Basi di Dati

Progetto A.A. 2022/2023

SISTEMA DI PRENOTAZIONE DI ESAMI IN UNA ASL

0280932

Luca Molinaro

Indice

1. Descrizione del Minimondo

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

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

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

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. 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.

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

risultati di un insieme di esami associati ad una prenotazione, e/o mostrare lo storico di tutti gli esami svolti da un determinato paziente dalla sua registrazione nel sistema.

2. Analisi dei Requisiti

Identificazione dei termini ambigui e correzioni possibili

Linea	Termine	Nuovo termine	Motivo correzione
30	Personale	Personale medico	Il termine viene utilizzato per indicare il personale del CUP.
7	Recapiti	Recapiti telefonici e di posta elettronica	Significato della frase: il numero di recapiti telefonici è arbitrario ma la tipologia di essi può essere solo telefonico o di posta elettronica.
21	Responsabile	Primario	Il responsabile del laboratorio fa parte del personale medico.
12	Numerico	Di esame medico	Potrebbe essere confuso con il codice univoco di laboratorio (che potrebbe anch'esso essere numerico)
19	Univoco	Di laboratorio, univoco	Si suppone che tutti i codici identificativi siano univoci nel loro campo, perciò va specificato a cosa il codice è riferito.

Specifica disambiguata

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

Ciascun paziente è identificato da un codice di tessera sanitaria ed è caratterizzato da un nome, un cognome, la data ed il luogo di nascita, un indirizzo di residenza ed un insieme arbitrario di recapiti telefonici e di posta elettronica. La gestione dei pazienti è in capo al personale del CUP, che può gestirne 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 di esame medico e sono caratterizzati dalla descrizione di esame medico (ad esempio Radiografia, ecc.). L'insieme degli esami disponibili presso la ASL è gestito 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.

I laboratori che eseguono gli esami sono identificati da un codice di laboratorio, univoco all'interno di un ospedale della ASL e sono caratterizzati dal nome del laboratorio, dal piano di ubicazione e dal

numero di stanza. Per i laboratori è prevista la designazione di un primario.

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.

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

Glossario dei Termini

Termine	Descrizione	Sinonimi	Collegamenti
Esame	Visita medica svolta dal personale medico sul paziente.	Visita	Paziente, Prenotazione, Personale medico.
Prenotazione	Il paziente, tramite il sistema, fissa una data in cui eseguire un esame.	Esame	Paziente, Esame.
Laboratorio	Luogo in cui vengono svolti gli esami, sono parte di un ospedale della ASL.	Ospedale	Primario.
Personale medico	Effettuano gli esami.	Medico, Primario.	Esame, Laboratorio.
Paziente	Prenota e viene sottoposto agli esami.	Utente, Prenotato.	Prenotazione, Esame.

Raggruppamento dei requisiti in insiemi omogenei

Frasi relative a Esame
Si vuole realizzare il sistema informativo di gestione di un sistema di prenotazioni di esami medici [...]. 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 è gestito dagli amministratori del sistema. Ciascun esame è associato ad un insieme di valori numerici, riportanti i risultati dei parametri legati allo specifico esame.Inoltre, per ciascun esame è possibile inserire da parte del personale medico una diagnosi testuale. [...] Per ogni 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.

Frasi relative a Prenotazione
Si vuole realizzare il sistema informativo di gestione di un sistema di prenotazioni di esami medici [...]. La gestione dei pazienti è in capo al personale del CUP, che può gestirne 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. [...] 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. [...] 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.

Frasi relative a Laboratorio

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. Per i laboratori è prevista la designazione di un primario.
Per ogni prenotazione di un esame da parte di un paziente si vuole memorizzare [...] il laboratorio presso cui è eseguito.

Frasi relative a Personale medico

Per ciascun esame è possibile inserire da parte del personale medico una diagnosi testuale. [...]

Per i laboratori è prevista la designazione di un primario. [...]

Gli amministratori possono generare dei report che mostrano per ciascun membro del personale medico quanti esami (e quali esami) ha svolto, su base mensile e/o annuale.

Frasi relative a Paziente

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 telefonici e di posta elettronica. La gestione dei pazienti è in capo al personale del CUP, che può gestirne nella sua interezza l'anagrafica e le prenotazioni degli esami. [...]

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. [...]

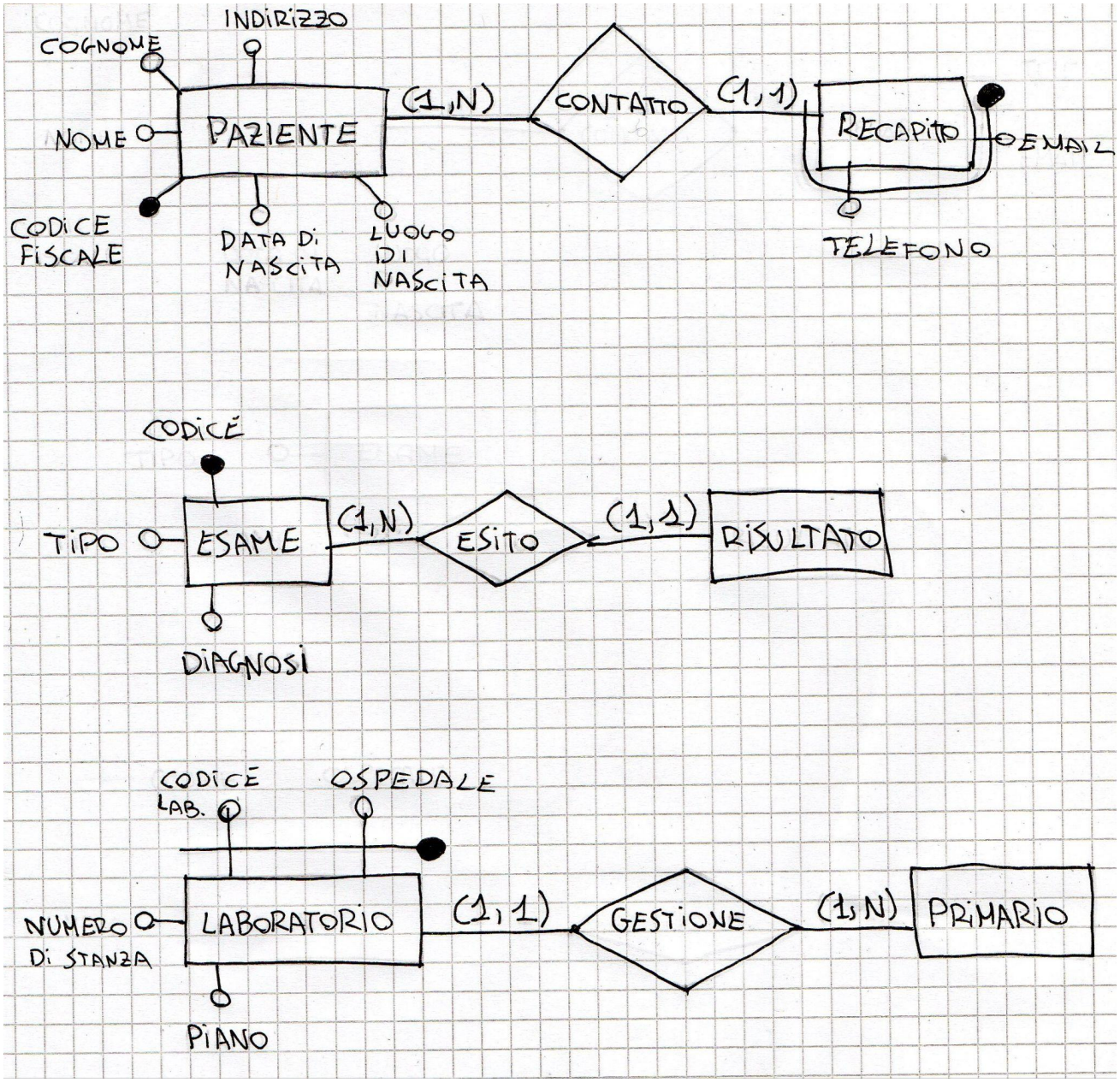
Il personale del CUP [...] ha la possibilità di [...] mostrare lo storico di tutti gli esami svolti da un determinato paziente dalla sua registrazione nel sistema.

3. Progettazione concettuale

Costruzione dello schema E-R

Per costruire lo schema E-R è stata inizialmente seguito una approccio “Bottom-Up”, generando degli schemi parziali per alcuni dei concetti identificati nell’analisi dei requisiti.

sono perciò state identificate le Entità: Paziente (in riferimento al quale è stato scritto erroneamente “codice fiscale” anziché “codice di tessera sanitaria”), Esame, Laboratorio e Primario, generando i seguenti schemi:



Nello specifico, poiché il numero di recapiti è arbitrario, si è deciso di renderlo un'entità collegata al paziente tramite una relazione in cui le cardinalità sono (1,N) da parte del paziente, dal momento che deve averne almeno uno, e (1,1) dal lato del recapito poiché esso deve essere di uno e un solo paziente.

Per ciò che riguarda gli esami, si è usata la stessa idea già usata per la relazione Paziente-recapito, poiché anche i risultati possono essere in numero arbitrario ma riferiti allo stesso esame.
Nel diagramma riportato ci sono delle differenze rispetto allo schema finale:

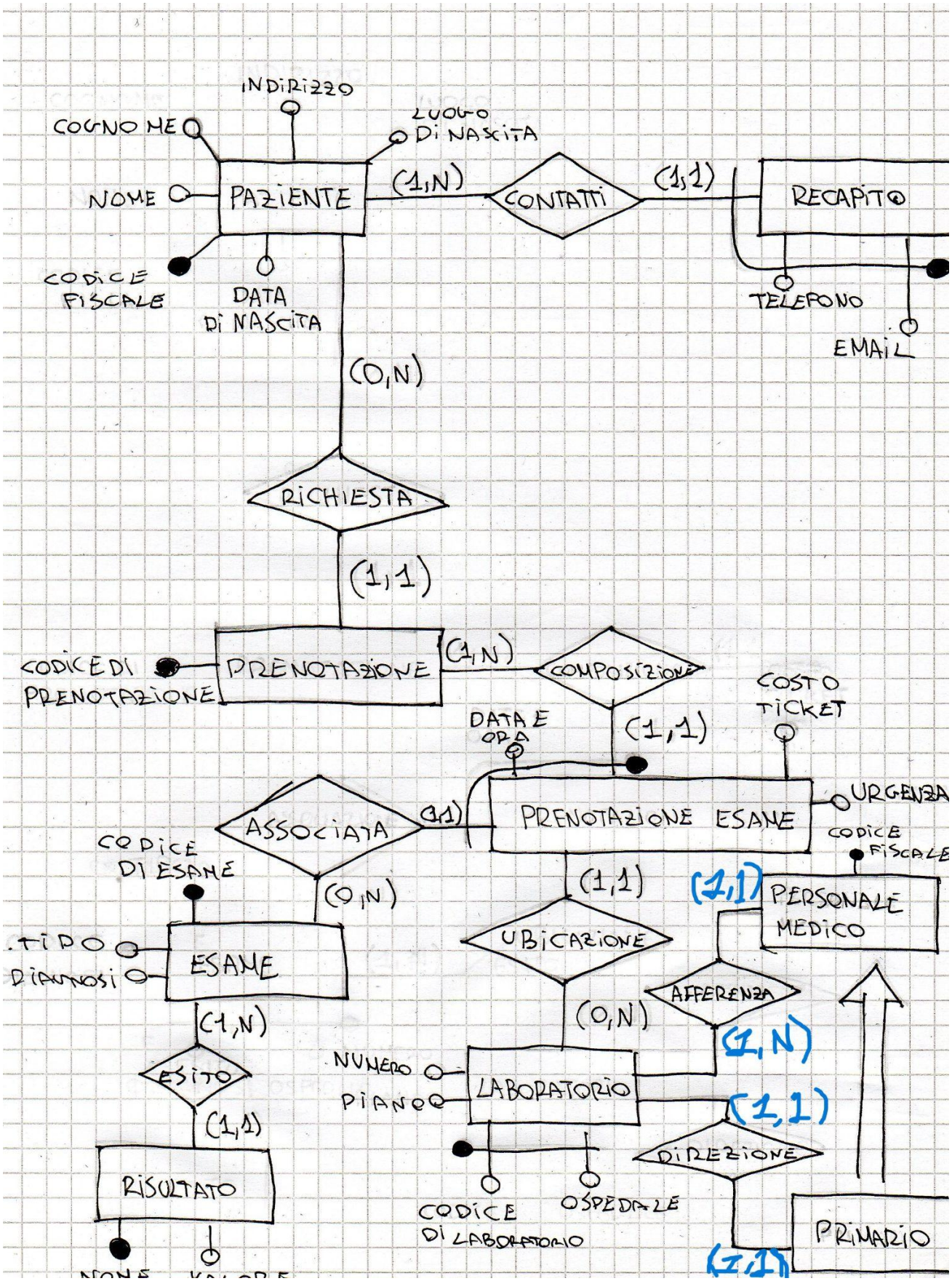
- mancano gli attributi dell'entità "risultato" (nome, valore).
- L'attributo "codice" dell'esame è stato poi sostituito con "codice di esame", come indicato nella disambiguazione dei termini.

Infine lo schema riguardante il Laboratorio è stato maggiormente modificato: infatti era stato dimenticato di aggiungere un entità "Personale medico", che è una generalizzazione del primario, con attributi "Nome" , "Cognome", "Numero esami" e l'identificativo "Codice fiscale".
In fase di integrazione è stata corretta questa parte del diagramma, è stato aggiunto un attributo identificativo del personale medico e del primario (Codice fiscale) ed è stata modificato la cardinalità del primario nella relazione "Gestione" (in seguito chiamata "Direzione") da (1,N) a (1,1), dicendo cioè che un Primario può essere responsabile di un solo laboratorio.
A questi schemi, procedendo raffinando come descritto e poi a macchia d'olio, è stata aggiunta in fase di integrazione un'entità relativa alle prenotazioni, tenendo conto delle seguenti considerazioni:

- Un paziente può prenotare più esami con lo stesso codice;
- Si vogliono memorizzare informazioni quali data e ora, costo del ticket e urgenza riguardo una prenotazione di uno specifico esame, indipendentemente da quanti altri esami vengano prenotati usando lo stesso codice;
- L'esame è un'entità a sé che non ha a che vedere con quando o dove un determinato paziente lo svolge;

Perciò è stata inserita un'entità intermedia tra "Prenotazione" (che rappresenta la prenotazione di uno o più esami con un codice) ed "Esame", chiamata "Prenotazione esame", che ha come attributi "Data e ora", "Costo ticket" e "Urgenza".

Integrazione finale



*Nel diagramma sopra riportato è presente un errore: la diagnosi non è un attributo dell'esame ma sono legati da una relazione, che chiameremo "Riferita" con cardinalità: "Esame--(0,1)--><Riferita><--(1,1)--Diagnosi"

Regole aziendali

- Un paziente non può ripetere lo stesso esame nello stesso giorno.
- Due pazienti non possono effettuare lo stesso esame nello stesso laboratorio nella stessa data e ora.

Dizionario dei dati

Entità	Descrizione	Attributi	Identificatori
Paziente	Informazioni sul paziente che effettua una prenotazione e viene sottoposto a esami.	Nome, Cognome, Data di nascita, Luogo di nascita	Codice tessera sanitaria
Recapito	Indirizzo e-mail e i numeri di telefono e di cellulare al quali il paziente può essere contattato.		Cellulare, Telefono, e-mail, Contatti
Prenotazione	La prenotazione di uno o più esami, effettuata da un paziente.		Codice di prenotazione
Prenotazione Esame	La prenotazione riferita ad un singolo esame medico.	Costo ticket, Urgenza	Data e ora, Codice Prenotazione, Ubicazione, Svolgimento
Primario	Unità del personale medico con responsabilità su un laboratorio.	Nome, Cognome, Numero esami	Codice Fiscale
Personale Medico	Si occupano di svolgere gli esami e scrivere eventuali diagnosi.	Nome, Cognome, Numero esami	Codice Fiscale
Laboratorio	Luogo in cui si svolgono gli esami.	Nome, Piano , Numero di stanza	Codice di laboratorio, Ospedale
Esame	Visita medica svolta dal personale medico sul paziente allo scopo di ottenere una diagnosi.	Descrizione di esame medico	Codice di esame medico
Risultato	Valore numerico indicativo di un parametro del paziente.	Valore	Nome
Diagnosi	Testo inserito dal personale medico in relazione ad un esame contenente le conclusioni tratte dai risultati.	Testo	Riferita

4. Progettazione logica

Volume dei dati

I volumi dei dati e la frequenza delle operazioni seguenti sono derivati dalle seguenti supposizioni:

- il numero di recapiti per paziente sia di 1,2;
- ogni laboratorio possa eseguire un esame ogni ora restando aperto 8 ore al giorno;
- la ASL sia composta da una decina strutture ospedaliere, alcuni più piccole di altre ma con in media 20 laboratori;
- alcuni laboratori molto specializzati non sono utilizzati continuamente, perciò assumiamo che al giorno l'80% dei laboratori sia utilizzato per esami;
- in media i pazienti prenotino 2 esami durante una chiamata al CUP;
- in media ci siano 35 risultati per esame, sebbene questo numero può variare estremamente in dipendenza dal tipo di esame;
- la lista degli esami disponibili non vari molto nel tempo;
- si mantengono le informazioni relative alle prenotazioni e ai risultati degli esami per 10 anni, supponendo che oltre quella data non abbiano più valore a livello medico;
- ogni anno siano 2.000 le persone che si trasferiscono nel territorio di competenza della ASL.

Concetto nello schema	Tipo	Volume atteso
Paziente	E	400.000
Recapito	E	480.000
Prenotazione	E	2.336.000
Prenotazione Esame	E	4.672.000
Primario	E	200
Personale Medico	E	2.500
Laboratorio	E	200
Esame	E	100
Risultato	E	3.500
Contatti	R	480.000
Richiesta	R	2.336.000
Composizione	R	4.672.000
Associata	R	4.672.000
Esito	R	3.500
Ubicazione	R	4.672.000
Afferenza	R	2.500
Direzione	R	200

Tavola delle operazioni

Cod.	Descrizione	Frequenza attesa
CUP1	Registra paziente	5/giorno
CUP2	Lista esami	1300/giorno
CUP3	Registra prenotazione	1280/giorno
CUP4	Report prenotazioni	5/giorno
CUP5	Storico esami	10/mese
AM1	Registra esame	10/anno
AM2	Report personale	1/mese
PM1	Registra risultati	1280/giorno
L1	Login	1500/giorno

Costo delle operazioni

CUP1 - Registra paziente:

- Paziente: 800.000
- Contatti: 960.000
- Recapito: 960.000

Costo totale: 2.720.000

Accessi/giorno: 13.600.000

CUP2 - Lista Esami:

- Esame: 100
- Esito: 3.500
- Risultato: 3.500

Costo totale: 7.100

Accessi/giorno: 10.650.000

CUP3 - Registra Prenotazione:

- Prenotazione: 4.672.000
- Composizione: 9.344.000
- Prenotazione Esame: 9.344.000
- Ubicazione: 9.344.000

Costo totale: 32.704.000
Accessi/giorno: 42.515.200.000

Accorpando le entità “Prenotazione” e “Prenotazione esame”, aggiungendo all’entità “Prenotazione esame” l’attributo “Codice Prenotazione”:

- Prenotazione Esame: 9.344.000
- Ubicazione: 9.344.000

Accessi/giorno: 24.294.400.000
Risparmiando 18.220.800.000 accessi.

Per le operazioni che seguono si considerano accorpate le entità.

CUP4 - Report prenotazioni:

- Prenotazione Esame: 4.672.000
- Associata: 4.672.000
- Esame: 100
- Esito: 3.500
- Risultato: 3.500

Costo totale: 9.351.100
Accessi/giorno: 46.755.500

CUP5 - Storico Esami:

- Paziente: 400.000
- Richiesta: 4.672.000
- Prenotazione Esame: 4.672.000
- Associata: 4.672.000
- Esame: 100
- Esito: 3.500
- Risultato: 3.500

Costo totale: 14.423.100

Accessi/mese: 144.231.000

Accessi/giorno: 4.807.700

AM1 Registra Esame:

- Esame: 200
- Esito: 7.000
- Risultato: 7000

Costo totale: 14.200

Accessi/anno: 142.000

Accessi/giorno: 390

AM2 - Report Personale:

- Personale Medico: 2.500

Accessi/mese: 2.500

Accessi/giorno: 83

PM1 - Registra Risultati:

- Prenotazione Esame: 4.672.000
- Associata: 9.344.000
- Esame: 200
- Esito: 7.000
- Risultato: 7.000

Costo totale: 14.030.200

Accessi/giorno: 17.958.656.000

L1 - Login:

Il numero approssimativo viene fuori supponendo che venga fatto un login per ogni esame inserito da parte del personale medico e che gli utenti dell'amministrazione e del CUP lo facciano una sola volta al giorno.

Accessi/giorno: 1500

Ristrutturazione dello schema E-R

La prima considerazione fatta in fase di ristrutturazione dello schema E-R è stato quello relativo all'accorpamento delle entità "Prenotazione" e "Prenotazione Esame".
Tale accorpamento permette di quasi dimezzare il numero di accessi dell'operazione più costosa ("Registra prenotazione") e diminuisce gli accessi, anche se in maniera minore, anche in operazioni quali "Report Prenotazioni" e "Storico Esami".

Per ciò che riguarda gli attributi duplicati e derivabili: L'attributo "Numero Esami" dell'entità "Personale Medico" e "Primario" è derivabile, ed era stato aggiunto per evitare di passare per associazioni ed entità che avevano un grande volume atteso, tuttavia in realtà questo passaggio non è necessario, poiché è sufficiente inserire un'associazione, che nello schema ristrutturato verrà chiamata "Svolgimento" tra le entità "Personale medico" (1,1) e "Esame" (0, N).

L'eliminazione della generalizzazione tra "Primario" e "Personale medico" è stata trattata accorpando l'entità figlia nell'entità genitore, basandosi su due considerazioni:

1. L'associazione "Afferenza" in realtà non era parte delle specifiche, non è vero che i membri del personale medico sia legati in qualche modo ad uno specifico laboratorio;
2. I dati relativi ai Primari sarebbero altrimenti ripetuti senza aggiungere altro tra quelli del Personale medico.

Per quanto concerne gli identificatori primari, sono stati scelti i seguenti:

- Paziente: Codice tessera sanitaria;
- Recapito: Questa entità aveva dei problemi nella decisione degli identificatori per come è stata progettata, perciò è stata separata in entità separate "Email", "Telefono" e "Cellulare", basandosi sulla considerazione che non tutte e tre debbano necessariamente esserci ma che se ci sono devono essere uniche per il paziente.
- Prenotazione Esame: Codice Prenotazione, Codice Esame, Data e ora; Questa entità ha altri identificatori utilizzabili, per esempio si sarebbe potuto utilizzare il laboratorio (codice di laboratorio e ospedale) e data e ora. La scelta di questo identificatore primario dipende dal fatto che ci sono operazioni basate sulla ricerca tramite codice di prenotazione mentre non ce ne sono riguardo la data e l'ora. Inizialmente era stato scelto l'altro identificatore proposto con l'idea di creare un indice sul codice di prenotazione, infine però questa idea è sembrata

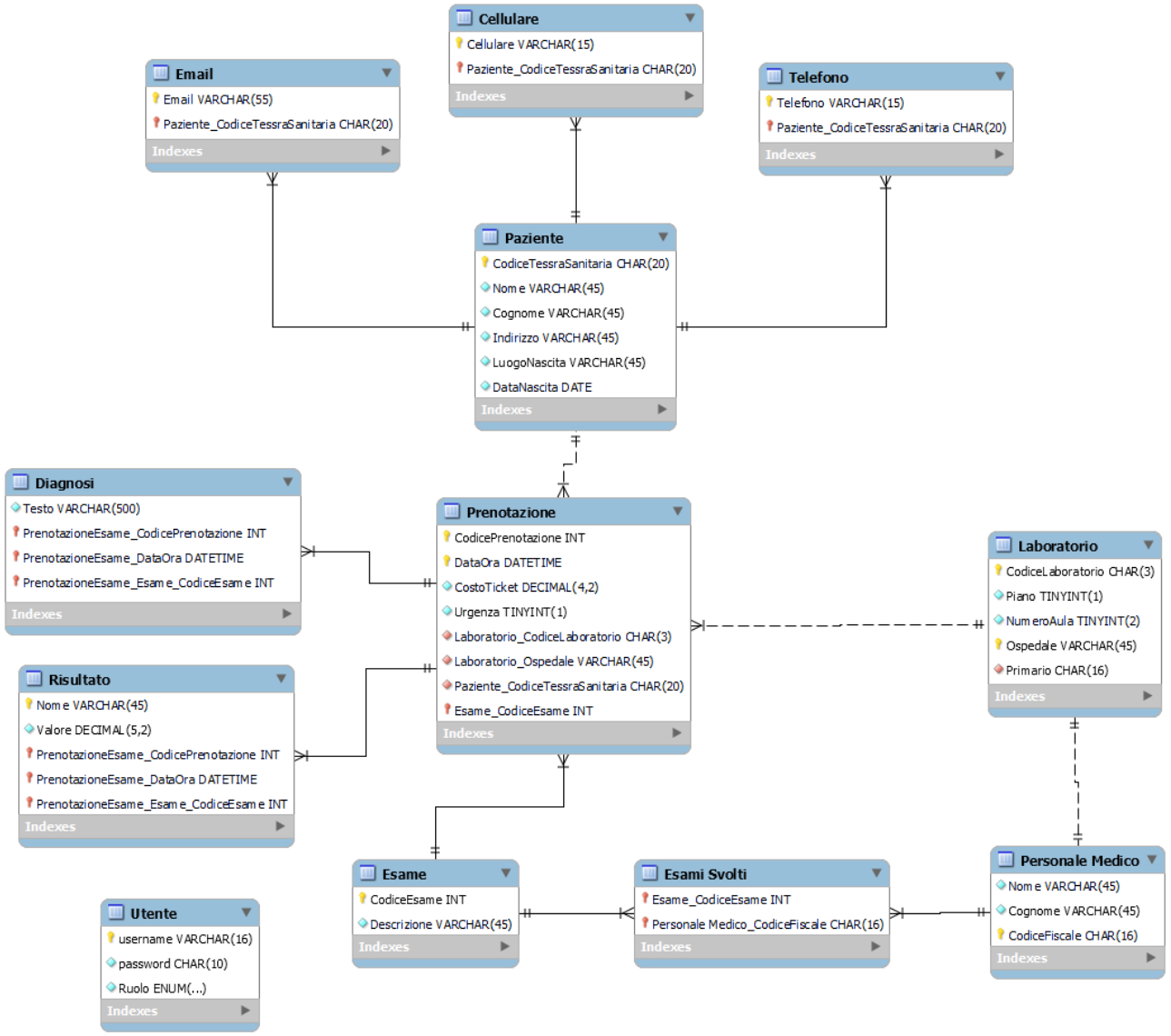
superflua considerando l'indice di chiave primaria che viene generato in ogni caso e la non necessità di averne uno sugli attributi data ora e laboratorio.

- Laboratorio: Codice di laboratorio, Ospedale;
- Personale Medico: Codice Fiscale;
- Esame: Codice Esame;
- Diagnosi: Codice Esame, Data e ora, Codice Prenotazione;
- Risultato: Nome, Codice Esame, Data e ora, Codice Prenotazione;

Trasformazione di attributi e identificatori

In fase di decisione degli identificatori primari e di trasformazione nel modello relazionale c'è stato un inghippo nella decisione degli identificatori per le entità "Diagnosi" e "Risultato": queste erano state associate all'entità "Esame" in fase di progettazione concettuale, decidendo come identificatore il codice d'esame cui erano associate, tuttavia, ovviamente, la cosa non poteva bastare dal momento che il codice dell'esame non dice nulla su chi è stato sottoposto a quell'esame o quando, perciò si è deciso di aggiungere come identificatori delle entità in questione la data e ora e il codice di prenotazione.

Traduzione di entità e associazioni



Normalizzazione del modello relazionale

Date le dipendenze funzionali:

CodicePrenotazione->CodiceTesseraSanitaria

CodiceEsame->Descrizione

Il modello fornito non è in forma normale di Boyce e Codd ma è in terza forma normale:

la seconda dipendenza ha a primo membro una superchiave minimale, e a secondo un attributo non chiave, perciò rispetta le condizioni per la forma normale di Boyce e Codd;

La prima dipendenza invece non ha una superchiave a primo membro perciò non è in forma normale di Boyce e Codd, tuttavia CodiceTesseraSanitaria fa parte di una possibile chiave per la relazione Prenotazione Esame(CodiceTesseraSanitaria, DataOra, CodiceEsame), ossia è in terza forma

normale. La scelta di non decomporre la relazione comporta che il valore del codice della tessera sanitaria del paziente sia ripetuto per ogni esame prenotato con lo stesso codice di prenotazione, tuttavia la scelta di accorpare l'attributo CodiceProntazione nell'entità deriva dal un miglioramento notevole negli accessi in un'operazione, perciò la perdita di memoria è accettabile.

5. Progettazione fisica

Utenti e privilegi

Si prevedono 4 ruoli:

- Login:
  - Grant in esecuzione sull'operazione L1;
- CUP:
  - Grant in esecuzione sulle operazioni CUP1, CUP2, CUP3, CUP4, CUP5;
- Amministratore:
  - Grant in esecuzione sulle operazioni AM1, AM2;
- Medico:
  - Grant in esecuzione sull'operazione PM1.

Strutture di memorizzazione

Tabella Paziente		
Colonna	**Tipo di dato**	**Attributi**
CodiceTesseraSanitaria	Char(20)	PK, NN
Nome	Varchar(45)	NN
Cognome	Varchar(45)	NN
Indirizzo	Varchar(55)	NN
DataDiNascita	DATE	NN
LuogoDiNascita	Varchar(45)	NN

Tabella Email		
Colonna	**Tipo di dato**	**Attributi**
CodiceTesseraSanitaria	Char(20)	PK, NN
E-mail	Varchar(45)	PK, NN

Tabella Telefono		
Colonna	**Tipo di dato**	**Attributi**
CodiceTesseraSanitaria	Char(20)	PK, NN
Telefono	Varchar(15)	PK, NN

Tabella Cellulare		
Colonna	**Tipo di dato**	**Attributi**
CodiceTesseraSanitaria	Char(20)	PK, NN
Cellulare	Varchar(15)	PK, NN

Tabella Prenotazione		
Colonna	**Tipo di dato**	**Attributi**
CodicePrenotazione	INT	PK, NN
DataOra	DATETIME	PK, NN
CostoTicket	Decimal(4, 2)	NN
Urgenza	TinyInt(1)	NN
CodiceLaboratorio	Char(3)	NN
Ospedale	Varchar(45)	NN
CodiceTesseraSanitaria	Char(20)	NN
CodiceEsame	INT	PK, NN

Tabella Laboratorio		
Colonna	**Tipo di dato**	**Attributi**
CodiceLaboratorio	Char(3)	PK, NN, G*
Piano	TinyInt(1)	NN
NumeroAula	TinyInt(2)	NN
Ospedale	Varchar(45)	PK, NN
Primario	Char(16)	NN

*“CodiceLaboratorio” è generato concatenando “Piano” e “NumeroAula”.

Tabella Personale Medico		
Colonna	**Tipo di dato**	**Attributi**
CodiceFiscale	Char(16)	PK, NN
Nome	Varchar(45)	NN
Cognome	Varchar(45)	NN

Tabella Esame		
Colonna	**Tipo di dato**	**Attributi**
CodiceEsame	INT	PK, NN, AI
Descrizione	Varchar(45)	NN, UQ

Tabella Esami Svolti		
Colonna	**Tipo di dato**	**Attributi**
CodiceEsame	INT	PK, NN
CodiceFiscale	Char(16)	PK, NN

Tabella Risultato		
Colonna	**Tipo di dato**	**Attributi**
CodiceEsame	INT	PK, NN
Nome	Varchar(45)	PK, NN
Valore	DECIMAL(5,2)	NN
CodicePrenotazione	INT	PK, NN
DataOra	DATETIME	PK, NN

Tabella Diagnosi		
Colonna	**Tipo di dato**	**Attributi**
CodiceEsame	INT	PK, NN
Testo	Varchar(500)	NN
CodicePrenotazione	INT	PK, NN
DataOra	DATETIME	PK, NN

Tabella Utente		
Colonna	**Tipo di dato**	**Attributi**
Username	VARCHAR(16)	PK, NN
Password	CHAR(10	NN
Ruolo	ENUM(CUP, AM, PM)	NN

Indici

L'indice "Paziente" è stato pensato per ordinare le prenotazioni per il campo CodiceTesseraSanitaria per favorire l'operazione "Storico Esami", che riporta gli esami svolti da un paziente dalla sua registrazione.

Tabella Prenotazione	
Indice Paziente	**Tipo:**
CodiceTesseraSanitaria	IDX

Trigger

Tabella Prenotazione: BEFORE INSERT

Il seguente trigger implementa un controllo per garantire che un paziente non prenoti (di conseguenza non esegua) lo stesso esame lo stesso giorno:

```
CREATE TRIGGER verifica_prenotazione
BEFORE INSERT ON Prenotazione
FOR EACH ROW
BEGIN
    IF EXISTS (
    SELECT 1
    FROM Prenotazione
    WHERE Paziente_CodiceTesseraSanitaria = NEW.Paziente_CodiceTesseraSanitaria AND Esame_CodiceEsame = NEW.Esame_CodiceEsame AND DATE(DataOra) = DATE(NEW.DataOra)
    ) THEN
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Un paziente non può prenotare lo stesso esame più volte nello stesso giorno';
    END IF;
END
```

Tabella Prenotazione: BEFORE INSERT

Il seguente trigger implementa un controllo per garantire che la data e l'ora dell'esame prenotato siano successive al momento in cui essa viene registrata nel sistema:

```
CREATE TRIGGER verifica_data_prenotazione
BEFORE INSERT ON Prenotazione
```

```
FOR EACH ROW
BEGIN
    IF NEW.DataOra <= NOW() THEN
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'La data e l''ora della prenotazione devono essere successive a quella in cui la prenotazione viene registrata';
    END IF;
END
```

Tabella Prenotazione: BEFORE INSERT

Il seguente trigger implementa un controllo per garantire che un paziente non prenoti due esami alla stessa ora:

```
CREATE TRIGGER doppia_prenotazione
BEFORE INSERT ON Prenotazione
FOR EACH ROW
BEGIN
    DECLARE prenotazioni INT;
    SELECT COUNT(*) INTO prenotazioni
    FROM Prenotazione
    WHERE DataOra = NEW.DataOra AND Paziente_CodiceTesseraSanitaria = NEW.Paziente_CodiceTesseraSanitaria;
    IF prenotazioni > 0 THEN
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Un paziente non può prenotare più esami alla stessa ora';
    END IF;
END //
```

Tabella Prenotazione: BEFORE INSERT

Il seguente trigger implementa un controllo per garantire che ogni codice di prenotazione sia associato ad un paziente:

```
CREATE TRIGGER verifica_paziente_prenotazione
BEFORE INSERT ON Prenotazione
FOR EACH ROW
BEGIN
    DECLARE prenotazioni INT;
    SELECT COUNT(*) INTO prenotazioni
    FROM Prenotazione
    WHERE CodicePrenotazione = NEW.CodicePrenotazione AND Paziente_CodiceTesseraSanitaria != NEW.Paziente_CodiceTesseraSanitaria;
    IF prenotazioni > 0 THEN
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Un codice di prenotazione può essere associato ad un solo paziente';
    END IF;
END //
```

Eventi

Non sono stati utilizzati eventi.

Viste

Non sono state utilizzate viste.

Stored Procedures e transazioni

Operazione CUP1:

```
CREATE PROCEDURE registra_paziente (in var_codiceTesseraSanitaria CHAR(20), in var_nome VARCHAR(45), in var_cognome VARCHAR(45), in var_dataNascita DATETIME, in var_luogoNascita VARCHAR(45), in var_indirizzo VARCHAR(45))
  BEGIN
      INSERT INTO Paziente (CodiceTesseraSanitaria, Nome, Cognome, DataNascita, LuogoNascita, Indirizzo)
      VALUES (var_codiceTesseraSanitaria, var_nome, var_cognome, var_dataNascita, var_luogoNascita, var_indirizzo);
  END
```

```
CREATE PROCEDURE registra_email (in var_email VARCHAR(45), in var_codiceTesseraSanitaria CHAR(20))
  BEGIN
      INSERT INTO Email (Email, Paziente_CodiceTesseraSanitaria)
      VALUES (var_email, var_codiceTesseraSanitaria);
  END
```

```
CREATE PROCEDURE registra_telefono (in var_telefono VARCHAR(15), in var_codiceTesseraSanitaria CHAR(20))
  BEGIN
      INSERT INTO Email (Telefono, Paziente_CodiceTesseraSanitaria)
      VALUES (var_telefono, var_codiceTesseraSanitaria);
  END
```

```
CREATE PROCEDURE registra_cellulare (in var_cellulare VARCHAR(15), in var_codiceTesseraSanitaria CHAR(20))
  BEGIN
      INSERT INTO Email (Cellulare, Paziente_CodiceTesseraSanitaria)
```

```
        VALUES (var_cellulare, var_codiceTesseraSanitaria);
    END
```

Operazione CUP2:

```
CREATE PROCEDURE lista_esami ()
    BEGIN
        SELECT *
        FROM Esame;
    END
```

Operazione CUP3:

```
CREATE PROCEDURE registra_prenotazione(in var_codicePrenotazione INT, in var_dataOra DATETIME, in var_codiceEsame INT, in var_costo DECIMAL(4,2),
in var_urgenza BOOLEAN, in var_paziente CHAR(20))
    BEGIN
            declare exit handler for sqlexception
        begin
            rollback;
            resignal;
        end;

        set transaction isolation level serializable;
        start transaction;
            INSERT INTO Prenotazione (CodicePrenotazione, DataOra, Esame_CodiceEsame, Costo, Urgenza, Laboratorio_CodiceLaboratorio, Laboratorio_Ospedale, Paziente_CodiceTesseraSanitaria)
            VALUES (var_codicePrenotazione, var_dataOra, var_codiceEsame, var_costo, var_urgenza, codice_laboratorio_disponibile, ospedale_laboratorio_disponibile, var_paziente);
        commit;
    END
```

In questa procedura la sezione di inserimento viene gestita con una transazione cui livello di isolamento viene impostato su “serializable” poiché all’interno della transazione si fa uso di “codice_laboratorio_disponibile” e “ospedale_laboratorio_disponibile”, che sono due funzioni che servono per scegliere un laboratorio tra quelli che non sono impegnati nella data e ora selezionate, poiché potrebbero esserci altre prenotazioni concorrenti che cercano di ottenere lo stesso laboratorio, un livello di isolamento read committed o repeatable read non è sufficiente

Operazione CUP4:

```
CREATE PROCEDURE report_prenotazioni(in var_codicePrenotazione INT)
  BEGIN
        declare exit handler for sqlexception
        begin
                rollback;
                resignal;
        end;
        set transaction isolation level read committed;
       set transaction read only;
        start transaction;
                SELECT Nome, Valore
                        FROM Risultato
                        WHERE Prenotazione_CodicePrenotazione = var_codicePrenotazione;
        commit;
  END
```

Operazione CUP5:

```
CREATE PROCEDURE storico_esami(in var_tesseraSanitaria CHAR(20))
BEGIN
       declare exit handler for sqlexception
        begin
                rollback;
                resignal;
        end;
```

```
 set transaction isolation level read committed;
set transaction read only;
start transaction;
        SELECT P.DataOra, P.Esame_CodiceEsame, E.Descrizione, P.Costo, P.Urgenza
            FROM Prenotazione AS P
            JOIN Esame AS E ON P.Esame_CodiceEsame = E.CodiceEsame
        WHERE P.Paziente_CodiceTesseraSanitaria = tesseraSanitaria AND EXISTS (
            SELECT 1
            FROM Risultato AS R
             WHERE R.Prenotazione_CodicePrenotazione = P.CodicePrenotazione AND
R.Prenotazione_DataOra = P.DataOra AND R.Prenotazione_Esame_CodiceEsame =
P.Esame_CodiceEsame
        );
 commit;
END
```

In questo caso oltre ad aver impostato il livello di isolamento su “read committed”, utilizzato per assicurarsi di non effettuare letture sporche, setta la transazione come read only affinché i dati che vengono riportati nello storico non vengano modificati durante la transazione.

Operazione AM1:

```
CREATE PROCEDURE registra_esame(in var_descrizione VARCHAR(45))
  BEGIN
      INSERT INTO Esame(Descrizione)
      VALUES (var_descrizione);
  END
```

Operazione AM2:

```
CREATE PROCEDURE report_personale_annuale()
  BEGIN
      SELECT
          Personale_CodiceFiscale,
          YEAR(DataOra) AS Anno,
          COUNT(*) AS NumeroEsami
      FROM Prenotazione
              JOIN EsamiSvolti ON Prenotazione.Esame_CodiceEsame =
EsamiSvolti.Esame_CodiceEsame
      GROUP BY Personale_CodiceFiscale, YEAR(DataOra)
      ORDER BY Personale_CodiceFiscale, Anno;
END


CREATE PROCEDURE report_personale_mensile()
  BEGIN
      SELECT
          Personale_CodiceFiscale,
          YEAR(DataOra) AS Anno,
          MONTH(DataOra) AS Mese,
          COUNT(*) AS NumeroEsami
      FROM Prenotazione
              JOIN EsamiSvolti ON Prenotazione.Esame_CodiceEsame =
EsamiSvolti.Esame_CodiceEsame
      GROUP BY Personale_CodiceFiscale, YEAR(DataOra), MONTH(DataOra)
      ORDER BY Personale_CodiceFiscale, Anno, Mese;
  END
```

Operazione PM:

```
CREATE PROCEDURE registra_risultato(in var_nome VARCHAR(45), in var_dataOra DATETIME, in var_codicePrenotazione INT, in var_codiceEsame INT, in var_valore DECIMAL(5,2))
  BEGIN
        declare exit handler for sqlexception
       begin
            rollback;
            resignal;
       end;
       start transaction;
      INSERT INTO Risultato (Nome, Prenotazione_DataOra, Prenotazione_CodicePrenotazione, Prenotazione_Esame_CodiceEsame, Valore)
       VALUES (var_nome, var_dataOra, var_codicePrenotazione, var_codiceEsame, var_valore);
      commit;
  END

CREATE PROCEDURE registra_diagnosi(in var_testo VARCHAR(500), in var_dataOra DATETIME, in var_codicePrenotazione INT, in var_codiceEsame INT)
  BEGIN
      INSERT INTO Diagnosi (Testo, Prenotazione_DataOra, Prenotazione_CodicePrenotazione, Prenotazione_Esame_CodiceEsame)
       VALUES (var_testo, var_dataOra, var_codicePrenotazione, var_codiceEsame);
  END



CREATE PROCEDURE esami_svolti(in var_codiceEsame INT, in var_codiceFiscale CHAR(16))
  BEGIN
      declare exit handler for sqlexception
       begin
            rollback;
            resignal;
```

```
        end;
        start transaction;
        INSERT INTO EsamiSvolti(Esame_CodiceEsame, Personale_CodiceFiscale)
       VALUES(var_codiceEsame, var_codiceFiscale);
       commit;
  END
```

Operazione L1:

```
CREATE PROCEDURE login (in var_username varchar(16), in var_pass CHAR(10), out var_role INT)
  BEGIN
        declare var_user_role ENUM('CUP', 'AM', 'PM');
        SELECT Ruolo from Utente
                where Username = var_username
                and Passwd = md5(var_pass)
                into var_user_role;
        if var_user_role = ‘AM’ then
                set var_role = 1;
        elseif var_user_role = ‘CUP’ then
                set var_role = 2;
        elseif var_user_role = 'PM' then
                set var_role = 3;
        else
                set var_role = 4;
        end if;
  END
```

Appendice: Implementazione

Codice SQL per istanziare il database

- Codice SQL per la creazione dello schema e delle tabelle:

```
DROP SCHEMA IF EXISTS asl;
CREATE SCHEMA asl;
USE asl;

CREATE TABLE Paziente (
        CodiceTesseraSanitaria CHAR(20) PRIMARY KEY,
        Nome VARCHAR(45) NOT NULL,
        Cognome VARCHAR(45) NOT NULL,
        DataNascita DATE NOT NULL,
        LuogoNascita VARCHAR(45) NOT NULL,
        Indirizzo VARCHAR(45) NOT NULL
);

CREATE TABLE Cellulare (
        Cellulare VARCHAR(15) NOT NULL,
        Paziente_CodiceTesseraSanitaria CHAR(20) NOT NULL,
        PRIMARY KEY (Cellulare, Paziente_CodiceTesseraSanitaria),
        FOREIGN KEY (Paziente_CodiceTesseraSanitaria) REFERENCES
Paziente(CodiceTesseraSanitaria)
);

CREATE TABLE Telefono (
        Telefono VARCHAR(15) NOT NULL,
        Paziente_CodiceTesseraSanitaria CHAR(20) NOT NULL,
        PRIMARY KEY (Telefono, Paziente_CodiceTesseraSanitaria),
        FOREIGN KEY (Paziente_CodiceTesseraSanitaria) REFERENCES
Paziente(CodiceTesseraSanitaria)
```

```
);

CREATE TABLE Email (
	Email VARCHAR(45) NOT NULL,
	Paziente_CodiceTesseraSanitaria CHAR(20) NOT NULL,
	PRIMARY KEY (Email, Paziente_CodiceTesseraSanitaria),
	FOREIGN KEY (Paziente_CodiceTesseraSanitaria) REFERENCES Paziente(CodiceTesseraSanitaria)
);

CREATE TABLE Esame (
	CodiceEsame INT AUTO_INCREMENT PRIMARY KEY,
	Descrizione VARCHAR(45) NOT NULL UNIQUE
);

CREATE TABLE Personale (
	CodiceFiscale CHAR(16) PRIMARY KEY,
	Nome VARCHAR(45) NOT NULL,
	Cognome VARCHAR(45) NOT NULL
);

CREATE TABLE EsamiSvolti (
	Esame_CodiceEsame INT NOT NULL,
	Personale_CodiceFiscale CHAR(16) NOT NULL,
	FOREIGN KEY (Esame_CodiceEsame) REFERENCES Esame(CodiceEsame),
	FOREIGN KEY (Personale_CodiceFiscale) REFERENCES Personale(CodiceFiscale)
);

CREATE TABLE Laboratorio (
	CodiceLaboratorio INT GENERATED ALWAYS AS (CONCAT(Piano, NumeroAula)) STORED NOT NULL,
	Ospedale VARCHAR(45) NOT NULL,
	NumeroAula INT NOT NULL,
	Piano INT NOT NULL,
```

```
    Primario CHAR(16) NOT NULL,
    PRIMARY KEY (CodiceLaboratorio, Ospedale),
    FOREIGN KEY (Primario) REFERENCES Personale(CodiceFiscale)
);

CREATE TABLE Prenotazione (
    CodicePrenotazione INT,
    DataOra DATETIME NOT NULL,
    Esame_CodiceEsame INT NOT NULL,
    Costo DECIMAL(4,2) NOT NULL,
    Urgenza BOOLEAN NOT NULL,
    Laboratorio_CodiceLaboratorio INT NOT NULL,
    Laboratorio_Ospedale VARCHAR(45) NOT NULL,
    Paziente_CodiceTesseraSanitaria CHAR(20) NOT NULL,
    PRIMARY KEY(CodicePrenotazione, DataOra, Esame_CodiceEsame),
    FOREIGN KEY (Esame_CodiceEsame) REFERENCES Esame(CodiceEsame),
    FOREIGN KEY (Laboratorio_CodiceLaboratorio, Laboratorio_Ospedale)
REFERENCES Laboratorio(CodiceLaboratorio, Ospedale),
    FOREIGN KEY (Paziente_CodiceTesseraSanitaria) REFERENCES
Paziente(CodiceTesseraSanitaria)
);

CREATE TABLE Risultato (
    Nome VARCHAR(45) NOT NULL,
    Prenotazione_DataOra DATETIME NOT NULL,
    Prenotazione_CodicePrenotazione INT NOT NULL,
    Prenotazione_Esame_CodiceEsame INT NOT NULL,
    Valore DECIMAL(5,2) NOT NULL,
    PRIMARY KEY (Nome, Prenotazione_DataOra, Prenotazione_CodicePrenotazione,
Prenotazione_Esame_CodiceEsame),
    FOREIGN KEY (Prenotazione_CodicePrenotazione, Prenotazione_DataOra,
Prenotazione_Esame_CodiceEsame) REFERENCES Prenotazione(CodicePrenotazione,
DataOra, Esame_CodiceEsame)
);
```

```
CREATE TABLE Diagnosi (
  Prenotazione_DataOra DATETIME NOT NULL,
  Prenotazione_CodicePrenotazione INT NOT NULL,
  Prenotazione_Esame_CodiceEsame INT NOT NULL,
  Testo VARCHAR(500) NOT NULL,
        PRIMARY KEY (Prenotazione_DataOra, Prenotazione_CodicePrenotazione, Prenotazione_Esame_CodiceEsame),
        FOREIGN KEY (Prenotazione_CodicePrenotazione, Prenotazione_DataOra, Prenotazione_Esame_CodiceEsame) REFERENCES Prenotazione(CodicePrenotazione, DataOra, Esame_CodiceEsame)
);

CREATE TABLE Utente (
        Username VARCHAR(16) PRIMARY KEY,
        Passwd CHAR(10) NOT NULL,
        Ruolo ENUM('CUP', 'AM', 'PM') NOT NULL
);
```

- Codice SQL per indici e funzioni:

```
CREATE FUNCTION codice_laboratorio_disponibile(var_dataOra DATETIME) RETURNS CHAR(3) DETERMINISTIC
  RETURN
    (SELECT CodiceLaboratorio
     FROM Laboratorio AS L
     WHERE NOT EXISTS (
        SELECT 1
        FROM Prenotazione AS P
          WHERE P.Laboratorio_CodiceLaboratorio = L.CodiceLaboratorio AND P.Laboratorio_Ospedale = L.Ospedale AND P.DataOra = var_dataOra
     )
```

```
);

CREATE FUNCTION ospedale_laboratorio_disponibile(var_dataOra DATETIME) RETURNS VARCHAR(45) DETERMINISTIC
  RETURN
      (SELECT Ospedale
       FROM Laboratorio AS L
       WHERE NOT EXISTS (
            SELECT 1
            FROM Prenotazione AS P
              WHERE P.Laboratorio_CodiceLaboratorio = L.CodiceLaboratorio AND P.Laboratorio_Ospedale = L.Ospedale AND P.DataOra = var_dataOra
       )
  );

CREATE INDEX idx_paziente_tessera
ON Prenotazione (Paziente_CodiceTesseraSanitaria);
```

- Codice SQL per popolare la base di dati:

```
USE asl;

INSERT INTO Personale (CodiceFiscale, Nome, Cognome)
VALUES
('CF00000000000001', 'Mario', 'Rossi'),
('CF00000000000002', 'Giuseppe', 'Verdi'),
('CF00000000000003', 'Luigi', 'Bianchi'),
('CF00000000000004', 'Francesco', 'Romano'),
('CF00000000000005', 'Antonio', 'Ricci'),
('CF00000000000006', 'Giovanni', 'Ferrari'),
('CF00000000000007', 'Angelo', 'Esposito'),
('CF00000000000008', 'Roberto', 'Conti'),
```

('CF00000000000009', 'Paolo', 'Vitale'),
('CF00000000000010', 'Marco', 'Rizzo'),
('CF00000000000011', 'Paolo', 'Conti'),
('CF00000000000012', 'Mario', 'Rossi'),
('CF00000000000013', 'Giuseppe', 'Verdi'),
('CF00000000000014', 'Luigi', 'Bianchi'),
('CF00000000000015', 'Francesco', 'Romano'),
('CF00000000000016', 'Antonio', 'Ricci'),
('CF00000000000017', 'Giovanni', 'Ferrari'),
('CF00000000000018', 'Angelo', 'Esposito'),
('CF00000000000019', 'Roberto', 'Conti'),
('CF00000000000020', 'Paolo', 'Vitale'),
('CF00000000000021', 'Marco', 'Rizzo'),
('CF00000000000022', 'Paolo', 'Conti'),
('CF00000000000023', 'Mario', 'Rossi'),
('CF00000000000024', 'Giuseppe', 'Verdi'),
('CF00000000000025', 'Luigi', 'Bianchi'),
('CF00000000000026', 'Francesco', 'Romano'),
('CF00000000000027', 'Antonio', 'Ricci'),
('CF00000000000028', 'Giovanni', 'Ferrari'),
('CF00000000000029', 'Angelo', 'Esposito'),
('CF123456789', 'Giuseppe', 'Verdi'),
('CF00000000000030', 'Roberto', 'Conti');

INSERT INTO Laboratorio (Ospedale, NumeroAula, Piano, Primario)
VALUES
('Ospedale 1', 01, 1, 'CF00000000000001'),
('Ospedale 1', 02, 1, 'CF00000000000002'),
('Ospedale 1', 03, 1, 'CF00000000000003'),
('Ospedale 1', 04, 2, 'CF00000000000004'),
('Ospedale 1', 05, 2, 'CF00000000000005'),
('Ospedale 1', 06, 2, 'CF00000000000006'),
('Ospedale 1', 07, 3, 'CF00000000000007'),
('Ospedale 1', 08, 3, 'CF00000000000008'),

('Ospedale 1', 09, 3, 'CF00000000000009'),
('Ospedale 1', 10, 4, 'CF00000000000010'),
('Ospedale 2', 01, 1, 'CF00000000000011'),
('Ospedale 2', 02, 1, 'CF00000000000012'),
('Ospedale 2', 03, 1, 'CF00000000000013'),
('Ospedale 2', 04, 2, 'CF00000000000014'),
('Ospedale 2', 05, 2, 'CF00000000000015'),
('Ospedale 2', 06, 2, 'CF00000000000016'),
('Ospedale 2', 07, 3, 'CF00000000000017'),
('Ospedale 2', 08, 3, 'CF00000000000018'),
('Ospedale 2', 09, 3, 'CF00000000000019'),
('Ospedale 2' ,10 ,4 ,'CF00000000000020');

INSERT INTO Paziente (CodiceTesseraSanitaria, Nome, Cognome, DataNascita, LuogoNascita, Indirizzo)
VALUES
('TS000000000000000001', 'Mario', 'Rossi', '1990-01-01', 'Roma', 'Via Roma 1'),
('TS000000000000000002', 'Giuseppe', 'Verdi', '1991-02-02', 'Milano', 'Via Milano 2'),
('TS000000000000000003', 'Luigi', 'Bianchi', '1992-03-03', 'Napoli', 'Via Napoli 3'),
('TS000000000000000004', 'Francesco', 'Romano', '1993-04-04', 'Torino', 'Via Torino 4'),
('TS000000000000000005', 'Antonio', 'Ricci', '1994-05-05', 'Palermo', 'Via Palermo 5'),
('TS000000000000000006', 'Giovanni', 'Ferrari', '1995-06-06', 'Genova', 'Via Genova 6'),
('TS000000000000000007', 'Angelo', 'Esposito', '1996-07-07', 'Bologna', 'Via Bologna 7'),
('TS000000000000000008', 'Mario', 'Rossi', '1990-01-01', 'Roma', 'Via Roma 1'),
('TS000000000000000009', 'Giuseppe', 'Verdi', '1991-02-02', 'Milano', 'Via Milano 2'),
('TS000000000000000010', 'Luigi', 'Bianchi', '1992-03-03', 'Napoli', 'Via Napoli 3'),
('TS000000000000000011', 'Francesco', 'Romano', '1993-04-04', 'Torino', 'Via Torino 4'),
('TS000000000000000013', 'Antonio', 'Ricci', '1994-05-05', 'Palermo', 'Via Palermo 5'),
('TS000000000000000014', 'Giovanni', 'Ferrari', '1995-06-06', 'Genova', 'Via Genova 6'),
('TS000000000000000015', 'Angelo', 'Esposito', '1996-07-07', 'Bologna', 'Via Bologna 7');

INSERT INTO Cellulare (Paziente_CodiceTesseraSanitaria, Cellulare)
VALUES
('TS000000000000000001', '1234567890'),

('TS000000000000000002', '1234567890'),
('TS000000000000000003', '3456789012'),
('TS000000000000000004', '4567890123'),
('TS000000000000000005', '5678901234'),
('TS000000000000000006', '6789012345'),
('TS000000000000000007', '7890123456'),
('TS000000000000000008', '8901234567'),
('TS000000000000000009', '9012345678'),
('TS000000000000000010', '0123456789'),
('TS000000000000000011', '1357924680'),
('TS000000000000000013', '2468013579'),
('TS000000000000000014', '3579246801'),
('TS000000000000000015', '4680135792');

INSERT INTO Esame (Descrizione)
VALUES
('Esame del sangue'),
('Radiografia'),
('Ecografia'),
('Risonanza magnetica'),
('Tomografia computerizzata'),
('Elettrocardiogramma'),
('Esame delle urine'),
('Biopsia'),
('Endoscopia'),
('Mammografia'),
('Esame del liquido cefalorachidiano'),
('Esame del midollo osseo'),
('Esame del tessuto muscolare'),
('Esame del tessuto nervoso'),
('Esame del tessuto cutaneo'),
('Esame del tessuto adiposo'),
('Esame del tessuto connettivo'),

```
('Esame del tessuto osseo'),
('Esame del tessuto cartilagineo'),
('Esame del tessuto epiteliale');


INSERT INTO Utente (Username, Passwd, Ruolo)
VALUES
('utente1', 'password1', 'CUP'),
('utente2', 'password2', 'AM'),
('utente3', 'password3', 'PM'),
('utente4', 'password4', 'CUP'),
('utente5', 'password5', 'AM'),
('utente6', 'password6', 'PM'),
('utente7', 'password7', 'CUP'),
('utente8', 'password8', 'AM'),
('utente9', 'password9', 'PM'),
('utente10','passwrd10','CUP'),
('utente11','passwrd11','AM'),
('utente12','passwrd12','PM'),
('utente13','passwrd13','CUP'),
('utente14','passwrd14','AM'),
('utente15','passwrd15','PM'),
('utente16','passwrd16','CUP'),
('utente17','passwrd17','AM'),
('utente18','passwrd18','PM'),
('utente19','passwrd19','CUP'),
('utente20','passwrd20','AM'),
('utente21','passwrd21','PM'),
('utente22','passwrd22','CUP'),
('utente23','passwrd23','AM'),
('utente24','passwrd24','PM'),
('utente25','passwrd25','CUP');
```

```
INSERT INTO Prenotazione ( DataOra, Esame_CodiceEsame, Costo, Urgenza, Laboratorio_CodiceLaboratorio, Laboratorio_Ospedale, Paziente_CodiceTesseraSanitaria)
VALUES ('2024-11-30 09:00:00', 1, 50.00, false, 11, 'Ospedale 1', 'TS00000000000000001');

INSERT INTO EsamiSvolti (Esame_CodiceEsame, Personale_CodiceFiscale)
VALUES (1, 'CF123456789');

INSERT INTO Risultato (Nome, Prenotazione_DataOra, Prenotazione_CodicePrenotazione, Prenotazione_Esame_CodiceEsame, Valore)
VALUES ('Emoglobina', '2024-11-30 09:00:00', 1, 1, 15.5);

INSERT INTO Diagnosi (Prenotazione_DataOra, Prenotazione_CodicePrenotazione, Prenotazione_Esame_CodiceEsame, Testo)
VALUES ('2024-11-30 09:00:00', 1, 1, 'Tutto nella norma');
```

- Ruoli e grant:

```
DROP USER IF EXISTS login, CUP, amministratore, medico;
CREATE USER login IDENTIFIED BY 'login';
CREATE USER CUP IDENTIFIED BY 'personaleCUP';
CREATE USER amministratore IDENTIFIED BY 'amministratore';
CREATE USER medico IDENTIFIED BY 'medico';

GRANT ALL PRIVILEGES ON asl TO CUP, amministratore, medico, login;

GRANT EXECUTE ON PROCEDURE login TO login;
GRANT EXECUTE ON PROCEDURE registra_paziente TO CUP;
GRANT EXECUTE ON PROCEDURE registra_email TO CUP;
GRANT EXECUTE ON PROCEDURE registra_telefono TO CUP;
GRANT EXECUTE ON PROCEDURE registra_cellulare TO CUP;
GRANT EXECUTE ON PROCEDURE lista_esami TO CUP;
GRANT EXECUTE ON PROCEDURE registra_prenotazione TO CUP;
```

```
GRANT EXECUTE ON PROCEDURE report_prenotazioni TO CUP;
GRANT EXECUTE ON PROCEDURE storico_esami TO CUP;
GRANT EXECUTE ON PROCEDURE registra_esame TO amministratore;
GRANT EXECUTE ON PROCEDURE report_personale_annuale TO amministratore;
GRANT EXECUTE ON PROCEDURE report_personale_mensile TO amministratore;
GRANT EXECUTE ON PROCEDURE registra_risultato TO medico;
GRANT EXECUTE ON PROCEDURE registra_diagnosi TO medico;
GRANT EXECUTE ON PROCEDURE esami_svolti TO medico;

FLUSH PRIVILEGES;
```

Codice del Front-End

- **CONTOLLER**

```
package controller;

import model.dao.ConnectionFactory;
import model.dao.ExamDAO;
import model.dao.StaffDao;
import model.domain.Role;
import view.AmministratoreView;

import java.io.IOException;
import java.sql.SQLException;

public class AmministratoreController implements Controller {

	@Override
	public void start() {
	try {
	ConnectionFactory.changeRole(Role.AMMINISTRATORE);
	} catch(SQLException e) {
	throw new RuntimeException(e);
	}

	while(true) {
	int choice;
	try {
		choice = AmministratoreView.showMenu();
	} catch(IOException e) {
		throw new RuntimeException(e);
	}

	switch(choice) {
		case 1 -> addExam();
```

```
                case 2 -> staffReport();
                case 3 -> System.exit(0);
                default -> throw new RuntimeException("Invalid choice");
        }
        }
        }

        private void staffReport() {
        String basis = AmministratoreView.showStaffReportChoice();
        StaffDao dao = new StaffDao();
        dao.getStaffReport(basis);
        }

        private void addExam() {
        String name = AmministratoreView.showAddExamMenu();
        ExamDAO dao = new ExamDAO();
        dao.addExam(name);
        }
}
```

```
package controller;


import model.domain.Credentials;

public class ApplicationController implements Controller {
	Credentials cred;

	@Override
	public void start() {
	LoginController loginController = new LoginController();
	loginController.start();
	cred = loginController.getCred();

	if(cred.getRole() == null) {
	throw new RuntimeException("Invalid credentials");
	}

	switch(cred.getRole()) {
	case CUP -> new CUPController().start();
	case AMMINISTRATORE -> new AmministratoreController().start();
	case MEDICO -> new MedicoController().start();
	default -> throw new RuntimeException("Invalid credentials");
	}
	}
}
```

```
package controller;

public interface Controller {

	void start();
}
```

```
package controller;

import model.dao.AddPatientDAO;
import model.dao.ConnectionFactory;
import model.dao.ExamDAO;
import model.dao.ReservationDAO;
import model.domain.Patient;
import model.domain.Reservation;
import model.domain.Role;
import view.CUPView;

import java.io.IOException;
import java.sql.SQLException;
import java.text.ParseException;

public class CUPController implements Controller{

	@Override
	public void start() {
	try {
	ConnectionFactory.changeRole(Role.CUP);
	} catch(SQLException e) {
	throw new RuntimeException(e);
	}

	while(true) {
	int choice;
	try {
		choice = CUPView.showMenu();
	} catch(IOException e) {
		throw new RuntimeException(e);
	}
```

```
switch(choice) {
        case 1 -> addPatient();
        case 2 -> addPatientInfo();
        case 3 -> listExams();
        case 4 -> addReservation();
        case 5 -> reservationsReport();
        case 6 -> examsReport();
        case 7 -> System.exit(0);
        default -> throw new RuntimeException("Invalid choice");
}
}
}

private void listExams() {
ExamDAO dao = new ExamDAO();
dao.getExams();
}

private void examsReport() {
String code = CUPView.showExamReportMenu();
ExamDAO dao = new ExamDAO();
dao.getExamsReport(code);
}

private void addReservation() {
Reservation reservation = CUPView.showAddReservationMenu();
ReservationDAO dao = new ReservationDAO();
dao.AddReservation(reservation);
}

public void addPatient() {
Patient patient;
try {
```

```
        patient = CUPView.showAddPatientMenu();
        } catch (ParseException e) {
        throw new RuntimeException(e);
        }
        AddPatientDAO dao = new AddPatientDAO();
        dao.AddPatient(patient);
        }

        public void addPatientInfo() {
        String[] information = CUPView.showAddPatientInfoMenu();
        AddPatientDAO dao = new AddPatientDAO();
        switch (information[1]) {
        case "cellphone" -> dao.AddCellphone(information);
        case "phone" -> dao.AddPhone(information);
        case "email" -> dao.AddCellEmail(information);
        }
        }
        private void reservationsReport() {
        int code = CUPView.showReservationReportMenu();
        ReservationDAO dao = new ReservationDAO();
        dao.reservationReport(code);
        }
}
```

```
package controller;


import exception.DAOException;
import model.dao.LoginProcedureDAO;
import model.domain.Credentials;
import view.LoginView;

import java.io.IOException;

public class LoginController implements Controller {
	Credentials cred = null;

	@Override
	public void start() {
	try {
	cred = LoginView.authenticate();
	} catch(IOException e) {
	throw new RuntimeException(e);
	}

	try {
	cred = new LoginProcedureDAO().execute(cred.getUsername(), cred.getPassword());
	} catch(DAOException e) {
	throw new RuntimeException(e);
	}
	}

	public Credentials getCred() {
	return cred;
	}
}
```

```
package controller;

import model.dao.ConnectionFactory;
import model.dao.ExamDAO;
import model.dao.ExamResultsDao;
import model.domain.ExamDiagnosis;
import model.domain.ExamResult;
import model.domain.Role;
import view.MedicoView;

import java.io.IOException;
import java.sql.SQLException;

public class MedicoController implements Controller {

	@Override
	public void start() {
	try {
	ConnectionFactory.changeRole(Role.MEDICO);
	} catch(SQLException e) {
	throw new RuntimeException(e);
	}

	while(true) {
	int choice;
	try {
		choice = MedicoView.showMenu();
	} catch(IOException e) {
		throw new RuntimeException(e);
	}

	switch(choice) {
		case 1 -> addExamResult();
		case 2 -> addDiagnosis();
```

```
                case 3 -> addDoneExam();
                case 4 -> System.exit(0);
                default -> throw new RuntimeException("Invalid choice");
        }
        }
        }

        private void addDoneExam() {
        String[] doneExam = MedicoView.showAddDoneExamMenu();
        ExamDAO dao = new ExamDAO();
        dao.AddDoneExam(doneExam);
        }

        private void addDiagnosis() {
        ExamDiagnosis diagnosis = MedicoView.showAddExamDiagnosisMenu();
        ExamResultsDao dao = new ExamResultsDao();
        dao.addExamDiagnosis(diagnosis);
        }

        private void addExamResult() {
        ExamResult result = MedicoView.showAddExamResultMenu();
        ExamResultsDao dao = new ExamResultsDao();
        dao.addExamResult(result);
        }
}
```

- **MODEL**
  - **DAO**

```
package model.dao;

import model.domain.Patient;
import java.sql.*;

public class AddPatientDAO {

	public void AddPatient(Patient patient){
	try {
	Connection conn = ConnectionFactory.getConnection();
	CallableStatement statement = conn.prepareCall("{call registra_paziente(?, ?, ?, ?, ?, ?)}");
	statement.setString(1, patient.getSanitaryCode());
	statement.setString(2,patient.getName());
	statement.setString(3, patient.getSurname());
	statement.setDate(4, patient.getBirthDate());
	statement.setString(5, patient.getBirthPlace());
	statement.setString(6, patient.getSurname());

	statement.execute();
	}
	catch (SQLException e){
	System.out.println("error: " + e.getMessage().toString());
	}
	}

	public void AddPhone(String[] strings){
	try {
	Connection conn = ConnectionFactory.getConnection();
	for (int i = 2; i < strings.length; i++){
		CallableStatement statement = conn.prepareCall("{call registra_telefono(?, ?)}");
```

```
            statement.setString(1, strings[i]);
            statement.setString(2, strings[0]);

            statement.execute();
}
}
catch (SQLException e){
System.out.println("error:" + e.getMessage().toString());
}
}

public void AddCellphone(String[] strings){
try {
Connection conn = ConnectionFactory.getConnection();
for (int i = 2; i < strings.length; i++){
            CallableStatement statement = conn.prepareCall("{call registra_cellulare(?, ?)}");
            statement.setString(1, strings[i]);
            statement.setString(2, strings[0]);

            statement.execute();
}
}
catch (SQLException e){
System.out.println("error:" + e.getMessage().toString());
}
}

public void AddEmail(String[] strings){
try {
Connection conn = ConnectionFactory.getConnection();
for (int i = 2; i < strings.length; i++){
            CallableStatement statement = conn.prepareCall("{call registra_email(?, ?)}");
            statement.setString(1, strings[i]);
            statement.setString(2, strings[0]);
```

```
                statement.execute();
        }
        }
        catch (SQLException e){
        System.out.println("error:" + e.getMessage());
        }
        }
}
```

```
package model.dao;

import model.domain.Role;

import java.io.*;
import java.sql.*;
import java.util.Properties;

public class ConnectionFactory {
	private static Connection connection;

	private ConnectionFactory() {}

	static {
	try (InputStream input = new FileInputStream("src/main/resources/db.properties")) {
	Properties properties = new Properties();
	properties.load(input);

	String connection_url = properties.getProperty("CONNECTION_URL");
	String user = properties.getProperty("LOGIN_USER");
	String pass = properties.getProperty("LOGIN_PASS");

	connection = DriverManager.getConnection(connection_url, user, pass);
	} catch (IOException | SQLException e) {
	e.printStackTrace();
	}
	}

	public static Connection getConnection() throws SQLException {
	return connection;
	}

	public static void changeRole(Role role) throws SQLException {
	connection.close();
```

```
        try (InputStream input = new FileInputStream("src/main/resources/db.properties")) {
        Properties properties = new Properties();
        properties.load(input);

        String connection_url = properties.getProperty("CONNECTION_URL");
        String user = properties.getProperty(role.name() + "_USER");
        String pass = properties.getProperty(role.name() + "_PASS");

        connection = DriverManager.getConnection(connection_url, user, pass);
        } catch (IOException | SQLException e) {
        e.printStackTrace();
        }
        }
}
```

```
package model.dao;

import java.sql.*;

public class ExamDAO {
	public void getExams(){
	try {
	Connection conn = ConnectionFactory.getConnection();
	CallableStatement statement = conn.prepareCall("{call lista_esami()}");
	ResultSet rs = statement.executeQuery();
	while (rs.next()) {
		int codiceEsame = rs.getInt("CodiceEsame");
		String descrizione = rs.getString("Descrizione");

		System.out.println(codiceEsame + ": " + descrizione);
	}
	}
	catch (SQLException e){
	System.out.println("error:" + e.getMessage().toString());
	}
	}

	public void getExamsReport(String code){
	try {
	Connection conn = ConnectionFactory.getConnection();
	CallableStatement statement = conn.prepareCall("{call storico_esami(?)}");
	statement.setString(1, code);
	ResultSet rs = statement.executeQuery();
	while (rs.next()) {
		Timestamp dataOra = rs.getTimestamp("DataOra");
		int codiceEsame = rs.getInt("Esame_CodiceEsame");
		String descrizione = rs.getString("Descrizione");
		double costo = rs.getDouble("Costo");
		boolean urgenza = rs.getBoolean("Urgenza");
```

```
                System.out.println(dataOra + " " + codiceEsame + " " + descrizione + " " + costo + " "
+ urgenza);
        }

        }
        catch (SQLException e){
        System.out.println("error:" + e.getMessage().toString());
        }
        }

        public void addExam(String name){
        try {
        Connection conn = ConnectionFactory.getConnection();
        CallableStatement statement = conn.prepareCall("{call registra_esame(?)}");
        statement.setString(1, name);

        statement.execute();
        }
        catch (SQLException e){
        System.out.println("error:" + e.getMessage().toString());
        }
        }

        public void AddDoneExam(String[] exam) {
        try {
        Connection conn = ConnectionFactory.getConnection();
        CallableStatement statement = conn.prepareCall("{call esami_svolti(?, ?)}");
        statement.setInt(1, Integer.parseInt(exam[0]));
        statement.setString(2, exam[1]);
        statement.execute();
        }
        catch (SQLException e){
        System.out.println("error: " + e.getMessage().toString());
```

```
        }
        }
}
```

```
package model.dao;

import model.domain.ExamDiagnosis;
import model.domain.ExamResult;

import java.sql.CallableStatement;
import java.sql.Connection;
import java.sql.SQLException;

public class ExamResultsDao {
	public void addExamResult(ExamResult result){
	try {
	Connection conn = ConnectionFactory.getConnection();
	for (int i = 0; i < result.getNames().size(); i++) {
		CallableStatement statement = conn.prepareCall("{call registra_risultato(?, ?, ?, ?, ?)}");
		statement.setInt(1, result.getReservationCode());
		statement.setTimestamp(2, result.getDateTime());
		statement.setInt(3, result.getExamCode());
		statement.setString(4, result.getNames().get(i));
		statement.setBigDecimal(5, result.getValues().get(i));

		statement.execute();
	}
	}
	catch (SQLException e){
	System.out.println("error:" + e.getMessage().toString());
	}
	}

	public void addExamDiagnosis(ExamDiagnosis diagnosis) {
	try {
	Connection conn = ConnectionFactory.getConnection();
	CallableStatement statement = conn.prepareCall("{call registra_diagnosi(?, ?, ?, ?)}");
```

```
        statement.setString(1, diagnosis.getText());
        statement.setTimestamp(2, diagnosis.getDateTime());
        statement.setInt(3, diagnosis.getReservationCode());
        statement.setInt(4, diagnosis.getExamCode());

        statement.execute();
        }
        catch (SQLException e){
        System.out.println("error:" + e.getMessage().toString());
        }
        }
}
```

```
package model.dao;

import model.domain.Credentials;
import model.domain.Role;

import java.sql.*;

public class LoginProcedureDAO {

	public Credentials execute(Object... params){
	String username = (String) params[0];
	String password = (String) params[1];
	int role = 0;

	try {
	Connection conn = ConnectionFactory.getConnection();
	CallableStatement cs = conn.prepareCall("{call login(?,?,?)}");
	cs.setString(1, username);
	cs.setString(2, password);
	cs.registerOutParameter(3, Types.NUMERIC);
	cs.executeQuery();
	role = cs.getInt(3);
	} catch(SQLException e) {
	System.out.println("Login error: " + e.getMessage().toString());
	}
	return new Credentials(username, password, Role.fromInt(role));
	}
}
```

```
package model.dao;

import model.domain.Reservation;

import java.sql.CallableStatement;
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;

public class ReservationDAO {

	public void reservationReport(int code) {
	try {
	Connection conn = ConnectionFactory.getConnection();
	CallableStatement cs = conn.prepareCall("{call report_prenotazioni(?)}");
	cs.setInt(1, code);
	ResultSet rs = cs.executeQuery();
	while (rs.next()) {
		String nome = rs.getString("Nome");
		double valore = rs.getDouble("Valore");

		System.out.println(nome + ": " + valore);
	}
	}
	catch (SQLException e){
	System.out.println("error:" + e.getMessage());
	}
	}

	public void AddReservation(Reservation reservation){
	try {
	Connection conn = ConnectionFactory.getConnection();
	CallableStatement statement = conn.prepareCall("{call registra_prenotazione(?, ?, ?, ?, ?,
?)}");
```

```
        statement.setInt(1, reservation.getReservationCode());
        statement.setTimestamp(2, reservation.getDateTime());
        statement.setInt(3, reservation.getExamCode());
        statement.setBigDecimal(4, reservation.getCost());
        statement.setBoolean(5, reservation.isUrgent());
        statement.setString(6, reservation.getPatientCode());

        statement.execute();
        }
        catch (SQLException e){
        System.out.println("error: " + e.getMessage().toString());
        }
        }
}
```

```
package model.dao;

import java.sql.*;

public class StaffDao {
	public void getStaffReport(String basis){
	try {
	Connection conn = ConnectionFactory.getConnection();
	CallableStatement statement;

	if(basis.equals("m")) {
		statement = conn.prepareCall("{call report_personale_mensile()}");
		ResultSet rs = statement.executeQuery();
		printM(rs);
	}
	else if(basis.equals("y")) {
		statement = conn.prepareCall("{call report_personale_annuale()}");
		ResultSet rs = statement.executeQuery();
		printY(rs);
	}
	else {
		statement = conn.prepareCall("{call report_personale_mensile()}");
		ResultSet rs = statement.executeQuery();
		System.out.println("MONTHLY BASIS:");
		printM(rs);

		statement = conn.prepareCall("{call report_personale_annuale()}");
		rs = statement.executeQuery();
		System.out.println("ANNUAL BASIS:");
		printY(rs);
	}

	}
	catch (SQLException e){
```

```
        System.out.println("error:" + e.getMessage().toString());
        }
        }

        private void printY(ResultSet rs) {
        try {
        while (rs.next()) {
                String codiceFiscale = rs.getString("Personale_CodiceFiscale");
                int anno = rs.getInt("Anno");
                int numeroEsami = rs.getInt("NumeroEsami");

                System.out.println(codiceFiscale + " - " + anno + " - " + numeroEsami);
        }
        } catch (SQLException e) {
        System.out.println("error: " + e.getMessage().toString());
        }
        }

        private void printM(ResultSet rs) {
        try {
        while (rs.next()) {
                String codiceFiscale = rs.getString("Personale_CodiceFiscale");
                int mese = rs.getInt("Mese");
                int numeroEsami = rs.getInt("NumeroEsami");

                System.out.println(codiceFiscale + " - " + mese + " - " + numeroEsami);
        }
        } catch (SQLException e) {
        System.out.println("error: " + e.getMessage().toString());
        }
        }
}
```

- **DOMAIN**

```
package model.domain;

public class Credentials {

	private final String username;
	private final String password;
	private final Role role;

	public Credentials(String username, String password, Role role) {
	this.username = username;
	this.password = password;
	this.role = role;
	}

	public String getUsername() {
	return username;
	}

	public String getPassword() {
	return password;
	}

	public Role getRole() {
	return role;
	}
}
```

```
package model.domain;

import java.sql.Timestamp;

public class ExamDiagnosis {
	private final Timestamp dateTime;
	private final int reservationCode;
	private final int examCode;
	private final String text;

	public ExamDiagnosis(String text, Timestamp dateTime, int reservationCode, int examCode)
{
	this.dateTime = dateTime;
	this.reservationCode = reservationCode;
	this.examCode= examCode;
	this.text = text;
	}

	public String getText() {
	return text;
	}

	public int getReservationCode() {
	return reservationCode;
	}

	public int getExamCode() {
	return examCode;
	}

	public Timestamp getDateTime() {
	return dateTime;
	}
}
```

```
public class ExamResult {
	private final Timestamp dateTime;
	private final int reservationCode;
	private final int examCode;
	private final List<String> names;
	private final List<BigDecimal> values;
	public ExamResult(int reservationCode,Timestamp dateTime, int examCode, List<String> names, List<BigDecimal> values) {
	this.dateTime = dateTime;
	this.reservationCode = reservationCode;
	this.examCode= examCode;
	this.names = names;
	this.values = values;
	}

	public List<String> getNames() {
	return names;
	}
	public List<BigDecimal> getValues() {
	return values;
	}

	public int getReservationCode() {
	return reservationCode;
	}

	public int getExamCode() {
	return examCode;
	}

	public Timestamp getDateTime() {
	return dateTime;
	}
}
```

```
package model.domain;

import java.sql.Date;

public class Patient {
	private final String name;
	private final String surname;
	private final String sanitaryCode;
	private final Date birthDate;
	private final String birthPlace;
	private final String address;


	public Patient(String sanitaryCode, String name, String surname, Date birthDate, String birthPlace, String address) {
	this.sanitaryCode = sanitaryCode;
	this.name = name;
	this.surname = surname;
	this.birthDate = birthDate;
	this.address = address;
	this.birthPlace = birthPlace;
	}
	public String getName() {
	return name;
	}

	public Date getBirthDate() {
	return birthDate;
	}

	public String getBirthPlace() {
	return birthPlace;
	}
```

```
        public String getSanitaryCode() {
        return sanitaryCode;
        }

        public String getSurname() {
        return surname;
        }
}
```

```
package model.domain;

import java.math.BigDecimal;
import java.sql.Timestamp;

public class Reservation {

	private final int reservationCode;
	private final Timestamp dateTime;
	private final int examCode;
	private final BigDecimal cost;
	private final boolean isUrgent;
	private final String patientCode;

	public Reservation(int reservationCode, Timestamp dateTime, int examCode, BigDecimal cost, boolean isUrgent, String patientCode){
	this.reservationCode = reservationCode;
	this.cost = cost;
	this.dateTime = dateTime;
	this.examCode = examCode;
	this.patientCode = patientCode;
	this.isUrgent = isUrgent;
	}
	public Timestamp getDateTime() {
	return dateTime;
	}

	public int getExamCode() {
	return examCode;
	}

	public BigDecimal getCost() {
	return cost;
```

```
        }

        public boolean isUrgent() {
        return isUrgent;
        }

        public String getPatientCode() {
        return patientCode;
        }

        public int getReservationCode() {
        return reservationCode;
        }
}
```

```
package model.domain;

public enum Role {
	AMMINISTRATORE(1),
	CUP(2),
	MEDICO(3);



	private final int id;

	private Role(int id) {
	this.id = id;
	}

	public static Role fromInt(int id) {
	for (Role type : values()) {
	if (type.getId() == id) {
		return type;
	}
	}
	return null;
	}

	public int getId() {
	return id;
	}
}
```

- **VIEW**

```
package view;

import java.io.IOException;
import java.util.Scanner;

public class AmministratoreView {
	public static int showMenu() throws IOException {
	System.out.println("**********************************");
	System.out.println("*  ADMINISTRATOR DASHBOARD  *");
	System.out.println("**********************************\n");
	System.out.println("*** What should I do for you? ***\n");
	System.out.println("1) Add Exam");
	System.out.println("2) Show Staff Report");
	System.out.println("3) Quit");



	Scanner input = new Scanner(System.in);
	int choice;
	while (true) {
	System.out.print("Please enter your choice: ");
	choice = input.nextInt();
	if (choice >= 1 && choice <= 3) {
		break;
	}
	System.out.println("Invalid option");
	}

	return choice;
	}

	public static String showAddExamMenu() {
	System.out.println("Insert the name of the Exam you want to add:\n");
	Scanner input = new Scanner(System.in);
```

```
        return input.nextLine();
        }

        public static String showStaffReportChoice() {
        Scanner input = new Scanner(System.in);
        String in;
        while (true) {
        System.out.print("Select the type of report you want: [ m (on a monthly basis) / y (on an annual basis) / my (both)]: \n");
        in = input.nextLine();
        if (in.equals("m") || in.equals("y") || in.equals("my")){
                break;
        }
        System.out.println("You must enter 'm', 'y' or 'my'");
        }
        return in;
        }
}
```

```
package view;

import model.domain.Patient;
import model.domain.Reservation;

import java.io.IOException;
import java.math.BigDecimal;
import java.sql.Timestamp;
import java.text.ParseException;
import java.sql.Date;
import java.util.Scanner;

public class CUPView {

	public static int showMenu() throws IOException {
	System.out.println("**********************************");
	System.out.println("*  CUP STAFF DASHBOARD  *");
	System.out.println("**********************************\n");
	System.out.println("*** What should I do for you? ***\n");
	System.out.println("1) Add patient");
	System.out.println("2) Add patient infos");
	System.out.println("3) List exams");
	System.out.println("4) Add reservation");
	System.out.println("5) reservations report");
	System.out.println("6) exams report");
	System.out.println("7) Quit");



	Scanner input = new Scanner(System.in);
	int choice = 0;
	while (true) {
	System.out.print("Please enter your choice: ");
	choice = input.nextInt();
	if (choice >= 1 && choice <= 7) {
```

```
            break;
        }
        System.out.println("Invalid option");
        }
        return choice;
        }

        public static Patient showAddPatientMenu() throws ParseException {
        System.out.println("Please enter patient information: sanitaryCode name surname birthDate(yyyy-MM-dd) birthPlace addresslikethis9\n");

        Scanner input = new Scanner(System.in);

        String in = input.nextLine();
        String[] entry = in.split(" ");
        return new Patient(entry[0], entry[1], entry[2], Date.valueOf(entry[3]), entry[4], entry[5]);
        }

        public static String[] showAddPatientInfoMenu() {
        System.out.println("Enter the Patient information like this: sanitaryCode [phone/cellphone/email] *information* *information* ...\n");
        Scanner input = new Scanner(System.in);
        String in = input.nextLine();
        return in.split(" ");
        }

        public static int showReservationReportMenu() {
        Scanner input = new Scanner(System.in);
        int in;
        while (true) {
        System.out.print("Insert the reservation code you want a report about: \n");
        in = input.nextInt();
        if (in >= 0) {
            break;
```

```
        }
        System.out.println("the reservation code must be a positive number");
        }
        return in;
        }

        public static String showExamReportMenu() {
        Scanner input = new Scanner(System.in);

        String in;
        while (true) {
        System.out.print("Insert the sanitary code of the patient you want a report about: \n");
        in = input.nextLine();
        if (in.matches("^[a-zA-Z0-9]{20}$")){
                break;
        }
        System.out.println("the sanitary code must be an alphanumeric string of 20 characters");
        }

        return in;
        }
        public static Reservation showAddReservationMenu() {
        System.out.println("Please enter reservation information: reservationCode,Date(yyyy-MM-dd
hh:mm:ss),ExamCode cost(cdu.dc),urgency(true or false),sanitaryCode\n");

        Scanner input = new Scanner(System.in);

        String in = input.nextLine();
        String[] entry = in.split(",");
        BigDecimal bigDecimal = new BigDecimal(entry[2]);
        return new Reservation(Integer.parseInt(entry[0]), Timestamp.valueOf(entry[1]),
Integer.parseInt(entry[2]), bigDecimal, Boolean.parseBoolean(entry[4]), entry[5]);
        }
}
```

```
package view;


import model.domain.Credentials;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

public class LoginView {
	public static Credentials authenticate() throws IOException {
	BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));
	System.out.print("username: ");
	String username = reader.readLine();
	System.out.print("password: ");
	String password = reader.readLine();

	return new Credentials(username, password, null);
	}
}
```

```
package view;

import model.domain.ExamDiagnosis;
import model.domain.ExamResult;

import java.io.IOException;
import java.math.BigDecimal;
import java.sql.Timestamp;
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

public class MedicoView {
	public static int showMenu() throws IOException {
	System.out.println("**********************************");
	System.out.println("* MEDICAL STAFF DASHBOARD   *");
	System.out.println("**********************************\n");
	System.out.println("*** What should I do for you? ***\n");
	System.out.println("1) Add Exam Results");
	System.out.println("2) Add Exam Diagnosis");
	System.out.println("3) Add Done Exam");
	System.out.println("4) Quit");


	Scanner input = new Scanner(System.in);
	int choice;
	while (true) {
	System.out.print("Please enter your choice: ");
	choice = input.nextInt();
	if (choice >= 1 && choice <= 4) {
		break;
	}
	System.out.println("Invalid option");
	}
```

```
return choice;
}

public static ExamResult showAddExamResultMenu() {
System.out.println("Enter the results information like this: reservationCode,Date(yyyy-MM-dd hh:mm:ss),ExamCode,*ResultName,ResultValue*,*ResultName,ResultValue* ...\n");
Scanner input = new Scanner(System.in);
String in = input.nextLine();
String[] entry = in.split(",");
List<BigDecimal> values = new ArrayList<>();
List<String> names = new ArrayList<>();
for (int i = 4; i < entry.length; i = i + 2){
names.add(entry[i-1]);
BigDecimal bigDecimal = new BigDecimal(entry[i]);
values.add(bigDecimal);
}
return new ExamResult(Integer.parseInt(entry[0]), Timestamp.valueOf(entry[1]), Integer.parseInt(entry[2]), names, values);
}

public static ExamDiagnosis showAddExamDiagnosisMenu() {
System.out.println("Enter the diagnosis like this: Diagnosis,Date(yyyy-MM-dd hh:mm:ss),ReservationCode,ExamCode\n");
Scanner input = new Scanner(System.in);
String in = input.nextLine();
String[] entry = in.split(",");
return new ExamDiagnosis(entry[0], Timestamp.valueOf(entry[1]), Integer.parseInt(entry[2]), Integer.parseInt(entry[2]));
}

public static String[] showAddDoneExamMenu() {
System.out.println("Enter the done Exam like this: ExamCode YourFiscalCode\n");
```

```
        Scanner input = new Scanner(System.in);
        String in = input.nextLine();
        return in.split(" ");
        }
}
```

- **MAIN**

```
import controller.ApplicationController;

public class Main {

	public static void main(String[] args) {
	ApplicationController applicationController = new ApplicationController();
	applicationController.start();
	}
}
```