Personale1`
    FOREIGN KEY (`primario`)
    REFERENCES `ASL_db`.`Personale_anagrafica` (`CF`)
    ON DELETE CASCADE
    ON UPDATE NO ACTION)
ENGINE = InnoDB;



-- -----------------------------------------------------
-- Table `ASL_db`.`Users`
-- -----------------------------------------------------
```

```
DROP TABLE IF EXISTS `ASL_db`.`Users` ;


CREATE TABLE IF NOT EXISTS `ASL_db`.`Users` (
  `username` VARCHAR(45) NOT NULL,
  `password` CHAR(32) NOT NULL,
  `ruolo` ENUM('personaleCUP', 'amministratore') NOT NULL,
  PRIMARY KEY (`username`))
ENGINE = InnoDB;



-- -----------------------------------------------------
-- Table `ASL_db`.`Personale_dati_lavorativi`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `ASL_db`.`Personale_dati_lavorativi` ;


CREATE TABLE IF NOT EXISTS `ASL_db`.`Personale_dati_lavorativi` (
  `personale` VARCHAR(45) NOT NULL,
  `ospedale` INT NOT NULL,
  `reparto` INT NOT NULL,
  PRIMARY KEY (`personale`),
  INDEX `fk_Personale_dati_lavorativi_Reparto1_idx` (`ospedale` ASC, `reparto` ASC) VISIBLE,
  CONSTRAINT `fk_Personale_dati_lavorativi_Personale_anagrafica1`
    FOREIGN KEY (`personale`)
    REFERENCES `ASL_db`.`Personale_anagrafica` (`CF`)
    ON DELETE CASCADE
```

```
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Personale_dati_lavorativi_Reparto1`
    FOREIGN KEY (`ospedale` , `reparto`)
    REFERENCES `ASL_db`.`Reparto` (`hospID` , `repID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;



-- -----------------------------------------------------
-- Table `ASL_db`.`Risultato`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `ASL_db`.`Risultato` ;

CREATE TABLE IF NOT EXISTS `ASL_db`.`Risultato` (
  `esame` INT NOT NULL,
  `paziente` VARCHAR(45) NOT NULL,
  `data` DATE NOT NULL,
  `parametro` VARCHAR(45) NOT NULL,
  `valore` FLOAT(8) NOT NULL,
  PRIMARY KEY (`esame`, `paziente`, `data`, `parametro`),
  INDEX `fk_Esame_effettivo_has_Parametro_Parametro1_idx` (`parametro` ASC) VISIBLE,
  INDEX `fk_Esame_effettivo_has_Parametro_Esame_effettivo1_idx` (`esame` ASC, `paziente`
ASC, `data` ASC) VISIBLE,
  CONSTRAINT `fk_Esame_effettivo_has_Parametro_Esame_effettivo1`
```

```
    FOREIGN KEY (`esame` , `paziente` , `data`)
    REFERENCES `ASL_db`.`Esame_effettivo` (`examID` , `paziente` , `data`)
    ON DELETE CASCADE
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Esame_effettivo_has_Parametro_Parametro1`
    FOREIGN KEY (`parametro`)
    REFERENCES `ASL_db`.`Parametro` (`nome`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


USE `ASL_db` ;

DELIMITER ;
SET SQL_MODE = '';
DROP USER IF EXISTS login;
SET
SQL_MODE='ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO
_ZERO_DATE,ERROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION';
CREATE USER 'login' IDENTIFIED BY 'loginUser0!';


GRANT EXECUTE ON procedure `ASL_db`.`login` TO 'login';
SET SQL_MODE = '';
DROP USER IF EXISTS personaleCUP;
```

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

CREATE USER 'personaleCUP' IDENTIFIED BY 'personaleCUP0!';


GRANT EXECUTE ON procedure `ASL_db`.`inserisci_paziente` TO 'personaleCUP';

GRANT EXECUTE ON procedure `ASL_db`.`aggiungi_email` TO 'personaleCUP';

GRANT EXECUTE ON procedure `ASL_db`.`aggiungi_recapito_tel` TO 'personaleCUP';

GRANT EXECUTE ON procedure `ASL_db`.`prenota_esame` TO 'personaleCUP';

GRANT EXECUTE ON procedure `ASL_db`.`scrivi_diagnosi` TO 'personaleCUP';

GRANT EXECUTE ON procedure `ASL_db`.`search_by_codiceP` TO 'personaleCUP';

GRANT EXECUTE ON procedure `ASL_db`.`inserisci_risultato_esame` TO 'personaleCUP';

GRANT EXECUTE ON procedure `ASL_db`.`report_paziente` TO 'personaleCUP';

GRANT EXECUTE ON procedure `ASL_db`.`assegna_personale_ad_esame` TO 'personaleCUP';

GRANT EXECUTE ON procedure `ASL_db`.`list_esami_disponibili` TO 'personaleCUP';

GRANT EXECUTE ON procedure `ASL_db`.`list_ospedali` TO 'personaleCUP';

GRANT EXECUTE ON procedure `ASL_db`.`list_laboratori` TO 'personaleCUP';

GRANT EXECUTE ON procedure `ASL_db`.`list_reparti` TO 'personaleCUP';

GRANT EXECUTE ON procedure `ASL_db`.`select_personale_by_hosp` TO 'personaleCUP';

GRANT EXECUTE ON procedure `ASL_db`.`select_personale_by_rep` TO 'personaleCUP';

GRANT EXECUTE ON procedure `ASL_db`.`list_patients` TO 'personaleCUP';

GRANT EXECUTE ON procedure `ASL_db`.`search_patient` TO 'personaleCUP';

GRANT EXECUTE ON procedure `ASL_db`.`search_recapiti` TO 'personaleCUP';

GRANT EXECUTE ON procedure `ASL_db`.`report_risultati_prenotazione` TO 'personaleCUP';

GRANT EXECUTE ON procedure `ASL_db`.`cancella_prenotazione` TO 'personaleCUP';
```

```
SET SQL_MODE = '';

DROP USER IF EXISTS amministratore;

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

CREATE USER 'amministratore' IDENTIFIED BY 'Amministratore0!';


GRANT EXECUTE ON procedure `ASL_db`.`registration` TO 'amministratore';

GRANT EXECUTE ON procedure `ASL_db`.`crea_parametro` TO 'amministratore';

GRANT EXECUTE ON procedure `ASL_db`.`crea_tipologia_esame` TO 'amministratore';

GRANT EXECUTE ON procedure `ASL_db`.`add_personale_anagrafica` TO 'amministratore';

GRANT EXECUTE ON procedure `ASL_db`.`add_personale_lavorativo` TO 'amministratore';

GRANT EXECUTE ON procedure `ASL_db`.`aggiungi_tipo_specializzazione` TO 'amministratore';

GRANT EXECUTE ON procedure `ASL_db`.`assegna_specializzazione` TO 'amministratore';

GRANT EXECUTE ON procedure `ASL_db`.`inserisci_ospedale` TO 'amministratore';

GRANT EXECUTE ON procedure `ASL_db`.`inserisci_laboratorio` TO 'amministratore';

GRANT EXECUTE ON procedure `ASL_db`.`inserisci_reparto` TO 'amministratore';

GRANT EXECUTE ON procedure `ASL_db`.`add_parametro_ad_esame` TO 'amministratore';

GRANT EXECUTE ON procedure `ASL_db`.`select_personale_by_hosp` TO 'amministratore';

GRANT EXECUTE ON procedure `ASL_db`.`select_personale_by_rep` TO 'amministratore';

GRANT EXECUTE ON procedure `ASL_db`.`report_personale_mese` TO 'amministratore';

GRANT EXECUTE ON procedure `ASL_db`.`report_personale_anno` TO 'amministratore';

GRANT EXECUTE ON procedure `ASL_db`.`list_ospedali` TO 'amministratore';

GRANT EXECUTE ON procedure `ASL_db`.`list_laboratori` TO 'amministratore';

GRANT EXECUTE ON procedure `ASL_db`.`list_reparti` TO 'amministratore';
```

```
GRANT EXECUTE ON procedure `ASL_db`.`list_esami_disponibili` TO 'amministratore';

GRANT EXECUTE ON procedure `ASL_db`.`search_strutture_primario` TO 'amministratore';


SET SQL_MODE=@OLD_SQL_MODE;

SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS;

SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS;

USE `ASL_db`;

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

-- Data for table `ASL_db`.`Users`

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

START TRANSACTION;

USE `ASL_db`;

INSERT INTO `ASL_db`.`Users` (`username`, `password`, `ruolo`) VALUES ('admin', 'c6009f08fc5fc6385f1ea1f5840e179f', 'amministratore');

INSERT INTO `ASL_db`.`Users` (`username`, `password`, `ruolo`) VALUES ('prova', 'c6009f08fc5fc6385f1ea1f5840e179f', 'personaleCUP');


COMMIT;
```

Codice del Front-End

defines.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;

/* used to parse json files into a configuration struct */
extern int parse_config(char *path, struct configuration *conf);

/* to handle I/O interaction */
extern char *getInput (unsigned int lung, char *stringa, bool hide);
extern char multiChoice (char *domanda, char choices[], int num);
extern bool yesOrNo (char *domanda, char yes, char no, bool predef, bool insensitive);

/* connection error */
extern void print_error(MYSQL *conn, char *message);

/* statement error */
extern void print_stmt_error(MYSQL_STMT *stmt, char *message);

/* exit with error */
extern void finish_with_error(MYSQL *conn, char *message);

/* exit with stmt error */
```

```
extern void finish_with_stmt_error(MYSQL *conn, MYSQL_STMT *stmt, char *message, bool
close_stmt);

/* prepare statement for execution */
extern bool setup_prepared_stmt(MYSQL_STMT **stmt, char *statement, MYSQL *conn);



/* run with different configurations */
extern void run_as_amministratore(MYSQL *conn);
extern void run_as_personaleCUP(MYSQL *conn);

/* prepare output tables to be printed well */
extern void dump_result_set(MYSQL *conn, MYSQL_STMT *stmt, char *title);



/* common operations */
extern void list_medical_structures(MYSQL *conn);
extern void list_ospedali(MYSQL *conn);
extern void list_laboratori(MYSQL *conn);
extern void list_reparti(MYSQL *conn);

extern void select_personale_by_hosp(MYSQL *conn);
extern void select_personale_by_rep(MYSQL *conn);

extern void list_exams(MYSQL *conn);
```

utils.c:

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

```
#include <string.h>

#include "defines.h"

#define FLOAT_SIZE 10

/*this function is used to print errors verified during the execution of some stmt */
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));
  }
}

/* to print connection errors */
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
  }
}

/* prepare statement*/
bool setup_prepared_stmt(MYSQL_STMT **stmt, char *statement, MYSQL *conn){

  bool update_length = true;

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

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

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


// exit with error
void finish_with_error(MYSQL *conn, char *message){

  print_error(conn, message);
```

```
  mysql_close(conn);
  exit(EXIT_FAILURE);
}



// exit with stmt error
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, num;

        mysql_field_seek(res_set, 0);
        putchar('+');
        for (i = 0; i < mysql_num_fields(res_set); i++) {
                field = mysql_fetch_field(res_set);
    if (field->type == MYSQL_TYPE_FLOAT){
      num = FLOAT_SIZE;
    }else{
```

```
    num = field->max_length + 2;
  }
            for (j = 0; j < num; 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);
   if (field->type == MYSQL_TYPE_FLOAT){
     printf(" %-*s |", FLOAT_SIZE - 2, field->name);
   }else{
                printf(" %-*s |", (int)field->max_length, field->name);
   }
        }
        putchar('\n');

        print_dashes(res_set);
}


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

```
        MYSQL_TIME *date;
        size_t attr_size;


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


bool is_null[mysql_stmt_num_rows(stmt)];      // added to print NULL values !!!!

        /* 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;
rs_bind[i].is_null = &(is_null[i]);

        if(rs_bind[i].buffer == NULL) {
```

```
                finish_with_stmt_error(conn, stmt, "Cannot allocate output buffers\n",
true);
            }
        }

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

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

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

            putchar('|');

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

                if (is_null[i]){//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_TIME:        //added
          date = (MYSQL_TIME *)rs_bind[i].buffer;
        //printf(" %02d:%-*.02d |", date->hour, (int)fields[i].max_length - 3, date->minute);
          char time[6];
          sprintf(time, "%02d:%02d", date->hour, date->minute);
          printf(" %-*s |", (int)fields[i].max_length, time);
          break;

                                case MYSQL_TYPE_DATE:
                                case MYSQL_TYPE_TIMESTAMP:
                                        date = (MYSQL_TIME *)rs_bind[i].buffer;
                                        printf(" %02d-%02d-%04d |", date->day, date->month,
date->year);  // modified to [DD-MM-YYYY] format
                                        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_SIZE -2, *(float
*)rs_bind[i].buffer);   //modified (float: max 10 digits)
                                        break;

                                case MYSQL_TYPE_LONG:
```

```
                                        case MYSQL_TYPE_SHORT:
        printf(" %-*d |", (int)fields[i].max_length, *(int*)rs_bind[i].buffer); //modified from int* to signed char*
        break;
                                        case MYSQL_TYPE_TINY:
                                                printf(" %-*d |", (int)fields[i].max_length, *(signed char *)rs_bind[i].buffer); //modified from int* to signed char*
        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);
```

```
        }
}
```

parse.c:

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

#include "defines.h"

#define FLOAT_SIZE 10

/*this function is used to print errors verified during the execution of some stmt */
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));
  }
}

/* to print connection errors */
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
  }
}

/* prepare statement*/
bool setup_prepared_stmt(MYSQL_STMT **stmt, char *statement, MYSQL *conn){

  bool update_length = true;

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

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

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

```
// exit with error
void finish_with_error(MYSQL *conn, char *message){

  print_error(conn, message);
  mysql_close(conn);
  exit(EXIT_FAILURE);
}



// exit with stmt error
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, num;

        mysql_field_seek(res_set, 0);
```

```
        putchar('+');
        for (i = 0; i < mysql_num_fields(res_set); i++) {
                field = mysql_fetch_field(res_set);
   if (field->type == MYSQL_TYPE_FLOAT){
    num = FLOAT_SIZE;
   }else{
    num = field->max_length + 2;
   }
                for (j = 0; j < num; 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);
   if (field->type == MYSQL_TYPE_FLOAT){
     printf(" %-*s |", FLOAT_SIZE - 2, field->name);
   }else{
                printf(" %-*s |", (int)field->max_length, field->name);
   }
        }
        putchar('\n');

        print_dashes(res_set);
}


void dump_result_set(MYSQL *conn, MYSQL_STMT *stmt, char *title)
{
```

```
int i;
int status;
int num_fields;       /* number of columns in result */
MYSQL_FIELD *fields;  /* for result set metadata */
MYSQL_BIND *rs_bind;  /* for output buffers */
MYSQL_RES *rs_metadata;
MYSQL_TIME *date;
size_t attr_size;



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


bool is_null[mysql_stmt_num_rows(stmt)];      // added to print NULL values !!!!

/* 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;
    rs_bind[i].is_null = &(is_null[i]);

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

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

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

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

                putchar('|');

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

                        if (is_null[i]){//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_TIME:        //added
          date = (MYSQL_TIME *)rs_bind[i].buffer;
        //printf(" %02d:%-*.02d |", date->hour, (int)fields[i].max_length - 3, date->minute);
          char time[6];
          sprintf(time, "%02d:%02d", date->hour, date->minute);
          printf(" %-*s |", (int)fields[i].max_length, time);
          break;

                        case MYSQL_TYPE_DATE:
                        case MYSQL_TYPE_TIMESTAMP:
                                date = (MYSQL_TIME *)rs_bind[i].buffer;
                                printf(" %02d-%02d-%04d |", date->day, date->month,
date->year);  // modified to [DD-MM-YYYY] format
                                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_SIZE -2, *(float
*)rs_bind[i].buffer);   //modified (float: max 10 digits)
                                        break;

                                    case MYSQL_TYPE_LONG:
                                    case MYSQL_TYPE_SHORT:
        printf(" %-*d |", (int)fields[i].max_length, *(int*)rs_bind[i].buffer); //modified from int* to
signed char*
        break;
                                    case MYSQL_TYPE_TINY:
                                        printf(" %-*d |", (int)fields[i].max_length, *(signed char
*)rs_bind[i].buffer); //modified from int* to signed char*
        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);
        }
}
```

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"

//signals handling
static volatile sig_atomic_t signo;
typedef struct sigaction sigaction_t;
static void handler(int s);

/* to acquire inputs from stdin */
char *getInput(unsigned int lung, char *stringa, bool hide){
```

```
  char c;
  unsigned int i;

  // variables needed to hide input
  sigaction_t sa, savealrm, saveint, savehup, savequit, saveterm;
  sigaction_t savetstp, savettin, savettou;
  struct termios term, oterm;

  if(hide){
    // svuota 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 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));
  }
}

// legge 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;

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

// controllo terminatore di stringa '\0'
if(i == lung -1)
  stringa[i] = '\0';
```

```
// se sono stati digitati più di lung-1 caratteri, svuota buffer tastiera
if(strlen(stringa) >= lung){
  do{
    c = getchar();
  }while (c != '\n');
}

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

  // ripristina impostazioni precedenti schermo
  (void) tcsetattr(fileno(stdin), TCSAFLUSH, &oterm);

  // ripristina gestione 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;
}



// to handle signals
static void handler (int s){
  // salvo il codice del segnale nella variabile statica per gestirlo in seguito
  signo = s;
}




/* genera domanda con multi-opzione di risposta */
char multiChoice(char *domanda, char choices[], int num){
  //genera 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'ultimo '/' inserito alla fine

  // chiede la risposta
  while(true){
    //mostra la domanda
    printf("%s [%s]: ", domanda, possib);
```

```
    char c;
    getInput(1, &c, false); //read 1 char

    // controlla se è un carattere valido
    for(i = 0; i < num; i++){
      if(c == choices[i]){
        free(possib);
        return c;
      }
    }
  } //else repeat while loop
}


// yes or no input function
bool yesOrNo(char *domanda, char yes, char no, bool predef, bool insensitive){

  // i caratteri 'yes' e 'no' devono essere lower_case
  yes = tolower(yes);
  no = tolower(no);

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

```
    n = toupper(no);
  }

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

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

    // controlla quale risposta ha ricevuto
    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;
                        }
  }
}
```

main.c:

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

#include "defines.h"

// protocol between client and DMBS
typedef enum {
  AMMINISTRATORE = 1,
  PERSONALECUP,
  FAILED_LOGIN
} role_t;

struct configuration conf;

static MYSQL *conn;


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

  MYSQL_BIND param[3];
  int role = 0;

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

```
// prepare params; clean memory to avoid unexpected errors!!!
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_LONG; //out; mysql_long --> int (C_lang)
param[2].buffer = &role;
param[2].buffer_length = sizeof(role);

// PARAM binding!
if (mysql_stmt_bind_param(login_procedure, param) != 0){
  print_stmt_error (login_procedure, "Could not bind parameters for login procedure");
  goto err1;
}

// Run procedure
if(mysql_stmt_execute(login_procedure) != 0){
  print_stmt_error(login_procedure, "Could not execute login procedure");
  goto err1;
}

// prepare output params
```

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

  // RESULT binding!
  if (mysql_stmt_bind_result(login_procedure, param)){
    print_stmt_error(login_procedure, "Could not retrieve output parameters");
    goto err1;
  }

  // retrieve output params
  if (mysql_stmt_fetch(login_procedure)){
    print_stmt_error(login_procedure, "Could not buffer results");
    goto err1;
  }

  // close stmt and return result
  mysql_stmt_close(login_procedure);
  return role;

  err1:
    mysql_stmt_close(login_procedure);
  err2:
    return FAILED_LOGIN;
}


int main(void){
```

```
  role_t role;

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

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

  // real CONNECTION to database server
  if (mysql_real_connect(conn, conf.host, conf.db_username, conf.db_password,
  conf.database, conf.port, NULL,
  CLIENT_MULTI_STATEMENTS | CLIENT_MULTI_RESULTS) == NULL){
      fprintf(stderr, "mysql_real_connect() failed\n");
      mysql_close(conn);
      exit(EXIT_FAILURE);
    }
  else{
    printf("Successful connection to database %s\n\n", conf.database);
  }

/*  if (mysql_real_connect(conn, "localhost", "login", "login",
  "ASL_db", 3306, NULL,
```

```
 CLIENT_MULTI_STATEMENTS | CLIENT_MULTI_RESULTS) == NULL){
    fprintf(stderr, "mysql_real_connect() failed: Error: %s\n", mysql_error(conn));
    mysql_close(conn);
    exit(EXIT_FAILURE);
  }
 else{
  printf("Successful connection to database %s\n\n", conf.database);
 }*/

 printf("Username: ");
 getInput(128, conf.username, false);
 printf("Password: ");
 getInput(128, conf.password, true); //read with hide option enabled

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

 switch(role){
  case AMMINISTRATORE:
    run_as_amministratore(conn);
    break;

  case PERSONALECUP:
    run_as_personaleCUP(conn);
    break;

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

```
      break;

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

  // disconnecting from the server
  printf ("Bye\n");
  mysql_close(conn);
  return 0;
}
```

common_op.c:

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

#include "defines.h"

void list_laboratori(MYSQL *conn){

        MYSQL_STMT *p_stmt;
        int status;
        char header[512];

        if (!setup_prepared_stmt(&p_stmt, "call list_laboratori()", conn)){
                finish_with_stmt_error(conn, p_stmt, "Unable to init 'list_laboratori' stmt\n", false);
        }
```

```
	// Nothing to bind!

	if (mysql_stmt_execute(p_stmt) != 0){
		print_stmt_error(p_stmt, "An error occurred during the execution of the query");
		goto close;
	}

	sprintf (header, "\nLaboratories\n");

	do{
		// dump result
		dump_result_set(conn, p_stmt, header);

		status = mysql_stmt_next_result(p_stmt);
		if(status > 0){
			finish_with_stmt_error(conn, p_stmt, "Unexpected condition", true);
		}
	}while(status == 0);

close:
	mysql_stmt_close(p_stmt);
}



void list_ospedali(MYSQL *conn){

	MYSQL_STMT *p_stmt;
```

```
	int status;
	char header[512];

	if (!setup_prepared_stmt(&p_stmt, "call list_ospedali()", conn)){
		finish_with_stmt_error(conn, p_stmt, "Unable to init 'list_ospedali' stmt\n", false);
	}

	// Nothing to bind!

	if (mysql_stmt_execute(p_stmt) != 0){
		print_stmt_error(p_stmt, "An error occurred during the execution of the query");
		goto close;
	}

	sprintf (header, "\nOspedali\n");

	do{
		// dump result
		dump_result_set(conn, p_stmt, header);

		status = mysql_stmt_next_result(p_stmt);
		if(status > 0){
			finish_with_stmt_error(conn, p_stmt, "Unexpected condition", true);
		}
	}while(status == 0);

close:
	mysql_stmt_close(p_stmt);
}
```

```
void list_reparti(MYSQL *conn){

	MYSQL_STMT *p_stmt;
	MYSQL_BIND param[1];

	int status;
	char header[512];

	char c_ospedale[46];
	int ospedale;

	printf("\nHospital (ID): ");
	getInput(46, c_ospedale, false);

	ospedale = atoi(c_ospedale);

	if (!setup_prepared_stmt(&p_stmt, "call list_reparti (?)", conn)){
		finish_with_stmt_error(conn, p_stmt, "Unable to init 'list_reparti' stmt\n", false);
	}

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

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

```
	if (mysql_stmt_bind_param(p_stmt, param) != 0){
		finish_with_stmt_error(conn, p_stmt, "Could not bind parameters for 'list_reparti' procedure\n", true);
	}

	if (mysql_stmt_execute(p_stmt) != 0){
		print_stmt_error(p_stmt, "An error occurred during the execution of the query");
		goto close;
	}

	sprintf (header, "\nDepartments of hospital %d\n", ospedale);

	do{
		// dump result
		dump_result_set(conn, p_stmt, header);

		status = mysql_stmt_next_result(p_stmt);
		if(status > 0){
			finish_with_stmt_error(conn, p_stmt, "Unexpected condition", true);
		}
	}while(status == 0);

close:
	mysql_stmt_close(p_stmt);
}

void select_personale_by_hosp(MYSQL *conn){
```

```
        MYSQL_STMT *p_stmt;
        MYSQL_BIND param[1];
        int status;
        char header[512];

        char c_ospedale[46];
        int ospedale;

        printf("\nHospital (ID): ");
        getInput(46, c_ospedale, false);

        ospedale = atoi(c_ospedale);

        if(!setup_prepared_stmt(&p_stmt, "call select_personale_by_hosp (?)", conn)){
                finish_with_stmt_error(conn, p_stmt, "Unable to init 'select_personale_by_hosp'
stmt\n", false);
        }

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

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

        if (mysql_stmt_bind_param(p_stmt, param) != 0){
                finish_with_stmt_error(conn, p_stmt, "Could not bind parameters for
'select_personale_by_hosp' procedure\n", true);
        }
```

```
	if (mysql_stmt_execute(p_stmt) != 0){
		print_stmt_error(p_stmt, "An error occurred during the execution of the query");
		goto close;
	}

	sprintf (header, "\nStaff members working in hospital %d\n", ospedale);

	do{
		// dump result
		dump_result_set(conn, p_stmt, header);

		status = mysql_stmt_next_result(p_stmt);
		if(status > 0){
			finish_with_stmt_error(conn, p_stmt, "Unexpected condition", true);
		}
	}while(status == 0);

close:
	mysql_stmt_close(p_stmt);
}


void select_personale_by_rep(MYSQL *conn){

	MYSQL_STMT *p_stmt;
	MYSQL_BIND param[2];
	int status;
	char header[512];
```

```
        char c_ospedale[46], c_reparto[46];
        int ospedale, reparto;

        printf("\nHospital (ID): ");
        getInput(46, c_ospedale, false);
        printf("\nDepartment (ID): ");
        getInput(46, c_reparto, false);

        ospedale = atoi(c_ospedale);
        reparto = atoi(c_reparto);

        if(!setup_prepared_stmt(&p_stmt, "call select_personale_by_rep (?, ?)", conn)){
                finish_with_stmt_error(conn, p_stmt, "Unable to init 'select_personale_by_rep' stmt\n", false);
        }

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

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

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

        if (mysql_stmt_bind_param(p_stmt, param) != 0){
                finish_with_stmt_error(conn, p_stmt, "Could not bind parameters for 'select_personale_by_rep' procedure\n", true);
        }
```

```
	if (mysql_stmt_execute(p_stmt) != 0){
		print_stmt_error(p_stmt, "An error occurred during the execution of the query");
		goto close;
	}

	sprintf (header, "\nStaff members working in hospital %d, department %d\n", ospedale,
reparto);

	do{
		// dump result
		dump_result_set(conn, p_stmt, header);

		status = mysql_stmt_next_result(p_stmt);
		if(status > 0){
			finish_with_stmt_error(conn, p_stmt, "Unexpected condition", true);
		}
	}while(status == 0);

close:
	mysql_stmt_close(p_stmt);
}



void list_exams(MYSQL *conn){

	MYSQL_STMT *p_stmt;
	int status;
```

```
	char header[512];

	if (!setup_prepared_stmt(&p_stmt, "call list_esami_disponibili()", conn)){
		finish_with_stmt_error(conn, p_stmt, "Unable to init 'list_esami_disponibili' stmt\n",
false);
	}

	// Nothing to bind!

	if (mysql_stmt_execute(p_stmt) != 0){
		print_stmt_error(p_stmt, "An error occurred during the execution of the query");
		goto close;
	}

	sprintf (header, "\nEsami disponibili\n");

	do{
		// dump result
		dump_result_set(conn, p_stmt, header);

		status = mysql_stmt_next_result(p_stmt);
		if(status > 0){
			finish_with_stmt_error(conn, p_stmt, "Unexpected condition", true);
		}
	}while(status == 0);

close:
	mysql_stmt_close(p_stmt);
}
```

```
// MENU
void list_medical_structures (MYSQL *conn){

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

        while(true){    // while back option is selected
                printf("\033[2J\033[H");        // clean screen
                printf("*** What do you want to list? ***\n\n");
                printf("1) Hospitals\n");
                printf("2) Laboratories\n");
                printf("3) Departments\n");
                printf("4) Go back\n");

                op = multiChoice("Select an option", options, 4);
                switch(op){
                        case '1':
                                list_ospedali(conn);
                                break;
                        case '2':
                                list_laboratori(conn);
                                break;
                        case '3':
                                list_reparti(conn);
                                break;
                        case '4':
                                printf("\nPress 'Enter' to continue\n");
```

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

        getchar();
    }
}
```

amministratore.c:

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

#include "defines.h"


static void registration(MYSQL *conn){

    MYSQL_STMT *prep_stmt;
    MYSQL_BIND param[3];
    char local_options[2] = {'1', '2'}; // possible roles
    char r;

    // input
    char username[46];
    char password[46];
    char var_ruolo[46];
```

```
// get required info
printf("\nUsername: ");
getInput(46, username, false);
printf("Password: ");
getInput(46, password, true);

printf("Assign a role. Possible options are:\n");
printf("\t1) Amministratore\n");
printf("\t2) PersonaleCUP\n");

r = multiChoice("Select role", local_options, 2);
// convert role into enum value
switch(r){
	case '1':
		strcpy(var_ruolo, "amministratore");
		break;

	case '2':
		strcpy(var_ruolo, "personaleCUP");
		break;

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

// prepare stmt
if(!setup_prepared_stmt(&prep_stmt, "call registration(?, ?, ?)", conn)){
```

```
            finish_with_stmt_error(conn, prep_stmt, "Unable to init registration stmt\n", false);
        }

        // prepare input params
        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 = var_ruolo;
        param[2].buffer_length = strlen(var_ruolo);

        // bind params
        if(mysql_stmt_bind_param(prep_stmt, param) != 0){
            finish_with_stmt_error(conn, prep_stmt, "Could not bind parameters for registration
procedure\n", true);
        }

        // run procedure
        if (mysql_stmt_execute(prep_stmt) != 0){
            print_stmt_error(prep_stmt, "An error occurred during registration");
            goto close;
        } else{
```

```
            printf("The new user has been created successfully\n");
        }

        // closing stmt
close:
        mysql_stmt_close(prep_stmt);
}



static void crea_parametro(MYSQL *conn){

        MYSQL_STMT *p_stmt;
        MYSQL_BIND param[1];

        char name[46];

        printf ("\nName of parameter: ");
        getInput(46, name, false);

        if(!setup_prepared_stmt(&p_stmt, "call crea_parametro(?)", conn)){
            finish_with_stmt_error(conn, p_stmt, "Unable to init 'crea_parametro' stmt", false);
        }

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

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

```
	if(mysql_stmt_bind_param(p_stmt, param) != 0){
		finish_with_stmt_error(conn, p_stmt, "Unable to bind parameters for 'crea_parametro' procedure", true);
	}

	//execute
	if(mysql_stmt_execute(p_stmt) != 0){
		print_stmt_error(p_stmt, "An error occurred during 'crea_parametro' procedure");
	}else{
		printf("\nParameter created successfully\n");
	}

	mysql_stmt_close(p_stmt);
}



static void crea_tipologia_esame(MYSQL *conn){

	MYSQL_STMT *p_stmt;
	MYSQL_BIND param[3];

	char description[101];
	char cost[12];
	unsigned int examId;  // output

	float f_cost;
```

```
        printf("\nDescription (exam name): " );
        getInput(101, description, false);
        printf("\nCost [€€.**]: ");
        getInput(12, cost, false);



        f_cost = strtof(cost, NULL);

        if(!setup_prepared_stmt(&p_stmt, "call crea_tipologia_esame(?, ?, ?)", conn)){
                finish_with_stmt_error(conn, p_stmt, "Unable to init 'crea_tipologia_esame'
statement", false);
        }

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

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

        param[1].buffer_type = MYSQL_TYPE_FLOAT;
        param[1].buffer = &f_cost;
        param[1].buffer_length = sizeof(f_cost);

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

        if(mysql_stmt_bind_param(p_stmt, param) != 0){
                finish_with_stmt_error(conn, p_stmt, "Unable to bind parameters for
'crea_tipologia_esame' procedure", true);
```

```
	}

	if(mysql_stmt_execute(p_stmt) != 0){
		print_stmt_error(p_stmt, "An error occurred during 'crea_tipologia_esame' procedure
");
		goto exit;
	}

	// get back the examID
	memset(param, 0, sizeof(param));
	param[0].buffer_type = MYSQL_TYPE_LONG;
	param[0].buffer = &examId;
	param[0].buffer_length = sizeof(examId);

	// bind result
	if (mysql_stmt_bind_result(p_stmt, param)){
		finish_with_stmt_error(conn, p_stmt, "Could not retrieve output parameter", true);
	}

	// Retrieve output param
	if (mysql_stmt_fetch(p_stmt)){
		finish_with_stmt_error(conn, p_stmt, "Could not buffer results", true);
	}

	printf("New type of exam correctly created with ID: %d\n", examId);

exit:
	mysql_stmt_close(p_stmt);
}
```

```
static void aggiungi_parametro_ad_esame(MYSQL *conn){
	MYSQL_STMT *p_stmt;
	MYSQL_BIND param[2];

	int examId;
	char parametro[46], c_examId[12];

	printf("\nExam (ID number): ");
	getInput(12, c_examId, false);
	printf("\nParameter name: ");
	getInput(46, parametro, false);

	examId = atoi(c_examId);

	if(!setup_prepared_stmt(&p_stmt, "call add_parametro_ad_esame(?, ?)", conn)){
		finish_with_stmt_error(conn, p_stmt, "Unable to init 'add_parametro_ad_esame' statement", false);
	}

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

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

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

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



	if(mysql_stmt_bind_param(p_stmt, param) != 0){
		finish_with_stmt_error(conn, p_stmt, "Unable to bind parameters for 'add_parametro_ad_esame' procedure", true);
	}

	if(mysql_stmt_execute(p_stmt) != 0){
		print_stmt_error(p_stmt, "An error occurred during 'add_parametro_ad_esame' procedure ");
	}else{
		printf("Parameter correctly added\n");
	}

	mysql_stmt_close(p_stmt);
}



static void crea_personale(MYSQL *conn){
	MYSQL_STMT *p_stmt;
	MYSQL_BIND param[6];
	bool is_null = true;



	char local_options[3] = {'1', '2', '3'}; // possible roles
	char r;

	//params
	char cf[46], nome[46], cognome[46], indirizzo[101], tipo[46], associazione[46];
```

```
printf("\nFiscal Code (CF): ");
getInput(46, cf, false);
printf("\nName: ");
getInput(46, nome, false);
printf("\nSurname: ");
getInput(46, cognome, false);
printf("\nAddress: ");
getInput(46, indirizzo, false);
printf("\nRole:\n");
printf("\t1) Medico\n");
printf("\t2) Primario\n");
printf("\t3) Volontario\n");

r = multiChoice("Select role", local_options, 3);
switch(r){
	case '1':
		strcpy(tipo, "medico");
		break;
	case '2':
		strcpy(tipo, "primario");
		break;
	case '3':
		strcpy(tipo, "volontario");
		break;
	default:
		fprintf(stderr, "Invalid condition at %s:%d\n", __FILE__, __LINE__);
		abort();
}
```

```
        if (r == '3'){
                printf("\nVoluntary association (optional): ");
                getInput(46, associazione, false);
                if (strcmp(associazione, "") != 0){
                        is_null = false;
                }
        }

        if(!setup_prepared_stmt(&p_stmt, "call add_personale_anagrafica (?, ?, ?, ?, ?, ?)", conn)){
                finish_with_stmt_error(conn, p_stmt, "Unable to init 'add_personale_anagrafica' statement", 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 = 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 = indirizzo;
        param[3].buffer_length = strlen(indirizzo);

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

        param[5].buffer_type = MYSQL_TYPE_VAR_STRING;    // possible NULL parameter
        param[5].buffer = associazione;
        param[5].is_null = &(is_null);
        param[5].buffer_length = strlen(associazione);




        if(mysql_stmt_bind_param(p_stmt, param) != 0){
                finish_with_stmt_error(conn, p_stmt, "Unable to bind parameters for
'add_personale_anagrafica' procedure", true);
        }

        if(mysql_stmt_execute(p_stmt) != 0){
                print_stmt_error(p_stmt, "An error occurred during 'add_personale_anagrafica'
procedure ");
        }else{
                printf("New staff member correctly created");
        }

        mysql_stmt_close(p_stmt);
}
```

```
static void aggiungi_dati_lavorativi(MYSQL *conn){
	MYSQL_STMT *p_stmt;
	MYSQL_BIND param[3];
	//params
	char cf[46], c_ospedale[46], c_reparto[46];
	int ospedale, reparto;

	printf("\nFiscal Code (CF): ");
	getInput(46, cf, false);
	printf("\nHospital (ID): ");
	getInput(46, c_ospedale, false);
	printf("\nDepartment (ID): ");
	getInput(46, c_reparto, false);

	ospedale = atoi(c_ospedale);
	reparto = atoi(c_reparto);

	if(!setup_prepared_stmt(&p_stmt, "call add_personale_lavorativo (?, ?, ?)", conn)){
		finish_with_stmt_error(conn, p_stmt, "Unable to init 'add_personale_lavorativo' statement", 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 = &ospedale;
	param[1].buffer_length = sizeof(ospedale);

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

	if(mysql_stmt_bind_param(p_stmt, param) != 0){
		finish_with_stmt_error(conn, p_stmt, "Unable to bind parameters for 'add_personale_lavorativo' procedure", true);
	}

	if(mysql_stmt_execute(p_stmt) != 0){
		print_stmt_error(p_stmt, "An error occurred during 'add_personale_lavorativo' procedure ");
	}else{
		printf("Work data correctly added\n");
	}

	mysql_stmt_close(p_stmt);
}


static void crea_specializzazione(MYSQL *conn){
	MYSQL_STMT *p_stmt;
	MYSQL_BIND param[1];

	char titolo[101];
```

```
        printf("\nSpecialization name: ");
        getInput(101, titolo, false);

        if(!setup_prepared_stmt(&p_stmt, "call aggiungi_tipo_specializzazione (?)", conn)){
                finish_with_stmt_error(conn, p_stmt, "Unable to init 'aggiungi_tipo_specializzazione' statement", false);
        }

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

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

        if(mysql_stmt_bind_param(p_stmt, param) != 0){
                finish_with_stmt_error(conn, p_stmt, "Unable to bind parameters for 'aggiungi_tipo_specializzazione' procedure", true);
        }

        if(mysql_stmt_execute(p_stmt) != 0){
                print_stmt_error(p_stmt, "An error occurred during 'aggiungi_tipo_specializzazione' procedure ");
        }else{
                printf("New specialization correctly created\n");
        }

        mysql_stmt_close(p_stmt);
}
```

```
static void assegna_specializzazione(MYSQL *conn){
	MYSQL_STMT *p_stmt;
	MYSQL_BIND param[2];

	char cf[46], titolo[101];

	printf("\nFiscal Code (CF): ");
	getInput(46, cf, false);
	printf("\nSpecialization name: ");
	getInput(101, titolo, false);

	if(!setup_prepared_stmt(&p_stmt, "call assegna_specializzazione (?, ?)", conn)){
		finish_with_stmt_error(conn, p_stmt, "Unable to init 'assegna_specializzazione' statement", 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 = titolo;
	param[1].buffer_length = strlen(titolo);

	if(mysql_stmt_bind_param(p_stmt, param) != 0){
		finish_with_stmt_error(conn, p_stmt, "Unable to bind parameters for 'assegna_specializzazione' procedure", true);
```

```
	}

	if(mysql_stmt_execute(p_stmt) != 0){
		print_stmt_error(p_stmt, "An error occurred during 'assegna_specializzazione' procedure ");
	}else{
		printf("Specialization correctly assigned\n");
	}

	mysql_stmt_close(p_stmt);
}

static void crea_ospedale (MYSQL *conn){

		MYSQL_STMT *p_stmt;
		MYSQL_BIND param[4];

		char nome[46], indirizzo[101], primario[46];
		int hospId;

		printf("\nHospital name: ");
		getInput(46, nome, false);
		printf("\nAddress: ");
		getInput(101, indirizzo, false);
		printf("\nHead physician (CF): ");
		getInput(46, primario, false);
```

```
        if(!setup_prepared_stmt(&p_stmt, "call inserisci_ospedale (?, ?, ?, ?)", conn)){
            finish_with_stmt_error(conn, p_stmt, "Unable to init 'inserisci_ospedale' statement", false);
        }

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

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

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

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

        param[3].buffer_type = MYSQL_TYPE_LONG;     //out
        param[3].buffer = &hospId;
        param[3].buffer_length = sizeof(hospId);

        if(mysql_stmt_bind_param(p_stmt, param) != 0){
            finish_with_stmt_error(conn, p_stmt, "Unable to bind parameters for 'inserisci_ospedale' procedure", true);
        }

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

```
                print_stmt_error(p_stmt, "An error occurred during 'inserisci_ospedale'
procedure ");
                goto exit_stmt;
        }

        //get back the hospId
        memset(param, 0, sizeof(param));
        param[0].buffer_type = MYSQL_TYPE_LONG;
        param[0].buffer = &hospId;
        param[0].buffer_length = sizeof(hospId);

        // binding results
        if(mysql_stmt_bind_result(p_stmt, param)){
                finish_with_stmt_error(conn, p_stmt, "Could not retrieve output parameters",
true);
        }

        //retrieve results
        if(mysql_stmt_fetch(p_stmt)){
                finish_with_stmt_error(conn, p_stmt, "Could not retrieve buffer results", true);
        }

        printf("New hospital correctly created with ID: %d\n", hospId);
    exit_stmt:
        mysql_stmt_close(p_stmt);

}


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

```
MYSQL_STMT *p_stmt;
MYSQL_BIND param[6];

char c_ospedale[46], c_labId[46], nome[46], c_piano[46], stanza[46], primario[46];
int hospId, piano, labId;

printf("\nHospital (ID): ");
getInput(46, c_ospedale, false);
printf("\nLaboratory ID: ");
getInput(46, c_labId, false);
printf("\nName: ");
getInput(46, nome, false);
printf("\nFloor [number]: ");
getInput(46, c_piano, false);
printf("\nRoom: ");
getInput(46, stanza, false);
printf("\nHead physician (CF): ");
getInput(46, primario, false);

hospId = atoi(c_ospedale);
labId = atoi(c_labId);
piano = atoi(c_piano);

if(!setup_prepared_stmt(&p_stmt, "call inserisci_laboratorio (?, ?, ?, ?, ?, ?)", conn)){
        finish_with_stmt_error(conn, p_stmt, "Unable to init 'inserisci_laboratorio'
statement", false);
}
```

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

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

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

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

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

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

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



if(mysql_stmt_bind_param(p_stmt, param) != 0){
```

```
                finish_with_stmt_error(conn, p_stmt, "Unable to bind parameters for
'inserisci_laboratorio' procedure", true);
        }

        if(mysql_stmt_execute(p_stmt) != 0){
                print_stmt_error(p_stmt, "An error occurred during 'inserisci_laboratorio'
procedure ");
        }else{
                printf("New laboratory correctly created \n");
        }

        mysql_stmt_close(p_stmt);

}

static void crea_reparto(MYSQL *conn){

        MYSQL_STMT *p_stmt;
        MYSQL_BIND param[4];

        char c_ospedale[46], c_repId[46], nome[46], tel[46] ;
        int hospId, repId;

        printf("\nHospital (ID): ");
        getInput(46, c_ospedale, false);
        printf("\nDepartment ID: ");
        getInput(46, c_repId, false);
        printf("\nName: ");
        getInput(46, nome, false);
```

```
        printf("\nPhone number: ");
        getInput(46, tel, false);

        hospId = atoi(c_ospedale);
        repId = atoi(c_repId);

        if(!setup_prepared_stmt(&p_stmt, "call inserisci_reparto (?, ?, ?, ?)", conn)){
                finish_with_stmt_error(conn, p_stmt, "Unable to init 'inserisci_reparto' statement",
false);
        }

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

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

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

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

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

```
	if(mysql_stmt_bind_param(p_stmt, param) != 0){
		finish_with_stmt_error(conn, p_stmt, "Unable to bind parameters for 'inserisci_reparto' procedure", true);
	}

	if(mysql_stmt_execute(p_stmt) != 0){
		print_stmt_error(p_stmt, "An error occurred during 'inserisci_reparto' procedure ");
	}else{
		printf("New department correctly created\n");
	}

	mysql_stmt_close(p_stmt);
}



static void report_personale(MYSQL *conn){

	MYSQL_STMT *p_stmt;
	MYSQL_BIND param[2];
	int status;
	char header[512];
	bool first = true;

	char options[] = {'1', '2'};
	char op;
	bool month_report = true;
```

```
char personale[46];
int num_esami;


printf("\nStaff member (CF): ");
getInput(46, personale, false);
printf("\nWhat report do you want to select?");
printf("\n\t1) Last month");
printf("\n\t2) Last year\n");

op = multiChoice("Select an option.", options, 2);
switch(op){
	case '1':
		month_report = true;
		break;

	case '2':
		month_report = false;
		break;

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


if(month_report){
	if(!setup_prepared_stmt(&p_stmt, "call report_personale_mese (?, ?)", conn)){
```

```
                finish_with_stmt_error(conn, p_stmt, "Unable to init 'report_personale_mese' statement", false);
            }
            sprintf(header, "\nExecuted exams in the last month by staff member with CF: %s\n", personale);
        }else{
            if(!setup_prepared_stmt(&p_stmt, "call report_personale_anno (?, ?)", conn)){
                finish_with_stmt_error(conn, p_stmt, "Unable to init 'report_personale_mese' statement", false);
            }
            sprintf(header, "\nExecuted exams in the last year by staff member with CF: %s\n", personale);
        }

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

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

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

        if(mysql_stmt_bind_param(p_stmt, param) != 0){
            finish_with_stmt_error(conn, p_stmt, "Unable to bind parameters for the procedure", true);
        }

        if(mysql_stmt_execute(p_stmt) != 0){
            print_stmt_error(p_stmt, "An error occurred during the procedure ");
```

```
        goto close;
}


//get back the num_exams
memset(param, 0, sizeof(param));
param[0].buffer_type = MYSQL_TYPE_LONG;
param[0].buffer = &num_esami;
param[0].buffer_length = sizeof(num_esami);


//MULTIPLE RESULT SETS!!!
        do{

                if(conn->server_status & SERVER_PS_OUT_PARAMS){
                        goto next;
                }

                if(first){
                        dump_result_set(conn, p_stmt, header);
                        first = false;
                }else{
                        dump_result_set(conn, p_stmt, "");
                }

        next:
                status = mysql_stmt_next_result(p_stmt);
                if (status > 0){            // > 0 error, 0 keep looking, -1 finished
                        finish_with_stmt_error(conn, p_stmt, "Unexpected condition", true);
```

```
                }
        }while(status == 0);

close:
    mysql_stmt_close(p_stmt);
}



static void search_strutture_primario(MYSQL *conn){

    MYSQL_STMT *p_stmt;
    MYSQL_BIND param[1];
    int status;
    char header[512];
    bool first = true;

    char primario[46];

    printf("\nStaff member (CF): ");
    getInput(46, primario, false);

    if(!setup_prepared_stmt(&p_stmt, "call search_strutture_primario (?)", conn)){
        finish_with_stmt_error(conn, p_stmt, "Unable to init 'search_strutture_primario' statement", false);
    }

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

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



	if(mysql_stmt_bind_param(p_stmt, param) != 0){
		finish_with_stmt_error(conn, p_stmt, "Unable to bind parameters for the procedure",
true);
	}

	if(mysql_stmt_execute(p_stmt) != 0){
		print_stmt_error(p_stmt, "An error occurred during the procedure ");
		goto close;
	}

	//MULTIPLE RESULT SETS!!!
		do{

			if(conn->server_status & SERVER_PS_OUT_PARAMS){
				goto next;
			}

			if(first){
				sprintf(header, "\nHospitals with head physician: %s\n", primario);
				first = false;
			}else{
				sprintf(header, "\nLaboratories with head physician: %s\n", primario);
			}
```

```
                dump_result_set(conn, p_stmt, header);
        next:
                status = mysql_stmt_next_result(p_stmt);
                if (status > 0){        // > 0 error, 0 keep looking, -1 finished
                        finish_with_stmt_error(conn, p_stmt, "Unexpected condition", true);
                }
        }while(status == 0);

close:
        mysql_stmt_close(p_stmt);
}



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

        while(true){    // while back option is selected
                printf("\033[2J\033[H");        // clean screen
                printf("*** What do you want to do? ***\n\n");
                printf("1) Create new exam type\n");
                printf("2) Create new parameter\n");
                printf("3) Assign parameter to an exam\n");
                printf("4) List available exams\n");
                printf("5) Go back\n");

                op = multiChoice("Select an option", options, 5);
```

```
                switch(op){
                        case '1':
                                crea_tipologia_esame(conn);
                                break;
                        case '2':
                                crea_parametro(conn);
                                break;
                        case '3':
                                aggiungi_parametro_ad_esame(conn);
                                break;
                        case '4':
                                list_exams(conn);
                                break;
                        case '5':
                                printf("\nPress 'Enter' to continue\n");
                                return;
                        default:
                                fprintf(stderr, "Invalid condition at %s:%d\n", __FILE__, __LINE__);
                                abort();
                }

                getchar();
        }
}

//MENU
static void manage_staff_members (MYSQL *conn){

        char options[] = {'1', '2', '3', '4', '5', '6', '7', '8', '9'};
```

```
char op;

while(true){    // while back option is selected
        printf("\033[2J\033[H");        // clean screen
        printf("*** What do you want to do? ***\n\n");
        printf("1) Create new staff member (personal data)\n");
        printf("2) Add work data to a staff member\n");
        printf("3) Create new specialization\n");
        printf("4) Assign specialization to a primary\n");
        printf("5) Search staff members by hospital\n");
        printf("6) Search staff members by department\n");
        printf("7) Report of executed exams by a staff member\n");
        printf("8) Search hospitals and laboratories by head physician\n");
        printf("9) Go back\n");

        op = multiChoice("Select an option", options, 9);
        switch(op){
                case '1':
                        crea_personale(conn);
                        break;
                case '2':
                        aggiungi_dati_lavorativi(conn);
                        break;
                case '3':
                        crea_specializzazione(conn);
                        break;
                case '4':
                        assegna_specializzazione(conn);
                        break;
```

```
                case '5':
                        select_personale_by_hosp(conn);
                        break;
                case '6':
                        select_personale_by_rep(conn);
                        break;
                case '7':
                        report_personale(conn);
                        break;
                case '8':
                        search_strutture_primario(conn);
                        break;
                case '9':
                        printf("\nPress 'Enter' to continue\n");
                        return;
                default:
                        fprintf(stderr, "Invalid condition at %s:%d\n", __FILE__, __LINE__);
                        abort();
        }

        getchar();
    }
}

//MENU
static void create_medical_structure (MYSQL *conn){

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

```
while(true){    // while back option is selected
        printf("\033[2J\033[H");        // clean screen
        printf("*** What do you want to do? ***\n\n");
        printf("1) Create new hospital\n");
        printf("2) Create new laboratory\n");
        printf("3) Create new department\n");
        printf("4) Go back\n");

        op = multiChoice("Select an option", options, 4);
        switch(op){
                case '1':
                        crea_ospedale(conn);
                        break;
                case '2':
                        crea_laboratorio(conn);
                        break;
                case '3':
                        crea_reparto(conn);
                        break;
                case '4':
                        printf("\nPress 'Enter' to continue\n");
                        return;
                default:
                        fprintf(stderr, "Invalid condition at %s:%d\n", __FILE__, __LINE__);
                        abort();
        }

        getchar();
```

```
        }
}


void run_as_amministratore(MYSQL *conn){

        char options[] = {'1', '2', '3', '4', '5', '6'};
        char op;

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

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

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

        // showing multiChoice after cleaning screen
        while(true){    // while quit option is selected
                printf("\033[2J\033[H");        // clean screen
                printf("*** What do you want to do? ***\n\n");
                printf("1) Create new user\n");
                printf("2) Manage exams\n");
                printf("3) Manage staff members\n");
                printf("4) Create new medical structure\n");
```

```
printf("5) List of medical structure\n");
printf("6) Quit\n");

op = multiChoice("Select an option", options, 6);
switch(op){
        case '1':
                registration(conn);
                break;

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

        case '3':
                manage_staff_members(conn);
                break;

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

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

        case '6':
                return;

        default:
```

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

                getchar(); //cleaning '\n'
        }
}
```

personaleCUP.c:

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

#include "defines.h"

static void add_patient(MYSQL *conn){

        MYSQL_STMT *prep_stmt;
        MYSQL_BIND param[6];

        //input
        char tess_sanitaria[46], nome[46], cognome[46], indirizzo [101], luogoN[46];
        MYSQL_TIME dataN;
        char day[3], month[3], year[5];

        // get input
        printf("\nTessera sanitaria: ");
        getInput(46, tess_sanitaria, false);
        printf("\nNome: ");
```

```
getInput(46, nome, false);
printf("\nCognome: ");
getInput(46, cognome, false);
printf("\nIndirizzo: ");
getInput(101, indirizzo, false);

printf("\nData di nascita [DD/MM/YYYY]");
printf("\nGiorno: ");
getInput(3, day, false);
printf("\nMese: ");
getInput(3, month, false);
printf("\nAnno: ");
getInput(5, year, false);

printf("\nLuogo di nascita: ");
getInput(46, luogoN, false);

// prepare stmt
if(!setup_prepared_stmt(&prep_stmt, "call inserisci_paziente(?, ?, ?, ?, ?, ?)", conn)){
        finish_with_stmt_error(conn, prep_stmt, "Unable to init 'registration' stmt\n", false);
}

// prepare input params
memset(param, 0, sizeof(param));
memset(&dataN, 0, sizeof(dataN));

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

```
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 = indirizzo;
param[3].buffer_length = strlen(indirizzo);

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

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

// set struct MYSQL_TIME params
dataN.day = atoi(day);
dataN.month = atoi(month);
dataN.year = atoi(year);
dataN.time_type = MYSQL_TIMESTAMP_DATE; //!!!

// bind params
if(mysql_stmt_bind_param(prep_stmt, param) != 0){
```

```
		finish_with_stmt_error(conn, prep_stmt, "Could not bind parameters for 'inserisci_paziente' procedure\n", true);
	}

	// run procedure
	if (mysql_stmt_execute(prep_stmt) != 0){
		print_stmt_error(prep_stmt, "An error occurred while creating the new patient");
	} else{
		printf("Nuovo paziente creato correttamente\n");
	}

	// closing stmt
	mysql_stmt_close(prep_stmt);
}


static void prenota_esame(MYSQL *conn){

	MYSQL_STMT *prep_stmt;
	MYSQL_BIND param[9];
	MYSQL_TIME date;
	MYSQL_TIME hour;

	//input
	// diagnosi sempre NULL nell'inserimento, si aggiorna in seguito con altra procedure
	bool is_null = true;

	char c_examId[46], paziente[46], diagnosi[256], c_lab[46], c_hosp[46], c_codiceP[46];
	int examId, lab, hosp, codiceP;
```

```
        char day[3], month[3], year[5], ore[3], minuti[3];
        int i_ore, i_minuti;
        char urgenza[10];

        // get input
        printf("\nTipologia esame (ID): ");
        getInput(46, c_examId, false);
        printf("\nPaziente (tessera sanitaria): ");
        getInput(46, paziente, false);

        printf("\nData esame [DD/MM/YYYY]");
        printf("\nGiorno: ");
        getInput(3, day, false);
        printf("\nMese: ");
        getInput(3, month, false);
        printf("\nAnno: ");
        getInput(5, year, false);

        printf("\nOrario esame [hh:mm]");
retry:
        printf("\nOre: ");
        getInput(3, ore, false);
        printf("\nMinuti: ");
        getInput(3, minuti, false);
        i_ore = atoi(ore);
        i_minuti = atoi(minuti);
        if(!((i_ore >= 0) && (i_ore <24) && (i_minuti >= 0) && (i_minuti < 60))){
                printf("\nLe ore devono essere comprese tra 0 e 23, i minuti tra 0 e 59.\nRiprova\n");
                goto retry;
```

```
        }

        if(yesOrNo("Con urgenza?", 'y', 'n', false, true))
                strcpy(urgenza, "si");
        else
                strcpy(urgenza, "no");

        printf("\nCodice Prenotazione: ");
        getInput(46, c_codiceP, false);
        printf("\nOspedale (ID): ");
        getInput(46, c_hosp, false);
        printf("\nLaboratorio (ID): ");
        getInput(46, c_lab, false);

        examId = atoi(c_examId);
        codiceP = atoi(c_codiceP);
        lab = atoi(c_lab);
        hosp = atoi(c_hosp);

        // prepare stmt
        if(!setup_prepared_stmt(&prep_stmt, "call prenota_esame(?, ?, ?, ?, ?, ?, ?, ?, ?)", conn)){
                finish_with_stmt_error(conn, prep_stmt, "Unable to init 'prenota_esame' stmt\n",
false);
        }

        // prepare input params
        memset(param, 0, sizeof(param));
        memset(&date, 0, sizeof(date));
        memset(&hour, 0, sizeof(hour));
```

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

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

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

param[3].buffer_type = MYSQL_TYPE_TIME;
param[3].buffer = (char *)&hour;
param[3].buffer_length = sizeof(hour);

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

param[5].buffer_type = MYSQL_TYPE_VAR_STRING;
param[5].buffer = diagnosi;
param[5].is_null = &is_null;          // always true!
param[5].buffer_length = strlen(diagnosi);

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

```
param[7].buffer_type = MYSQL_TYPE_LONG;
param[7].buffer = &lab;
param[7].buffer_length = sizeof(lab);

param[8].buffer_type = MYSQL_TYPE_LONG;
param[8].buffer = &hosp;
param[8].buffer_length = sizeof(examId);



// set struct MYSQL_TIME for date param
date.day = atoi(day);
date.month = atoi(month);
date.year = atoi(year);
date.time_type = MYSQL_TIMESTAMP_DATE;

// set struct MYSQL_TIME for hour param
hour.hour = i_ore;
hour.minute = i_minuti;
hour.time_type = MYSQL_TIMESTAMP_TIME; //!!!

// bind params
if(mysql_stmt_bind_param(prep_stmt, param) != 0){
        finish_with_stmt_error(conn, prep_stmt, "Could not bind parameters for
'prenota_esame' procedure\n", true);
}

// run procedure
if (mysql_stmt_execute(prep_stmt) != 0){
```

```
            print_stmt_error(prep_stmt, "An error occurred creating the reservation");
        } else{
            printf("Esame prenotato correttamente\n");
        }

        // closing stmt
        mysql_stmt_close(prep_stmt);
}


static void add_diagnosi(MYSQL *conn){

        MYSQL_STMT *prep_stmt;
        MYSQL_BIND param[4];

        //input
        char c_examId[46], paziente[46], diagnosi[257];
        MYSQL_TIME date;
        char day[3], month[3], year[5];
        int examId;

        // get input
        printf("\nCodice esame: ");
        getInput(46, c_examId, false);
        printf("\nPaziente (tessera sanitaria): ");
        getInput(46, paziente, false);

        printf("\nData esame [DD/MM/YYYY]");
        printf("\nGiorno: ");
```

```
        getInput(3, day, false);
        printf("\nMese: ");
        getInput(3, month, false);
        printf("\nAnno: ");
        getInput(5, year, false);
        printf("\nDiagnosi (max 256 caratteri):\n");
        getInput(257, diagnosi, false);

        examId = atoi (c_examId);

        // prepare stmt
        if(!setup_prepared_stmt(&prep_stmt, "call scrivi_diagnosi(?, ?, ?, ?)", conn)){
                finish_with_stmt_error(conn, prep_stmt, "Unable to init 'scrivi_diagnosi' stmt\n",
false);
        }

        // prepare input params
        memset(param, 0, sizeof(param));
        memset(&date, 0, sizeof(date));

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

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

        param[2].buffer_type = MYSQL_TYPE_DATE;
```

```
        param[2].buffer = (char *)&date;
        param[2].buffer_length = sizeof(date);

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

        // set struct MYSQL_TIME params
        date.day = atoi(day);
        date.month = atoi(month);
        date.year = atoi(year);
        date.time_type = MYSQL_TIMESTAMP_DATE;

        // bind params
        if(mysql_stmt_bind_param(prep_stmt, param) != 0){
                finish_with_stmt_error(conn, prep_stmt, "Could not bind parameters for
'scrivi_diagnosi' procedure\n", true);
        }

        // run procedure
        if (mysql_stmt_execute(prep_stmt) != 0){
                print_stmt_error(prep_stmt, "An error occurred during the operation");
        } else{
                printf("\nDiagnosi assegnata correttamente\n");
        }

        // closing stmt
        mysql_stmt_close(prep_stmt);
}
```

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

	MYSQL_STMT *p_stmt;
	MYSQL_BIND param[1];
	int status;
	char header[512];

	char c_codiceP[46];
	int codiceP;

	printf("\nCodice prenotazione: ");
	getInput(46, c_codiceP, false);

	codiceP = atoi(c_codiceP);

	if (!setup_prepared_stmt(&p_stmt, "call search_by_codiceP (?)", conn)){
		finish_with_stmt_error(conn, p_stmt, "Unable to init 'search_by_codiceP' stmt\n",
false);
	}

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

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

```
	if (mysql_stmt_bind_param(p_stmt, param) != 0){
		finish_with_stmt_error(conn, p_stmt, "Could not bind params for 'search_by_codiceP' procedure\n", true);
	}

	if (mysql_stmt_execute(p_stmt) != 0){
		print_stmt_error(p_stmt, "An error occurred during the execution of the query");
		goto close;
	}

	sprintf (header, "\nEsami con codice di prenotazione %d\n", codiceP);

	do{
		// dump result
		dump_result_set(conn, p_stmt, header);

		status = mysql_stmt_next_result(p_stmt);
		if(status > 0){
			finish_with_stmt_error(conn, p_stmt, "Unexpected condition", true);
		}
	}while(status == 0);

close:
	mysql_stmt_close(p_stmt);
}



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

```
        MYSQL_STMT *prep_stmt;
        MYSQL_BIND param[2];

        char email[46], paziente[46];

        printf("\nPaziente (tessera sanitaria): ");
        getInput(46, paziente, false);
        printf("\nEmail: ");
        getInput(46, email, false);

        if(!setup_prepared_stmt(&prep_stmt, "call aggiungi_email(?, ?)", conn)){
                finish_with_stmt_error(conn, prep_stmt, "Unable to init 'aggiungi_email' stmt\n",
false);
        }

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

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

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

        if(mysql_stmt_bind_param(prep_stmt, param) != 0){
                finish_with_stmt_error(conn, prep_stmt, "Could not bind parameters for
'aggiungi_email' procedure\n", true);
        }
```

```
	if (mysql_stmt_execute(prep_stmt) != 0){
		print_stmt_error(prep_stmt, "An error occurred during the operation");
	} else{
		printf("\nEmail assegnata correttamente\n");
	}

	mysql_stmt_close(prep_stmt);
}


static void add_telefono(MYSQL *conn){

	MYSQL_STMT *prep_stmt;
	MYSQL_BIND param[3];

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

	char numero[46], paziente[46], tipo[46];

	printf("\nPaziente (tessera sanitaria): ");
	getInput(46, paziente, false);
	printf("\nTipologia ");
	printf("\n\t1) Cellulare");
	printf("\n\t2) Fisso\n");

	op = multiChoice("Seleziona tipologia di numero di telefono", options, 2);
	switch(op){
```

```
            case '1':
                strcpy(tipo, "cellulare");
                break;
            case '2':
                strcpy(tipo, "fisso");
                break;
            default:
                fprintf(stderr, "Invalid condition at %s: %d\n", __FILE__, __LINE__);
                abort();
        }

        printf("\nNumero di telefono: ");
        getInput(46, numero, false);

        if(!setup_prepared_stmt(&prep_stmt, "call aggiungi_recapito_tel(?, ?, ?)", conn)){
            finish_with_stmt_error(conn, prep_stmt, "Unable to init 'aggiungi_recapito_tel'
stmt\n", false);
        }

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

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

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

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



	if(mysql_stmt_bind_param(prep_stmt, param) != 0){
		finish_with_stmt_error(conn, prep_stmt, "Could not bind parameters for 'aggiungi_recapito_tel' procedure\n", true);
	}

	if (mysql_stmt_execute(prep_stmt) != 0){
		print_stmt_error(prep_stmt, "An error occurred during the operation");
	} else{
		printf("\nNumero di telefono assegnato correttamente\n");
	}

	mysql_stmt_close(prep_stmt);
}




static void add_personale_ad_esame(MYSQL *conn){

	MYSQL_STMT *p_stmt;
	MYSQL_BIND param[4];
	MYSQL_TIME date;

	char c_esame[46], paziente[46], day[3], month[3], year[5], personale[46];
```

```
        int examId;

        printf("\nEsame (ID): ");
        getInput(46, c_esame, false);
        printf("\nPaziente (tessera sanitaria): ");
        getInput(46, paziente, false);

        printf("\nData esame [DD/MM/YYYY]");
        printf("\nGiorno: ");
        getInput(3, day, false);
        printf("\nMese: ");
        getInput(3, month, false);
        printf("\nAnno: ");
        getInput(5, year, false);

        printf("\nMembro del personale da assegnare all'esame (CF): ");
        getInput(46, personale, false);

        examId = atoi(c_esame);



        if(!setup_prepared_stmt(&p_stmt, "call assegna_personale_ad_esame(?, ?, ?, ?)", conn)){
                finish_with_stmt_error(conn, p_stmt, "Unable to init 'assegna_personale_ad_esame'
stmt\n", false);
        }

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

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

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

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

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

// set struct MYSQL_TIME for date param
date.day = atoi(day);
date.month = atoi(month);
date.year = atoi(year);
date.time_type = MYSQL_TIMESTAMP_DATE;



if(mysql_stmt_bind_param(p_stmt, param) != 0){
        finish_with_stmt_error(conn, p_stmt, "Could not bind parameters for
'assegna_personale_ad_esame' procedure\n", true);
}

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

```
            print_stmt_error(p_stmt, "An error occurred during the operation");
        } else{
            printf("\nMembro del personale assegnato correttamente\n");
        }

        mysql_stmt_close(p_stmt);
}




static void search_by_tess_sanitaria(MYSQL *conn){

        MYSQL_STMT *p_stmt;
        MYSQL_BIND param[1];
        int status;
        char header[512];

        char paziente[46];

        printf("\nPaziente (tessera sanitaria): ");
        getInput(46, paziente, false);

        if(!setup_prepared_stmt(&p_stmt, "call search_patient(?)", conn)){
            finish_with_stmt_error(conn, p_stmt, "Unable to init 'search_patient' stmt\n", false);
        }

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

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

        if(mysql_stmt_bind_param(p_stmt, param) != 0){
                finish_with_stmt_error(conn, p_stmt, "Could not bind parameters for 'search_patient' procedure\n", true);
        }

        if (mysql_stmt_execute(p_stmt) != 0){
                print_stmt_error(p_stmt, "An error occurred during the operation");
                goto close;
        }

        sprintf (header, "\nPaziente con tessera sanitaria %s\n", paziente);

        do{
                // dump result
                dump_result_set(conn, p_stmt, header);

                status = mysql_stmt_next_result(p_stmt);
                if(status > 0){
                        finish_with_stmt_error(conn, p_stmt, "Unexpected condition", true);
                }
        }while(status == 0);

close:
        mysql_stmt_close(p_stmt);
```

```
}


void list_patients(MYSQL *conn){

        MYSQL_STMT *p_stmt;
        int status;
        char header[512];

        if (!setup_prepared_stmt(&p_stmt, "call list_patients()", conn)){
                finish_with_stmt_error(conn, p_stmt, "Unable to init 'list_patients' stmt\n", false);
        }

        // Nothing to bind!

        if (mysql_stmt_execute(p_stmt) != 0){
                print_stmt_error(p_stmt, "An error occurred during the execution of the query");
                goto close;
        }

        sprintf (header, "\nPazienti registrati\n");

        do{
                // dump result
                dump_result_set(conn, p_stmt, header);

                status = mysql_stmt_next_result(p_stmt);
                if(status > 0){
                        finish_with_stmt_error(conn, p_stmt, "Unexpected condition", true);
```

```
            }
        }while(status == 0);

close:
        mysql_stmt_close(p_stmt);
}



static void search_recapiti(MYSQL *conn){

        MYSQL_STMT *p_stmt;
        MYSQL_BIND param[1];
        int status;
        char header[512];

        char paziente[46];

        printf("\nPaziente (tessera sanitaria): ");
        getInput(46, paziente, false);

        if(!setup_prepared_stmt(&p_stmt, "call search_recapiti(?)", conn)){
                finish_with_stmt_error(conn, p_stmt, "Unable to init 'search_recapiti' stmt\n", false);
        }

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

        param[0].buffer_type = MYSQL_TYPE_VAR_STRING;
        param[0].buffer = paziente;
```

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


	if(mysql_stmt_bind_param(p_stmt, param) != 0){
		finish_with_stmt_error(conn, p_stmt, "Could not bind parameters for 'search_recapiti' procedure\n", true);
	}


	if (mysql_stmt_execute(p_stmt) != 0){
		print_stmt_error(p_stmt, "An error occurred during the operation");
		goto close;
	}


	sprintf (header, "\nRecapiti del paziente con tessera sanitaria %s\n", paziente);


	do{
		// dump result
		dump_result_set(conn, p_stmt, header);


		status = mysql_stmt_next_result(p_stmt);
		if(status > 0){
			finish_with_stmt_error(conn, p_stmt, "Unexpected condition", true);
		}
	}while(status == 0);


close:
	mysql_stmt_close(p_stmt);
}
```

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

	MYSQL_STMT *p_stmt;
	MYSQL_BIND param[1];
	int status;
	char header[512];
	bool first = true;

	char paziente[46];

	printf("\nPaziente (tessera sanitaria): ");
	getInput(46, paziente, false);

	if (!setup_prepared_stmt(&p_stmt, "call report_paziente (?)", conn)){
		finish_with_stmt_error(conn, p_stmt, "Unable to initialize 'report_paziente' stmt\n",
false);
	}

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

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

	if (mysql_stmt_bind_param(p_stmt, param) != 0){
		finish_with_stmt_error(conn, p_stmt, "Could not bind parameters for 'report_paziente'
stmt\n", true);
	}
```

```
	if (mysql_stmt_execute(p_stmt) != 0){
		print_stmt_error(p_stmt, "An error occurred while retrieving output.");
		goto exit;
	}

//MULTIPLE RESULT SETS!!!
	do{

		if(conn->server_status & SERVER_PS_OUT_PARAMS){
			goto next;
		}

		if(first){
			sprintf(header, "\nStorico esami eseguiti dal paziente con tessera sanitaria %s\n", paziente);
			first = false;
		}else{
			sprintf(header, "\nPrenotazioni attive per il paziente con tessera sanitaria %s\n", paziente);
		}

		dump_result_set(conn, p_stmt, header);

	next:
		status = mysql_stmt_next_result(p_stmt);
		if (status > 0){		// > 0 error, 0 keep looking, -1 finished
			finish_with_stmt_error(conn, p_stmt, "Unexpected condition", true);
		}
	}while(status == 0);
```

```
exit:
	mysql_stmt_close(p_stmt);
}



static void inserisci_risultato_esame(MYSQL *conn){

	MYSQL_STMT *p_stmt;
	MYSQL_BIND param[5];
	MYSQL_TIME date;

	char c_esame[46], paziente[46], day[3], month[3], year[5], parametro[46], c_valore[12];
	int examId;
	float valore;

	printf("\nEsame (ID): ");
	getInput(46, c_esame, false);
	printf("\nPaziente (tessera sanitaria): ");
	getInput(46, paziente, false);

	printf("\nData esame [DD/MM/YYYY]");
	printf("\nGiorno: ");
	getInput(3, day, false);
	printf("\nMese: ");
	getInput(3, month, false);
	printf("\nAnno: ");
	getInput(5, year, false);
```

```
        printf("\nParametro di cui si vuole assegnare il valore: ");
        getInput(46, parametro, false);
        printf("\nValore parametro: ");
        getInput(12, c_valore, false);

        examId = atoi(c_esame);
        valore = strtof(c_valore, NULL);

        if(!setup_prepared_stmt(&p_stmt, "call inserisci_risultato_esame(?, ?, ?, ?, ?)", conn)){
                finish_with_stmt_error(conn, p_stmt, "Unable to init 'inserisci_risultato_esame' stmt\n", false);
        }

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

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

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

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

        param[3].buffer_type = MYSQL_TYPE_VAR_STRING;
```

```
        param[3].buffer = parametro;
        param[3].buffer_length = strlen(parametro);

        param[4].buffer_type = MYSQL_TYPE_FLOAT;
        param[4].buffer = &valore;
        param[4].buffer_length = sizeof(valore);

        // set struct MYSQL_TIME for date param
        date.day = atoi(day);
        date.month = atoi(month);
        date.year = atoi(year);
        date.time_type = MYSQL_TIMESTAMP_DATE;



        if(mysql_stmt_bind_param(p_stmt, param) != 0){
                finish_with_stmt_error(conn, p_stmt, "Could not bind parameters for
'inserisci_risultato_esame' procedure\n", true);
        }

        if (mysql_stmt_execute(p_stmt) != 0){
                print_stmt_error(p_stmt, "An error occurred during the operation");
        } else{
                printf("\nValore del parametro assegnato correttamente\n");
        }

        mysql_stmt_close(p_stmt);
}
```

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

	MYSQL_STMT *p_stmt;
	MYSQL_BIND param[1];
	int status;
	char header[512];
	int count = 1;

	char c_codiceP[12];
	int codiceP;

	printf("\nCodice di prenotazione: ");
	getInput(12, c_codiceP, false);

	codiceP = atoi(c_codiceP);

	if (!setup_prepared_stmt(&p_stmt, "call report_risultati_prenotazione (?)", conn)){
		finish_with_stmt_error(conn, p_stmt, "Unable to initialize 
'report_risultati_prenotazione' stmt\n", false);
	}

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

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

```
	if (mysql_stmt_bind_param(p_stmt, param) != 0){
		finish_with_stmt_error(conn, p_stmt, "Could not bind parameters for 'report_risultati_prenotazione' stmt\n", true);
	}

	if (mysql_stmt_execute(p_stmt) != 0){
		print_stmt_error(p_stmt, "An error occurred while retrieving output.");
		goto exit;
	}

//MULTIPLE RESULT SETS!!!
	do{

		if(conn->server_status & SERVER_PS_OUT_PARAMS){
			goto next;
		}

		sprintf(header, "\nRisultati esame n° %d con codice di prenotazione %d\n", count, codiceP);
		dump_result_set(conn, p_stmt, header);
		count++;

	next:
		status = mysql_stmt_next_result(p_stmt);
		if (status > 0){		// > 0 error, 0 keep looking, -1 finished
			finish_with_stmt_error(conn, p_stmt, "Unexpected condition", true);
		}
	}while(status == 0);

exit:
```

```
	mysql_stmt_close(p_stmt);
}




//MENU
static void manage_patients(MYSQL *conn){

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

	while(true){
		printf("\033[2J\033[H");
		printf("*** Cosa desideri fare? ***\n\n");
		printf("1) Aggiungi un nuovo paziente\n");
		printf("2) Aggiungi un'email di un paziente\n");
		printf("3) Aggiungi un numero di telefono di un paziente\n");
		printf("4) Cerca paziente\n");
		printf("5) Lista tutti i pazienti\n");
		printf("6) Cerca i recapiti di un paziente\n");
		printf("7) Report paziente (storico esami e prenotazioni)\n");
		printf("8) Indietro\n");

		op = multiChoice("Select an option", options, 8);

		switch(op){
			case '1':
```

```
            add_patient(conn);
            break;

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

        case '3':
            add_telefono(conn);
            break;

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

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

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

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

        case '8':
            printf("\nPress 'Enter' to continue\n");
```

```
                        return;

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

        getchar(); // read the '\n'
    }
}



static void cancella_prenotazione(MYSQL *conn){

    MYSQL_STMT *p_stmt;
    MYSQL_BIND param[3];
    MYSQL_TIME date;

    char c_esame[46], paziente[46], day[3], month[3], year[5];
    int examId;

    printf("\nEsame (ID): ");
    getInput(46, c_esame, false);
    printf("\nPaziente (tessera sanitaria): ");
    getInput(46, paziente, false);

    printf("\nData esame [DD/MM/YYYY]");
    printf("\nGiorno: ");
```

```
        getInput(3, day, false);
        printf("\nMese: ");
        getInput(3, month, false);
        printf("\nAnno: ");
        getInput(5, year, false);

        examId = atoi(c_esame);



        if(!setup_prepared_stmt(&p_stmt, "call cancella_prenotazione(?, ?, ?)", conn)){
                finish_with_stmt_error(conn, p_stmt, "Unable to init 'cancella_prenotazione' stmt\n",
false);
        }

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

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

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

        param[2].buffer_type = MYSQL_TYPE_DATE;
        param[2].buffer = (char *)&date;
        param[2].buffer_length = sizeof(date);
```

```
	// set struct MYSQL_TIME for date param
	date.day = atoi(day);
	date.month = atoi(month);
	date.year = atoi(year);
	date.time_type = MYSQL_TIMESTAMP_DATE;



	if(mysql_stmt_bind_param(p_stmt, param) != 0){
		finish_with_stmt_error(conn, p_stmt, "Could not bind parameters for 'cancella_prenotazione' procedure\n", true);
	}

	if (mysql_stmt_execute(p_stmt) != 0){
		print_stmt_error(p_stmt, "An error occurred during the operation");
	}

	mysql_stmt_close(p_stmt);
}









//MENU
static void manage_exams(MYSQL *conn){

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

```
while(true){
        printf("\033[2J\033[H");
        printf("*** Cosa desideri fare? ***\n\n");
        printf("1) Prenota un esame\n");
        printf("2) Aggiungi diagnosi\n");
        printf("3) Cerca esame per codice di prenotazione\n");
        printf("4) Visualizza gli esami disponibili\n");
        printf("5) Assegna un membro del personale ad un esame\n");
        printf("6) Inserisci risultato di un esame (singolo parametro)\n");
        printf("7) Visualizza risultati esami per codice di prenotazione\n");
        printf("8) Cancella prenotazione di un esame\n");
        printf("9) Indietro\n");

        op = multiChoice("Select an option", options, 9);

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

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

                case '3':
                        search_by_codiceP(conn);
                        break;
```

```
            case '4':
                list_exams(conn);
                break;

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

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

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

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

            case '9':
                printf("\nPress 'Enter' to continue\n");
                return;

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

```
            getchar(); // read the '\n'
        }
}



//MENU
static void search_staff_members (MYSQL *conn){

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

        while(true){    // while back option is selected
                printf("\033[2J\033[H");        // clean screen
                printf("*** Cosa desideri fare? ***\n\n");
                printf("1) Cerca membri del personale per ospedale\n");
                printf("2) Cerca membri del personale per reparto\n");
                printf("3) Indietro\n");

                op = multiChoice("Select an option", options, 7);
                switch(op){
                        case '1':
                                select_personale_by_hosp(conn);
                                break;
                        case '2':
                                select_personale_by_rep(conn);
                                break;
                        case '3':
                                printf("\nPress 'Enter' to continue\n");
```

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

            getchar();
        }
}



void run_as_personaleCUP(MYSQL *conn){

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

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

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

        //change user on DBMS
        if(mysql_change_user(conn, conf.db_username, conf.db_password, conf.database)){
            fprintf(stderr, "mysql_change_user() failed, Error: %s\n", mysql_error(conn));
            exit(EXIT_FAILURE);
        }
```

```
while(true){
        printf("\033[2J\033[H");
        printf("*** Cosa desideri fare? ***\n\n");
        printf("1) Gestisci pazienti\n");
        printf("2) Gestisci esami\n");
        printf("3) Visualizza le strutture mediche\n");
        printf("4) Cerca membri del personale\n");
        printf("5) Esci\n");

        op = multiChoice("Select an option", options, 5);

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

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

                case '3':
                        list_medical_structures(conn);
                        break;

                case '4':
                        search_staff_members(conn);
                        break;
```

```
                        case '5':
                                return;

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

                getchar(); // read the '\n'
        }
}
```

login.json:

```
{
  "host": "localhost",
  "username": "login",
  "password": "loginUser0!",
  "port": 3306,
  "database": "ASL_db"
}
```

amministratore.json:

```
{
  "host": "localhost",
  "username": "amministratore",
  "password": "Amministratore0!",
  "port": 3306,
  "database": "ASL_db"
}
```

personaleCUP.json:

```
{
  "host": "localhost",
  "username": "personaleCUP",
  "password": "personaleCUP0!",
  "port": 3306,
  "database": "ASL_db"
}
```