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

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 2](#_Toc606296459)

[2. Analisi dei Requisiti 4](#_Toc1289394997)

[3. Progettazione concettuale 10](#_Toc2081466291)

[4. Progettazione logica 18](#_Toc2147004904)

[5. Progettazione fisica 27](#_Toc518560220)

[Appendice: Implementazione 77](#_Toc403811585)

# Descrizione del Minimondo

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46 | 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. |

# 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

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46 | 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. |

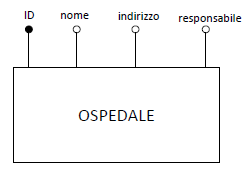
|  |
| --- |
| **Frasi relative ai primari** |
| Tra il personale del reparto, nel caso dei medici primari del reparto è noto l’elenco delle specializzazioni.  I primari degli ospedali e dei laboratori vanno individuati all’interno del personale medico, che può essere gestito unicamente dagli amministratori del sistema. |

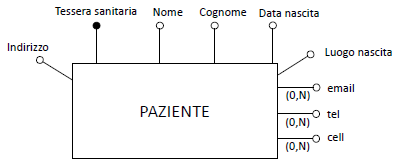
|  |
| --- |
| **Frasi relative ai volontari** |
| […] mentre per il personale volontario è noto il nome dell’associazione di appartenenza, se disponibile. |

# Progettazione concettuale

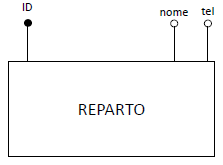
## Costruzione dello schema E-R

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

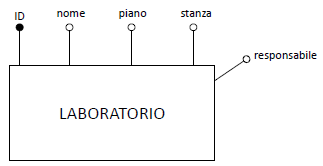
****…Gli ospedali della ASL sono identificati da un codice numerico e sono caratterizzati da un nome, un indirizzo e dal nome di un primario. Anche la gestione degli ospedali è in capo agli amministratori del sistema…

…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…

****

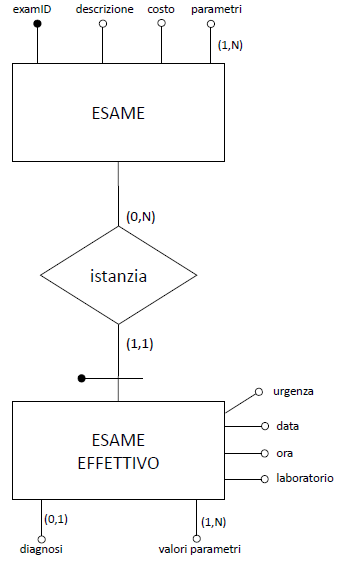
…I laboratori che eseguono gli esami sono identificati da un codice univoco all’interno di un ospedale della ASL e sono caratterizzati dal nome del laboratorio, dal piano di ubicazione e dal numero di stanza. Anche per i laboratori è prevista la designazione di un primario…

****

…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…

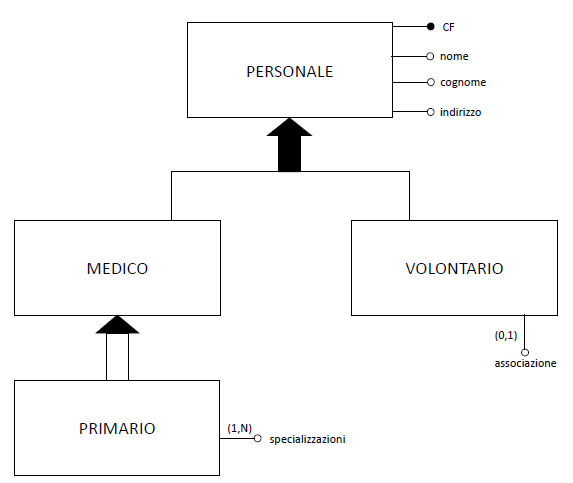
…Per ogni esame prenotato da parte di un paziente si vuole memorizzare la data e l’ora dell’esame il laboratorio presso cui è eseguito, il costo del ticket e se tale esame è prescritto con urgenza…

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…

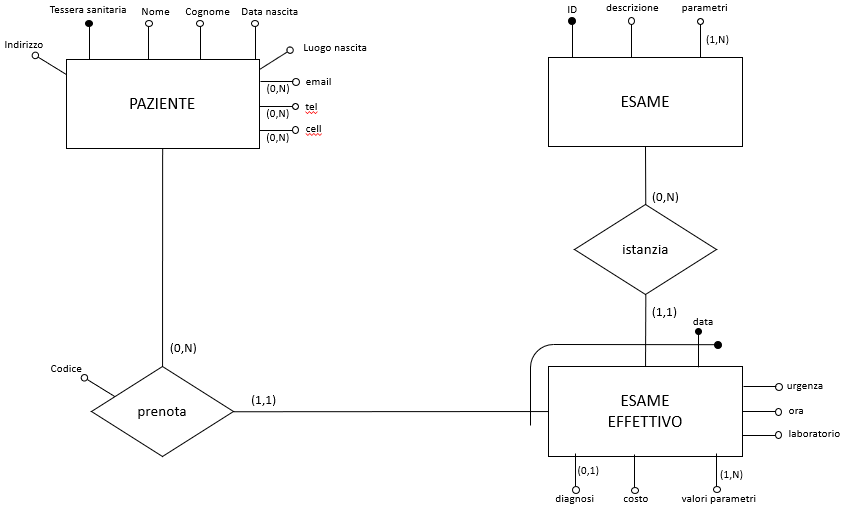


Si è proseguito nella costruzione dello schema E-R individuando, tramite un’analisi delle specifiche, le associazioni cui partecipano le diverse entità precedentemente rappresentate.

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:

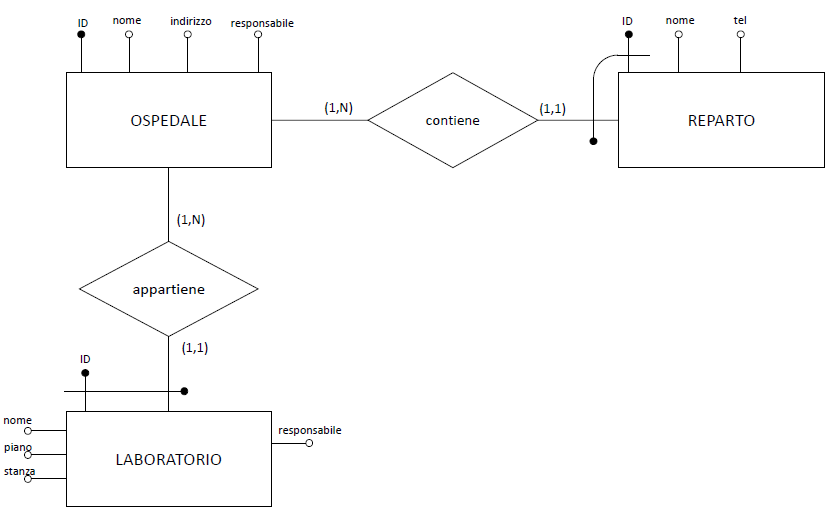
…Si tenga presente che ogni paziente può effettuare più prenotazioni dello stesso esame in date diverse. Si noti inoltre che lo stesso esame non può essere ripetuto nella stessa data dallo stesso paziente…



…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*.

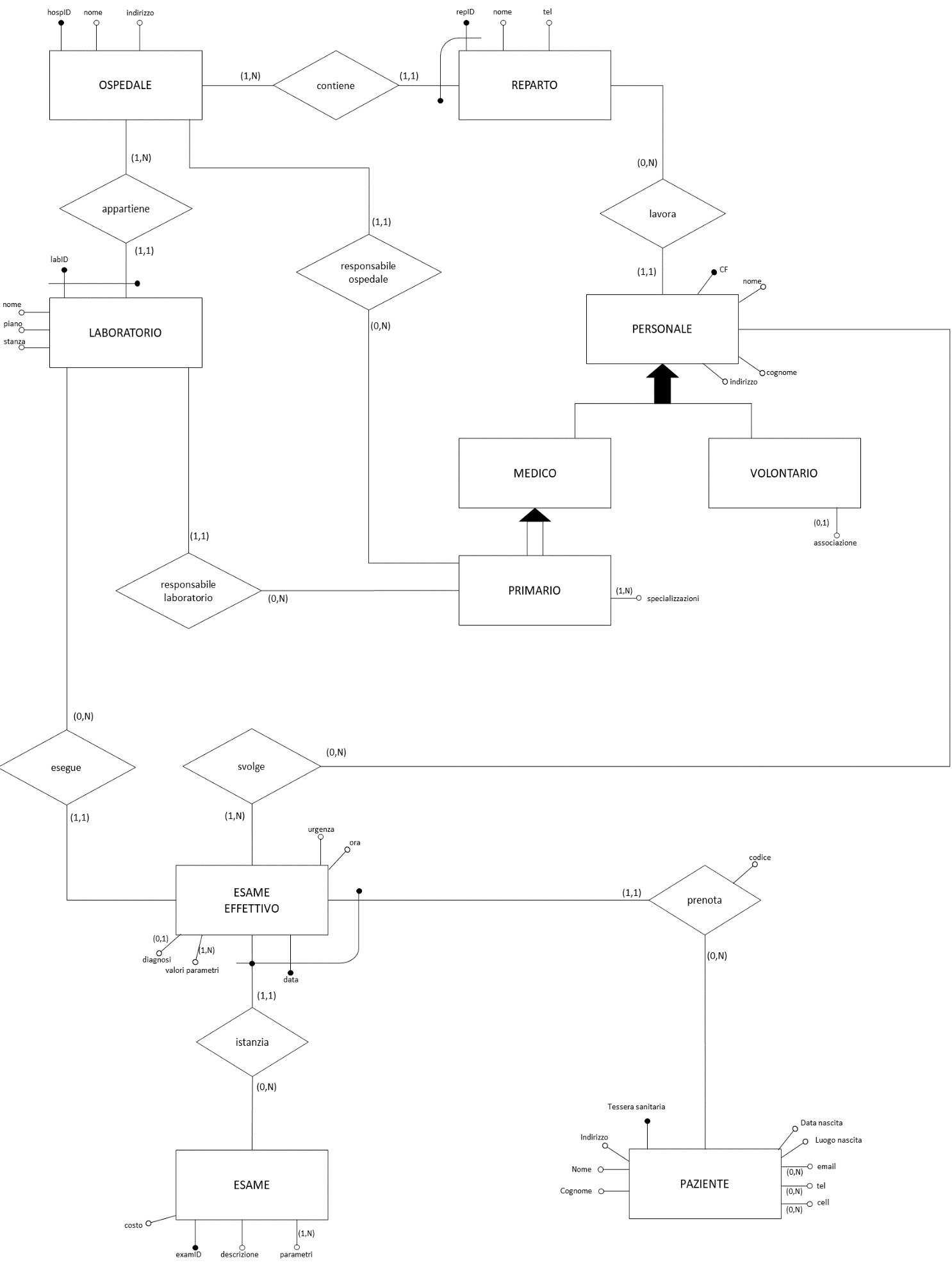
Analogamente, l’attributo “laboratorio” dell’entità *ESAME EFFETTIVO* è stato sostituito con l’associazione “esegue” tra le entità *ESAME* *EFFETTIVO* e *LABORATORIO*.

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:



## Regole aziendali

## Ogni primario deve essere responsabile di almeno uno tra ospedali e laboratori;

1. Ogni esame effettivo deve avere tanti valori numerici quanti sono i parametri dell’entità esame;
2. 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 |  |

# Progettazione logica

## Volume dei dati

|  |  |  |
| --- | --- | --- |
| **Concetto nello schema** | **Tipo[[1]](#footnote-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 |

## 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:

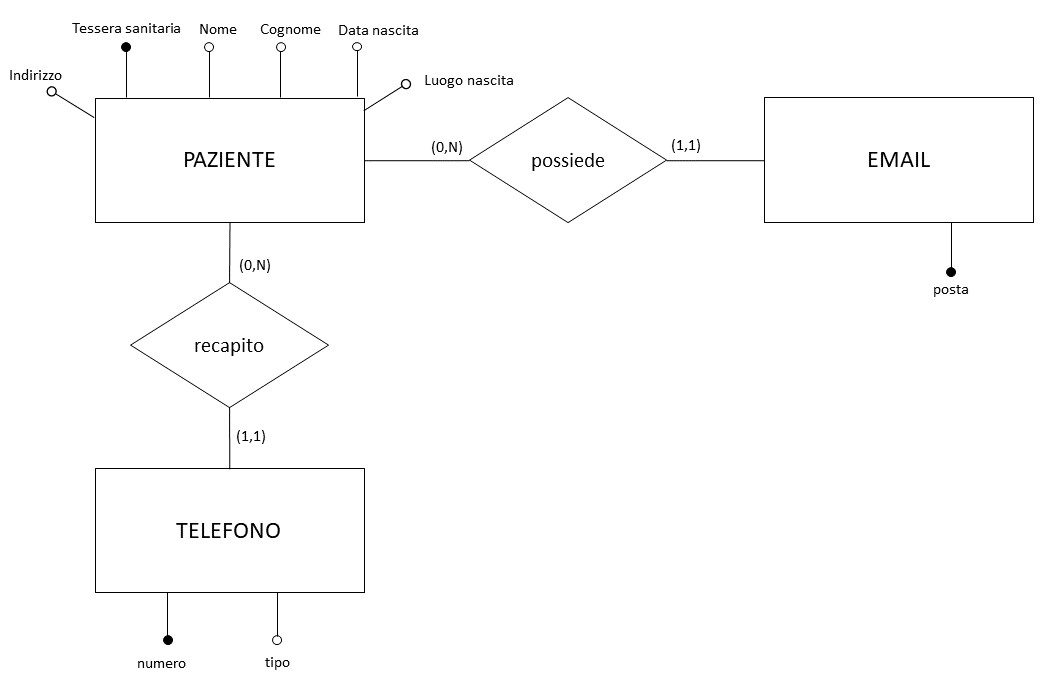
1 \* 5 = 5 accessi/settimana

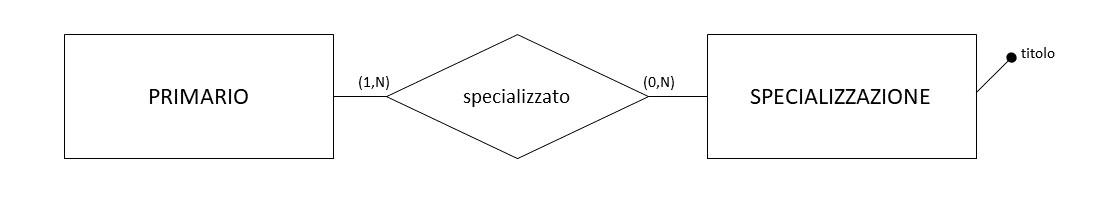
## Ristrutturazione dello schema E-R

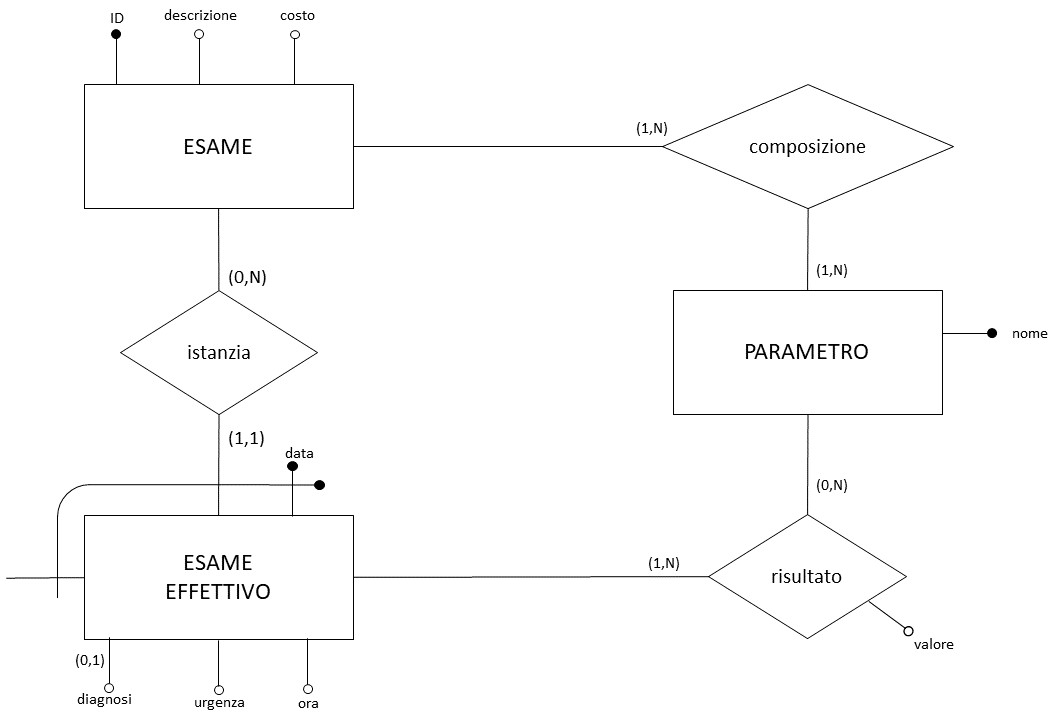
Lo schema E-R è stato ristrutturato innanzitutto rimuovendo gli attributi multivalore e traformandoli in entità. In particolare, tale operazione ha riguardato:

* 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:





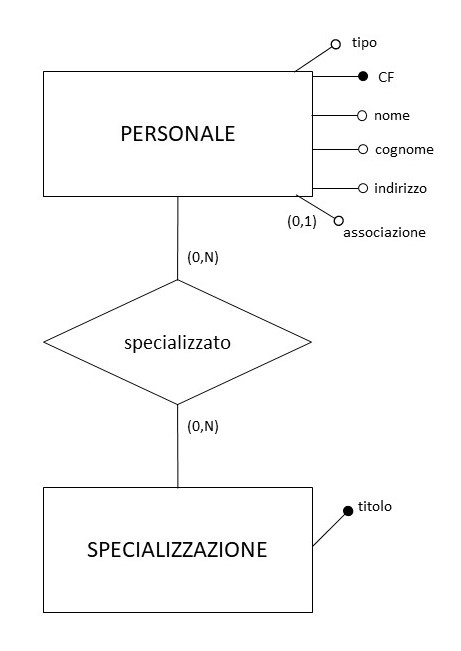


Successivamente, si è proseguito nella ristrutturazione dello schema E-R rimuovendo le generalizzazioni. Si è dapprima deciso di accorpare l’entità figlia *PRIMARIO* nell’entità padre *MEDICO,* introducendo l’attributo “tipo” per distinguere un semplice medico da un primario. Inoltre, l’entità *MEDICO* 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:

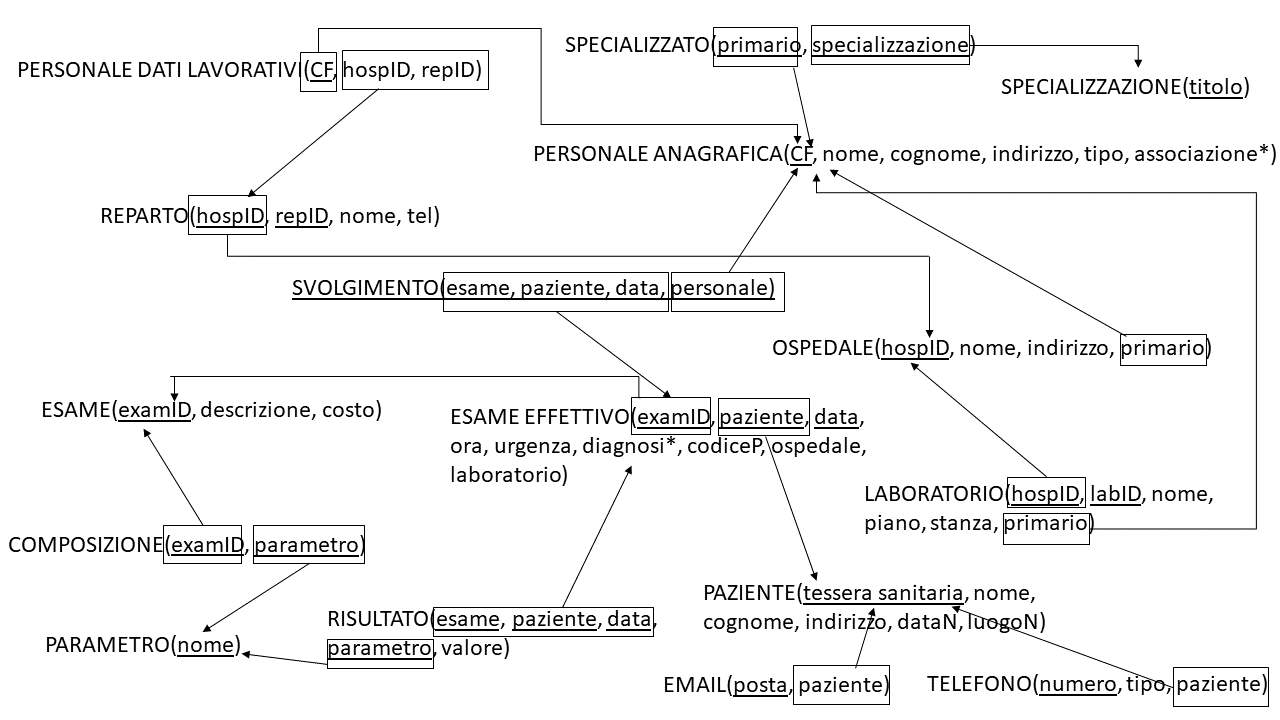
* Solo un membro del personale di tipo primario può essere responsabile di un ospedale e/o di un laboratorio;
* Solo per un membro del personale di tipo primario bisogna indicare l’elenco delle specializzazioni;
* Solo per un membro del personale di tipo volontario è possibile indicare l’associazione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

paziente examID data 🡪 A

dove A ∈ {“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.

# 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]](#footnote-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 |
| 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]](#footnote-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 |

|  |  |
| --- | --- |
| **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\_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"

}

1. Indicare con E le entità, con R le relazioni [↑](#footnote-ref-1)
2. PK = primary key, NN = not null, UQ = unique, UN = unsigned, AI = auto increment. È ovviamente possibile specificare più di un attributo per ciascuna colonna. [↑](#footnote-ref-2)
3. IDX = index, UQ = unique, FT = full text, PR = primary. [↑](#footnote-ref-3)