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

Basi di Dati e Conoscenza

Progetto A.A. 2018/2019

TITOLO DEL PROGETTO

0214381

Salvatore Foderaro

**Indice**

[1. Descrizione del Minimondo 2](#_Toc606296459)

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

[3. Progettazione concettuale 9](#_Toc2081466291)

[4. Progettazione logica 11](#_Toc2147004904)

[5. Progettazione fisica 16](#_Toc518560220)

[Appendice: Implementazione 18](#_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 | Realizzare un servizio di “directory aziendale”, che consenta di tenere traccia di tutte le  informazioni legate ai recapiti ed alle mansioni di un’azienda.  Ogni dipendente dell’azienda, identificato da codice fiscale, nome, cognome, data di nascita,  luogo di nascita, indirizzo di residenza svolge una particolare mansione all’interno dell’azienda.  Tali mansioni vengono svolte in differenti uffici dell’azienda. L’azienda ha a disposizione un  numero arbitrario di edifici, ciascuno di un numero arbitrario di piani. In ciascun piano può  esserci un numero arbitrario di uffici. Ciascun ufficio è assegnato ad una mansione specifica  dell’azienda. Un ufficio ha a disposizione un certo numero di postazioni, ciascuno associato ad  un numero telefonico interno ed esterno. La gestione degli uffici, piani, edifici è demandata al  Settore Spazi dell’azienda, anch’esso composto da dipendenti assegnati a particolari uffici.  L’assegnazione di un dipendente ad un ufficio avviene su base di turnazioni periodiche. Ogni  dipendente, pertanto, è associato alla data dell’ultimo trasferimento. [Periodicamente, il sistema  indica qual è l’insieme dei dipendenti che deve essere soggetto a trasferimento.] Su base  periodica, l’ufficio spazi genera un report indicante tutti i dipendenti, raggruppati per mansione,  che devono essere spostati.  Un dipendente dell’ufficio spazi deve poter effettuare un trasferimento di dipendente.  Taletrasferimento deve essere effettuato rispettando l’associazione tra le mansioni e gli uffici.  Pertanto, dato un dipendente da trasferire, il dipendente del Settore Spazi può effettuare uno  scambio tra due dipendenti assegnati alla stessa mansione, o individuare una postazione libera  utilizzabile per quella mansione. Si noti che un dipendente non può essere riassegnato ad una  postazione in cui era già stato assegnato nei passati tre anni.  Il Settore Amministrativo può decidere in qualsiasi momento di modificare la mansione di un  dipendente. In quel caso, il Settore Spazi troverà tale dipendente indicato come da trasferire.  Inoltre, il settore amministrativo può generare, per ciascun dipendente, un report indicante a  quali uffici esso è stato assegnato nel tempo.  Ciascun dipendente ha accesso alla directory aziendale per conoscere l’attuale ubicazione di un  qualsiasi altro dipendente, ricercandolo per nome, per cognome o utilizzando entrambi. Inoltre,  un dipendente può ricercare un certo numero di telefono interno per sapere a quale ufficio,  piano, edificio e mansione questo è associato, scoprendo anche quale dipendente è attualmente  associato allo stesso e sapendo se esso è in procinto di essere trasferito o meno  Si noti che, quando si effettua la ricerca di un dipendente, è necessario restituire tutti i recapiti  associati allo stesso. In particolare, un dipendente ha un indirizzo email personale ed uno  associato all’ufficio a cui fa capo. |

# Analisi dei Requisiti

## Identificazione dei termini ambigui e correzioni possibili

|  |  |  |  |
| --- | --- | --- | --- |
| **Linea** | **Termine** | **Nuovo termine** | **Motivo correzione** |
| 2 | Di un’azienda | Dei dipendenti di un’azienda | Le informazioni delle quali si vuole tenere traccia sono relative ai dipendenti che lavorano all’interno dell’azienda |
| 5 | Tali mansione | Ogni mansione |  |
| 8 | Ciascuno | Ed è | Ad ogni ufficio è associate un numero telefonico interno ed esterno, non alla singola postazione |
| 12 | Data dell’ultimo trasferimento | Storico delle assegnazioni | Per rispettare il vincolo temporale della riassegnazione ad una postazione, devo mantenere non solo l’ultimo trasferimento, ma tutti quanti |
| 28 | Numero di telefono | Numero di telefono interno |  |
| 29 | Quale dipendente | Quali dipendenti | All’interno di un ufficio sono presenti più di un dipendente, non un singolo |

### Specifica disambiguata

|  |
| --- |
| Realizzare un servizio di “directory aziendale”, che consenta di tenere traccia di tutte le informazioni legate ai recapiti ed alle mansioni dei dipendenti di un’azienda.  Ogni dipendente dell’azienda, identificato da codice fiscale, nome, cognome, data di nascita, luogo di nascita, indirizzo di residenza svolge una particolare mansione all’interno dell’azienda. Ogni mansione viene svolta in differenti uffici dell’azienda. L’azienda ha a disposizione un numero arbitrario di edifici, ciascuno di un numero arbitrario di piani. In ciascun piano può esserci un numero arbitrario di uffici. Ciascun ufficio è assegnato ad una mansione specifica dell’azienda. Un ufficio ha a disposizione un certo numero di postazioni, ed è associato ad un numero telefonico interno ed esterno. La gestione degli uffici, piani, edifici è demandata al Settore Spazi dell’azienda, anch’esso composto da dipendenti assegnati a particolari uffici.  L’assegnazione di un dipendente ad un ufficio avviene su base di turnazioni periodiche. Per ogni dipendente viene mantenuto lo storico delle assegnazioni. Periodicamente, il sistema indica qual è l’insieme dei dipendenti che deve essere soggetto a trasferimento. Su base periodica, l’ufficio spazi genera un report indicante tutti i dipendenti, raggruppati per mansione, che devono essere spostati.  Un dipendente dell’ufficio spazi deve poter effettuare un trasferimento di dipendente. Tale trasferimento deve essere effettuato rispettando l’associazione tra le mansioni e gli uffici. Pertanto, dato un dipendente da trasferire, il dipendente del Settore Spazi può effettuare uno scambio tra due dipendenti assegnati alla stessa mansione, o individuare una postazione libera utilizzabile per quella mansione. Si noti che un dipendente non può essere riassegnato ad una postazione in cui era già stato assegnato nei passati tre anni.  Il Settore Amministrativo può decidere in qualsiasi momento di modificare la mansione di un dipendente. In quel caso, il Settore Spazi troverà tale dipendente indicato come da trasferire. Inoltre, il settore amministrativo può generare, per ciascun dipendente, un report indicante a quali uffici esso è stato assegnato nel tempo.  Ciascun dipendente ha accesso alla directory aziendale per conoscere l’attuale di un qualsiasi altro dipendente, ricercandolo per nome, per cognome o utilizzando entrambi. Inoltre, un dipendente può ricercare un certo numero di telefono interno per sapere a quale ufficio, piano, edificio e mansione questo è associato, scoprendo anche quali dipendenti sono attualmente associati allo stesso e sapendo se, e chi, è in procinto di essere trasferito.  Si noti che, quando si effettua la ricerca di un dipendente, è necessario restituire tutti i recapiti associati allo stesso. In particolare, un dipendente ha un indirizzo email personale ed uno associato all’ufficio a cui fa capo. |

## Glossario dei Termini

|  |  |  |  |
| --- | --- | --- | --- |
| **Termine** | **Descrizione** | **Sinonimi** | **Collegamenti** |
| Dipendente | Una persona che lavora all’interno dell’azienda |  | Postazione, Ufficio, Mansione |
| Mansione | Tipo di lavoro svolto all’interno dell’azienda |  | Ufficio, Dipendente |
| Postazione | Postazione di lavoro del singolo dipendente |  |  |
| Ufficio | Luogo fisico dove sono presenti diverse postazioni che svolgono la stessa mansione | Ubicazione | Postazione, Mansione, Dipendente, |
| Assegnazione | Operazione che associa un dipendente ad una postazione per lo svolgimento di una particolare mansion | Trasferimento | Dipendente, Postazione, Ufficio |

## Raggruppamento dei requisiti in insiemi omogenei

|  |
| --- |
| **Frasi relative a ... Dipendente** |
| * Ogni dipendente dell’azienda, identificato da codice fiscale, nome, cognome, data di nascita, luogo di nascita, indirizzo di residenza svolge una particolare mansione all’interno dell’azienda. * Ogni dipendente svolge una particolare mansione * Un Dipendente può essere assegnato al Settore spazi, Settore amministrativo o ad un settore che svolge una qualsiasi altra mansione * Per ogni dipendente viene mantenuto lo storico delle assegnazioni. * Un dipendente dell’ufficio spazi deve poter effettuare un trasferimento di dipendente * Dato un dipendente da trasferire, il dipendente del Settore Spazi può effettuare uno scambio tra due dipendenti assegnati alla stessa mansione, o individuare una postazione libera all’interno di un ufficio che svolge la stessa mansione. * Un dipendente non può essere riassegnato ad una postazione in cui era già stato assegnato nei passati tre anni * Ciascun dipendente ha accesso alla directory aziendale per conoscere l’attuale ubicazione di un qualsiasi altro dipendente * Un dipendente può ricercare un certo numero di telefono interno per sapere a quale ufficio, piano, edificio e mansione questo è associato * A ciascun dipendente è associato un indirizzo email personale ed uno relativo all’ufficio di appartenenza * Quando si effettua la ricerca di un dipendente, è necessario restituire tutti i recapiti associati allo stesso |

|  |
| --- |
| **Frasi relative a ... Mansione** |
| * Ogni mansione è svolta da un numero diversi di dipendenti * Ogni mansione viene svolta in differenti uffici dell’azienda * La mansione di un dipendente può essere modificata da un dipendente del Settore Amministrativo |

|  |
| --- |
| **Frasi relative a ... Ufficio** |
| * Un ufficio ha a disposizione un certo numero di postazioni * A ciascun ufficio è associato un numero telefonico interno ed esterno. * A ciascun ufficio è associato un indirizzo email |

|  |
| --- |
| **Frasi relative a ... Postazione** |
| * Ogni postazione è associata ad un ufficio * Ad ogni postazione è assegnato un dipendente che svolge una particolare mansione |

|  |
| --- |
| **Frasi relative a ... Assegnazione** |
| * L’assegnazione di un dipendente ad una postazione avviene su base di turnazioni periodiche * Un dipendente non può essere riassegnato ad una postazione in cui era già stato assegnato nei passati tre anni. |

# Progettazione concettuale

## Costruzione dello schema E-R

 Per poter indicare I vari tipi di dipendente all’interno della base di dati è stata utilizzata una generalizzazione. L’entità principale è **Dipendente**, da cui le tre specializzazioni ereditano tutti gli attributi.



Per distinguere gli impieghi passati di un dipendente da quelli attualmente in svolgimento è stata utilizzata una generalizzazione, con **Impiego corrente** ed **Impiego passato** figli dell’entità principale; **Impiego passato**, oltre ad ereditare chiave ed attributi dell’entità padre, possiede l’attributo *Termine*.

Per introdurre il concetto di postazione abilitata/non abilitata e rispettare I vincoli di chiave per gli impieghi passati del singolo dipendente, **Postazione attiva** specializza l’entità **Postazione**, aggiungendo l’attributo relativo alla disponibilità.



Per le identità **Ufficio** e **Mansione** sono stati aggiunte chiavi ed attributi richiesti dalla specifica.



Per l’integrazione dello schema, **Postazione** viene messa in relazione con **Ufficio** con cardinalità (1,1), e risulta debole rispetto a quest’ultima.

### 

Per mantenre l’archivio degli impieghi passati e rispettare I vincoli del sistema, **Impiego corrente** viene posto in relazione diretta con **Postazione Attiva**, mentre il generico **Impiego** con la generica **Postazione**, sia essa attiva o meno.

Per poter permettere l’inserimento di un Dipendente all’interno del sistema senza dover necessariamente associare una mansione, le relazioni di **Dipendente** con **Impiego Corrente** ed **Impiego** hanno cardinalità minima 0.

### Integrazione finale

## Regole aziendali

* Un dipendente non deve essere assegnato ad una postazione se il tempo trascorso dall’ultima assegnazione alla stessa postazione è inferiore ai tre anni
* L’assegnazione dei dipendenti deve avvenire solamente alle postazioni attive
* Un dipendente deve essere assegnato ad una postazione con la sua stessa mansione
* I dipendenti dei settori spazi ed amministrativo devono essere associati alle relative mansioni
* **Numero Postazioni Libere** si ottiene tramite l’associazione Composizione
* **Un Ufficio** deve decrementare il *numero di Postazioni* libere dopo l’assegnazione di un dipendente ad una sua postazione attiva precedentemente libera
* **Indirizzo E-Mail ufficio (Dipendente)** si ottiene da **Ufficio** tramite le associazioni **Impiego corrente**, **Assegnazione** e **Composizione**
* Un **Dipendente** deve impostare l’attributo **Da Trasferire** a **True** dopo che la sua mansione è stata modificata
* Una **Postazione attiva** deve impostare l’attributoDisponibile a **False** dopo l’assegnazione di un dipendente ad essa
* Dopo l’assegnazione ad una nuova postazione, l’impiego corrente deve essere inserito tra gli impieghi passati

## Dizionario dei dati

|  |  |  |  |
| --- | --- | --- | --- |
| **Entità** | **Descrizione** | **Attributi** | **Identificatori** |
| Dipendente | Rappresenta l’entità del singolo dipendente all’interno della base di dati | Nome, Cognome, Data di nascita, Luogo di nascita, Indirizzo residenza, Indirizzo e-mail, Indirizzo e-mail ufficio, Da Trasferire | Codice Fiscale |
| Dipendente Settore Amministrativo | Generalizzazione di dipendente | Eredita da: Dipendente | Eredita da: Dipendente |
| Dipendete Settore Spazi | Generalizzazione di dipendente | Eredita da: Dipendente | Eredita da: Dipendente |
| Altro Dipendente | Generalizzazione di dipendente | Eredita da: Dipendente | Eredita da: Dipendente |
| Impiego | Rappresenta il singolo impiego svolto da un utente dopo l’associazione ad una postazione | Inizio | ID, Assegnazione (Postazione), Matricola (Dipendente) |
| Impiego corrente | Generalizzazione di Impiego, rappresenta il singolo impiego attualmente in svolgimento da parte del dipendente | Eredita da: Impiego | Eredita da: Impiego |
| Impiego passato | Generalizzazione di impiego, rappresenta tutti gli impieghi svolti dal singolo dipendente | Eredita da: Impiego, Fine | Eredita da: Impiego |
| Postazione | Rappresentalela postazioni presenti nel sistema, e possono essere attive o meno |  | ID, Ufficio (Ufficio) |
| Postazione attiva | Rappresenta le postazioni attive all’interno dei vari uffici, con associato o meno un dipendente | Disponibile | Eredita da: Postazione |
| Ufficio | Rappresenta l’ufficio all’interno del quale sono presenti un numero variabile di postazioni; svolte una particolare mansione | Telefono interno, Telefono esterno, E-mail, Piano, Edificio, Numero postazioni libere | ID |
| Mansione | Rappresenta le mansioni svolte dai vari dipendenti | Nome mansione | ID |

# Progettazione logica

## Volume dei dati

|  |  |  |
| --- | --- | --- |
| **Concetto nello schema** | **Tipo[[1]](#footnote-2)** | **Volume atteso** |
| D-I | R | 15 \* 2000 |
| Assegnazione | R | 3 \* 2000 |
| Composizione | R | 2100 |
| Svolgimento | R | 105 |
| Occupazione | R | 2000 |
| Dipendente | E | 2100 |
| Dipendente Settore Spazi | E | 100 |
| Dipendente Settore Amministrativo | E | 1900 |
| Altro Dipendente | E | 50 |
| Impiego | E | 15 \* 2000 |
| Impiego Attuale | E | 2000 |
| Impiego passato | E | 14 \* 2000 |
| Postazione | E | 2100 |
| Postazione attiva | E | 1800 |
| Ufficio | E | 110 |
| Mansione | E | 10 |
| Impiego attuale | R | 2000 |

## Tavola delle operazioni

|  |  |  |
| --- | --- | --- |
| **Cod.** | **Descrizione** | **Frequenza attesa** |
| 1 | Generazione Report Utenti da trasferire | 1/giorno |
| 2 | Trasferimento Dipendente (Altro dipendente) | 20/giorno |
| 3 | Trasferimento Dipendente (Postazione Libera) | 20/giorno |
| 4 | Modifica Mansione Dipendente | 20/giorno |
| 5 | Generazione Report Assegnazioni Passate | 50/giorno |
| 6 | Ricerca Altro dipendente | 200/giorno |
| 7 | Ricerca ufficio | 200/giorno |
| 8 | Disabilita postazione | 5/giorno |
| 9 | Abilita postazione | 5/giorno |
| 10 | Inserisci postazione | 2/giorno |
| 11 | Assegnare postazione ad un ufficio | 2/giorno |

## Costo delle operazioni

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Trasferimento Dipendente (Altro dipendente)** | | | | |
| **Concetto nello schema** | **Tipo** | **Tipo di accesso** | **Descrizione** | **N° accessi** |
| Dipendente | E | Lettura | Leggo l’attuale mansione del dipenente 1 | 1 |
| Occupazione | R | Lettura | Leggo la postazione attuale del dipendente 1 | 1 |
| Assegnazione Attuale | R | Lettura | Trovo I dipendenti non associati alla postazione nei precedenti tre anni | 13 |
| Dipendente | E | Lettura | Controllo che il dipendente 2 sia da trasferire | 1 |
| Dipendente | E | Lettura | Leggo l’attuale mansione del dipendente 2 | 1 |
| Assegnazione | R | Lettura | Leggo la postazione attuale del dipendente 2 | 1 |
| Assegnazione | R | Scrittura | Controllo che il dipendente 1 non sia stato associato nei precedenti tre anni all’attuale postazione del dipendente 2 | 13 |
| Impiego | E | Scrittura | Modifico l’impiego corrente dei dipendenti 1 e 2 | 2\*2 |
| Impiego passato | E | Scrittura | Scrivo l’attuale impiego come impiego passato | 2\*1 |
| Impiego Passato | E | Scrittura | Modifico l’assegnazione della postazione attuale dei dipendenti 1 e 2 | 2\*2 |
| Impiego Attuale | E | Scrittura | Modifico gli indirizzi E-Mail relativi all’ufficio dei dipendenti 1 e 2 | 2\*2 |
| Costo operazione: 72 | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Trasferimento Dipendente (Postazione Libera)** | | | | |
| **Concetto** | **Tipo** | **Tipo di accesso** | **Descrizione** | **N° accessi** |
| Mansione dipendente | R | Lettura | Leggo l’attuale mansione del dipendente | 1 |
| Svolgimento | R | Lettura | Leggo tutti gli uffici, con postazioni attive disponibili, che svolgono la stessa mansione del dipendente | 11 |
| Composizione | R | Lettura | Leggo tutte le postazioni libere dell’ufficio | 17 |
| Impiego corrente | E | Lettura | Leggo l’impiego corrente del dipendente | 1 |
| Impiego passato | E | Scrittura | Scrivo impiego corrente in impiego passato | 1 |
| Impiego corrente + Del | E | Scrittura | Modifico l’impiego corrente | 2 |
| Impiego attuale + Del | E | Scrittura | Modifico l’assegnazione dell’impiego attuale | 2 |
| Postazione attiva | E | Scrittua | Imposto la postazione come non disponibile | 1 |
| Ufficio | E | Scrittura | Decremento il numero di postazioni attive disponibili nell’ufficio | 1 |
| Dipendente | E | Scrittura | Modifico l’indirizzo e-mail relativo all’ufficio | 1 |
| Costo operazione: 46 | | | | |

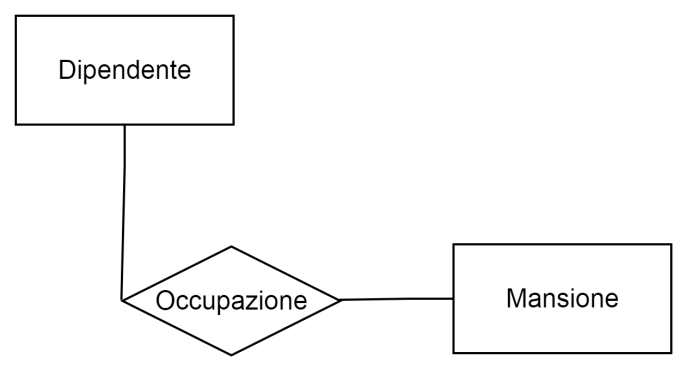
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Modifica Mansione Dipendente** | | | | |
| **Concetto** | **Tipo** | **Tipo di accesso** | **Descrizione** | **N° accessi** |
| Mansione dipendente | R | Lettura | Leggo l’attuale mansione del dipendente | 1 |
| Dipendente | E | Scrittura | Imposto il dipendente come da trasferire | 1 |
| Mansione dipendente + Del | R | Scrittura | Scrivo la nuova mansione del dipendente | 1 |
| Costo operazione: 5 | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Abilita postazione/Disabilita postazione** | | | | |
| **Concetto** | **Tipo** | **Tipo di accesso** | **Descrizione** | **N° accessi** |
| Postazione | E | Lettura | Leggo la postazione | 1 |
| Postazione attiva | E | Scrittura | Inserisco/elimino la postazione la postazione | 1 |
| Composizione | R | Lettura | Leggo l’ufficio associato alla postazione | 17 |
| Ufficio | E | Scrittura | Aggiorno il numero di postazioni disponibili all’interno dell’ufficio | 1 |
| Costo operazione: 22 | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Inserisci postazione** | | | | |
| **Concetto** | **Tipo** | **Tipo di accesso** | **Descrizione** | **N° accessi** |
| Postazione | E | Scrittura | Inserisco la postazione | 1 |
| Costo operazione: 2 | | | | |

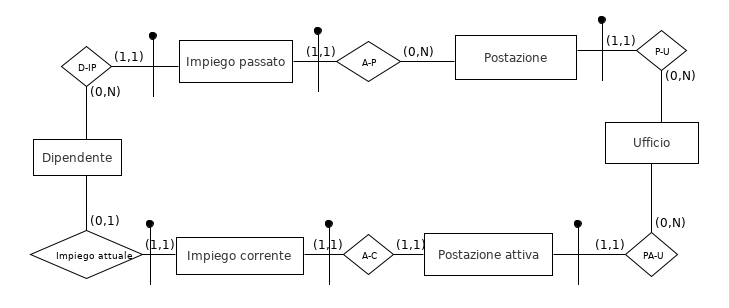
## Ristrutturazione dello schema E-R

**Ridondanza Associazione *Occupazione/ E-Mail ufficio / Postazioni disponibili***

La presenza della relazione **Mansione dipendente** genera una ridondanza; a fronte di un minimo spreco di memoria, permette di accedere direttamente all’informazione relativa al dipendente, senza dover navigare lo scherma per intero. La ricorrente necessità di accesso all’attributo permette di ottimizzare I tempi delle varie operazioni.

Altre ridondanze presenti all’interno dello schema sono relative agli attributi **E-Mail ufficio** e **Postazioni disponibili**, rispettivamente delle entità *Dipendente* ed *Ufficio:* la prima è necessaria per rispettare il vincolo aziendale (***Indirizzo E-Mail ufficio (Dipendente)*** *si ottiene da* ***Ufficio*** *tramite le associazioni* ***Impiego*)** ed evitare di dover navigare l’intero schema per ottenere l’informazione; la seconda permette di ottenere il numero di postazioni libere all’interno del singolo ufficio accedendo ad un singolo attributo, non dovendo dunque andare ogni volta a calcolare il numero.

**Eliminazione Generalizzazioni**

Per permettere una stoicizzazione delle postazioni, oltre agli impieghi, viene efettuata una separazione tra le postazioni effettivamente attive, con assegnazioni tutt’ora valide, e postazioni che contengono anche postazioni non più attive.

Per quanto riguarda la generalizzazione per il tipo di **Dipendente**, viene effettuato l’accorpamento dei figli nel genitore, aggiungendo gli attributi necessari per distinguere il generico dipendente dalle altre due tipologie di impiegati.

**Identificatori Primari**

|  |  |
| --- | --- |
| **Entità** | **Identificatore Primario** |
| Dipendente | Matricola |
| Impiego Attuale | MatricolaDipendente (Dipendente), IDPostazione (Postazione) |
| Impiego Passato | MatricolaDipendente (Dipendente), IDPostazione (Postazione) |
| Postazione | ID, IDUfficio (Ufficio) |
| Postazione attiva | ID, IDUfficio (Ufficio) |
| Ufficio | ID |
| Mansione | ID |

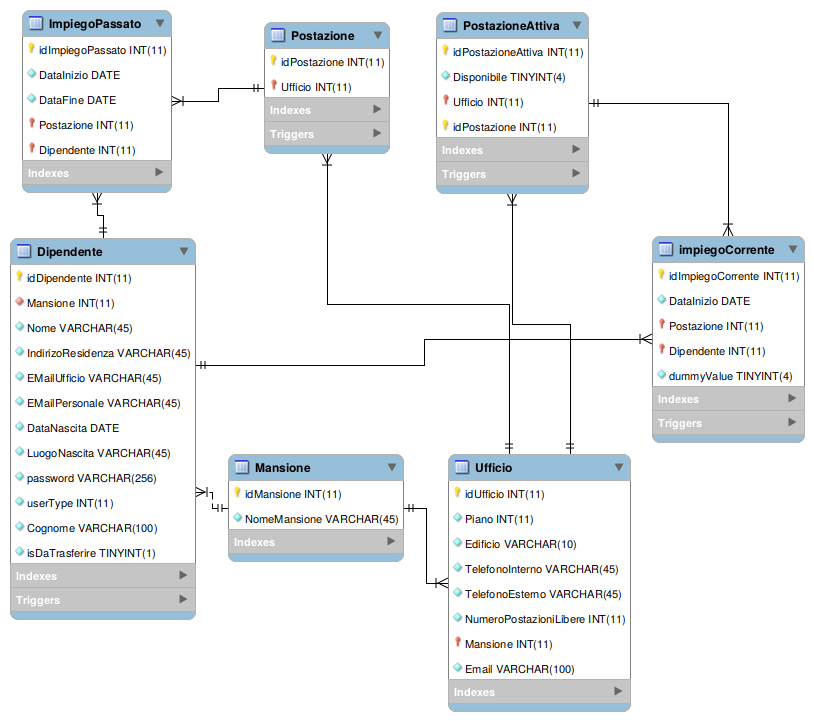
## Trasformazione di attributi e identificatori

|  |  |  |
| --- | --- | --- |
| **Entità** | **Identificatore** | **Nuovo Identificatore** |
| Impiego corrente | Matricola (Dipendente) | ID |
| Impiego passato | Matricola (Dipendente) | ID |
| Postazione | ID (Ufficio) | ID |

## Traduzione di entità e associazioni

|  |  |
| --- | --- |
| **Nome** |  |
| Dipendente | Matricola, Nome, cognome, Indirizzo residenza, Da trasferire, E-Mail personale, E-Mail ufficio, Data di nascita, Luogo di nascita, userType, **Mansione** |
| Mansione | ID, Nome mansione |
| Ufficio | ID, Piano, Edificio, Telefono interno, Telefono esterno, Numero postazioni libere, **Mansione** |
| Postazione | ID, **Ufficio** |
| Postazione attiva | ID, Disponibile, **Ufficio** |
| Impiego attuale | ID, **Dipendente**, Inizio, **Postazione** |
| Impiego passato | ID, **Dipendente**, Inizio, Fine, **Postazione** |

\***Grassetto** indica chiave esterna, Sottolineato indica chiave primaria



## Normalizzazione del modello relazionale

**2NF**

|  |  |  |  |
| --- | --- | --- | --- |
| **Tabella<PostazioneAttiva>** | | | |
| **IdUfficio** | **Mansione** | **Piano** | Edificio |
| 1 | 1 | 1 | 1 |
| 2 | 1 | 1 | 1 |
| 3 | 2 | 2 | 1 |
| 4 | 3 | 2 | 1 |

Nel caso in cui la singola mansione viene svolta in un piano ubicato in un particolare edificio, è possibile normalizzare lo schema nel seguente modo:

|  |  |  |
| --- | --- | --- |
| **Mansione** | **Piano** | Edificio |
| 1 | 1 | 1 |
| 1 | 1 | 1 |
| 2 | 2 | 1 |
| 3 | 2 | 1 |

|  |  |  |  |
| --- | --- | --- | --- |
| **IdUfficio** | **Mansione** | | |
| **1** | 1 | | |
| 2 | | 1 |  |
| 3 | | 2 |  |
| 4 | | 3 |  |

Nel caso in cui la singola mansione viene svolta in un particolare edificio, è possibile normalizzare lo schema nel seguente modo:

|  |  |  |
| --- | --- | --- |
| **IdUfficio** | **Mansione** | Piano |
| 1 | 1 | 1 |
| 2 | 1 | 1 |
| 3 | 2 | 1 |
| 4 | 3 | 1 |

|  |  |  |
| --- | --- | --- |
| **Mansione** | **Edificio** | |
| 1 | 1 | |
| 1 | 1 |  |
| 2 | 2 |  |
| 3 | 2 |  |

# Progettazione fisica

## Utenti e privilegi

|  |  |  |  |
| --- | --- | --- | --- |
| **Utente<tempUser>** | | | |
| **Lettura** | **Aggiornamento** | **Eliminazione** | Inserimento |
| **Dipendente** |  |  |  |
| **Motivazione:** L’utente temUser è necessario per la sola operazione di Login, dunque deve solamente accedere alla tabella Dipendente per controllare la corrispondenza tra il numero del dipendente e la password associata | | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Utente<otherUser>** | | | |
| **Lettura** | **Aggiornamento** | **Eliminazione** | Inserimento |
| **Dipendente, ImpiegoPassato, ImpiegoCorrente, Postazione, Postazione Attiva, Mansione** |  |  |  |
| **Motivazione:** Tutti gli altri utenti, che non appartengono al settore amministrativo o al settore spazi, hanno solamente la possibilità di effettuare delle ricerche all’interno della base di dati, senza poter effettuare operazioni di modifica dei dati**.** | | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Utente<amministrativoUser>** | | | |
| **Lettura** | **Aggiornamento** | **Eliminazione** | Inserimento |
| **Dipendente, ImpiegoPassato, ImpiegoCorrente, Postazione, Postazione Attiva, Mansione** | Dipendente |  | Dipendente |
| **Motivazione:** L’utente del settore amministrativo deve poter modificare l’assegnazione del dipendente ad una mansione ed indicare l’utente come da trasferire, oltre ad avere accesso alla ricerca di altri utenti. (Registrazione dipendente) | | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Utente<spaziUser>** | | | |
| **Lettura** | **Aggiornamento** | **Eliminazione** | Inserimento |
| **Dipendente, ImpiegoPassato, ImpiegoCorrente, Postazione, Postazione Attiva, Mansione** | Postazione, PostazioneAttiva, Ufficio, ImpiegoCorrente, ImpiegoPassato | Postazione, PostazioneAttiva | Postazione, ImpiegoPassato, PostazioneAttiva |
| **Motivazione:** L’utente del settore spazi, deve poter modificare l’assegnazione dei singoli utenti alle varie postazioni ed aggiungere/rimuovere/modificare le varie postazioni, oltre ad avere accesso alla ricerca di altri utenti. | | | |

## Strutture di memorizzazione

|  |  |  |
| --- | --- | --- |
| **Tabella <Dipendente>** | | |
| **Attributo** | **Tipo di dato** | **Attributi[[2]](#footnote-3)** |
| **idDipendente** | Int | NN, AI, PK |
| **Mansione** | Int | NN |
| **Nome** | Varchar | NN |
| **IndirizzoResidenza** | Varchar | NN |
| **EmailPersonale** | Varchar | NN |
| **DataNascita** | Dte | NN |
| **LuogoNascita** | Varchar | NN |
| **Password** | Varchar | NN |
| **UserType** | Int | NN |
| **Cognome** | Varchar | NN |

|  |  |  |
| --- | --- | --- |
| **Tabella <ImpiegoPassato>** | | |
| **Attributo** | **Tipo di dato** | **Attributi[[3]](#footnote-4)** |
| **IdImpiegoPassato** | Int | NN, AI, PK |
| **Postazione** | Int | NN, PK |
| **Dipendente** | Int | NN, PK |
| **DataInizio** | Date | NN |
| **DataFine** | Date | NN |

|  |  |  |
| --- | --- | --- |
| **Tabella <Mansione>** | | |
| **Attributo** | **Tipo di dato** | **Attributi[[4]](#footnote-5)** |
| **IdMansione** | Int | NN, AI, PK |
| **NomeMansione** | Int | NN |

|  |  |  |
| --- | --- | --- |
| **Tabella <Postazione>** | | |
| **Attributo** | **Tipo di dato** | **Attributi[[5]](#footnote-6)** |
| **IdPostazione** | Int | NN, AI, PK |
| **Ufficio** | Int | NN, PRI |

|  |  |  |
| --- | --- | --- |
| **Tabella <PostazioneAttiva>** | | |
| **Attributo** | **Tipo di dato** | **Attributi[[6]](#footnote-7)** |
| **IdPostazioneAttiva** | Int | NN, AI, PK |
| **Disponibile** | Int | NN |
| **Ufficio** | Int | NN, PK |
| **IdPostazione** | Int | NN, PK |

|  |  |  |
| --- | --- | --- |
| **Tabella <Ufficio>** | | |
| **Attributo** | **Tipo di dato** | **Attributi[[7]](#footnote-8)** |
| **IdUfficio** | Int | NN, AI, PK |
| **Piano** | Int | NN |
| **Edificio** | Varchar | NN |
| **TelefonoInterno** | Varchar | NN |
| **TelefonoEsterno** | Varchar | NN |
| **NumeroPostazioni** | Varchar | NN |
| **Mansione** | Int | NN, PK |
| **Email** | Varchar | NN |

|  |  |  |
| --- | --- | --- |
| **Tabella <ImpiegoCorrente>** | | |
| **Attributo** | **Tipo di dato** | **Attributi[[8]](#footnote-9)** |
| **IdImpiegoCorrente** | Int | NN, AI, PK |
| **DataInizio** | Int | NN |
| **Postazione** | Int | NN, PK |
| **Dipendente** | Int | NN, PK |
| **DummyValue** | TinyInt | NN |

## Indici

|  |  |
| --- | --- |
| **Tabella <Dipendente>** | |
| **Indice <nome>** | **Tipo[[9]](#footnote-10):** |
| IdDipendente | UQ |
| Mansione | IDX |

|  |  |
| --- | --- |
| **Tabella <impiegoCorrente>** | |
| **Indice <nome>** | **Tipo[[10]](#footnote-11):** |
| IdImpiegoCorrente | UQ |
| Postazione | IDX |
| Dipendente | IDX |

|  |  |
| --- | --- |
| **Tabella <ImpiegoPassato>** | |
| **Indice <nome>** | **Tipo[[11]](#footnote-12):** |
| IdImpiegoPassato | UQ |
| Postazione | IDX |
| Dipendente | IDX |

|  |  |
| --- | --- |
| **Tabella <Postazione>** | |
| **Indice <nome>** | **Tipo[[12]](#footnote-13):** |
| IdPostazione | UQ |
| Ufficio | IDX |

|  |  |
| --- | --- |
| **Tabella <PostazioneAttiva>** | |
| **Indice <nome>** | **Tipo[[13]](#footnote-14):** |
| IdPostazioneAttiva | UQ |
| Ufficio | IDX |
| IdPostazione | IDX |

|  |  |
| --- | --- |
| **Tabella <Ufficio>** | |
| **Indice <nome>** | **Tipo[[14]](#footnote-15):** |
| IdUfficio | UQ |
| Mansione | IDX |

## Trigger

**ImpiegoCorrente (Before Insert):** necessario per controllare che il dipendente relativo al nuovo impiego da inserire sia presente all’interno del sistema ed indicato come da trasferire.

**CREATE** **DEFINER**=`root`@`localhost` **TRIGGER** `mydb`.`impiegoCorrente\_BEFORE\_INSERT` **BEFORE** **INSERT** **ON** `impiegoCorrente` **FOR** **EACH** **ROW**

**BEGIN**

**IF** **NOT** **EXISTS** (**SELECT** 1 **FROM** Dipendente **WHERE** idDipendente = **NEW**.Dipendente **AND** isDaTrasferire **is** **true**) **THEN**

signal **sqlstate** '45000'

**SET** MESSAGE\_TEXT = 'L\'ID del dipendente non è valido o non disponibile per il trasferimento!';

**END** **IF**;

**END**

**ImpiegoCorrente (After Insert):** necessario per soddisfare il vincolo aziendale relativo alla necessità che l’attributo “EmailUfficio” del singolo dipendente sia ereditata dall’ufficio nel quale lavora. Necessario anche per la ridondanza dei dati e poter aggiornare il numero di postazioni libere all’interno dell’ufficio.

**CREATE** **DEFINER**=`root`@`localhost` **TRIGGER** `mydb`.`impiegoCorrente\_AFTER\_INSERT` **AFTER** **INSERT** **ON** `impiegoCorrente` **FOR** **EACH** **ROW**

**BEGIN**

**UPDATE** PostazioneAttiva **set** Disponibile = **false** **WHERE** idPostazioneAttiva = **NEW**.Postazione;

**UPDATE** Dipendente **set** eMailUfficio = (**SELECT** ufficio.Email **FROM** PostazioneAttiva **JOIN** Ufficio **as** ufficio **on** PostazioneAttiva.Ufficio = ufficio.idUfficio **WHERE** PostazioneAttiva.idPostazioneAttiva = **NEW**.Postazione), isDaTrasferire = **FALSE** **WHERE** idDipendente = **NEW**.Dipendente;

**END**

**ImpiegoCorrente (Before Update):** necessario per soddisfare il vincolo aziendale relativo alla non possibilità di assegnare un dipendente alla stessa postazione se vi era stato già assegnato nei due anni precedenti

**CREATE** **DEFINER**=`root`@`localhost` **TRIGGER** `mydb`.`impiegoCorrente\_BEFORE\_UPDATE` **BEFORE** **UPDATE** **ON** `impiegoCorrente` **FOR** **EACH** **ROW**

**BEGIN**

**DECLARE** mansioneDipendente **int**;

**SET** mansioneDipendente = getMansioneUtente(**NEW**.Dipendente);

**IF** (**NEW**.Postazione != **OLD**.Postazione **AND** **OLD**.dummyValue = 0) **THEN**

**IF** **NOT** **EXISTS**(**SELECT** idPostazioneAttiva **FROM** PostazioneAttiva

**JOIN** Ufficio **on** PostazioneAttiva.Ufficio = Ufficio.idUfficio

**JOIN** Mansione **on** Ufficio.Mansione = mansioneDipendente

**WHERE** (

**NOT** **EXISTS** (**SELECT** \* **FROM** ImpiegoPassato IP **WHERE** IP.dipendente = **NEW**.Dipendente **and** IP.postazione = **NEW**.Postazione)

**or**

**EXISTS** (**SELECT** \* **from** ImpiegoPassato IP **WHERE** IP.dipendente = **NEW**.Dipendente **and** IP.postazione = **NEW**.Postazione **and** IP.DataInizio < **DATE**(DATE\_sub(NOW(), **INTERVAL** 2 **YEAR**)))

)) **THEN**

signal **sqlstate** '45000'

**SET** MESSAGE\_TEXT = 'Il dipendente è stato già assegnato a questa postazione nei precedenti due anni!';

**END** **IF**;

**SET** **NEW**.DataInizio = NOW();

**INSERT** **INTO** ImpiegoPassato(DataInizio, DataFine, Postazione, Dipendente) **VALUES** (**OLD**.dataInizio, NOW(), **OLD**.Postazione, **OLD**.Dipendente);

**UPDATE** Dipendente **set** eMailUfficio = (**SELECT** ufficio.Email **FROM** PostazioneAttiva **JOIN** Ufficio **as** ufficio **on** PostazioneAttiva.Ufficio = ufficio.idUfficio **WHERE** PostazioneAttiva.idPostazioneAttiva = **NEW**.Postazione), isDaTrasferire = **FALSE** **WHERE** idDipendente = **NEW**.Dipendente;

**SET** **NEW**.dummyValue = 1;

**END** **IF**;

**IF** (**NEW**.Postazione != **OLD**.Postazione **AND** **OLD**.dummyValue != 0) **THEN**

**IF** **NOT** **EXISTS**(**SELECT** idPostazioneAttiva **FROM** PostazioneAttiva

**JOIN** Ufficio **on** PostazioneAttiva.Ufficio = Ufficio.idUfficio

**JOIN** Mansione **on** Ufficio.Mansione = mansioneDipendente

**WHERE** (

**NOT** **EXISTS** (**SELECT** \* **FROM** ImpiegoPassato IP **WHERE** IP.dipendente = **NEW**.Dipendente **and** IP.postazione = **NEW**.Postazione)

**or**

**EXISTS** (**SELECT** \* **from** ImpiegoPassato IP **WHERE** IP.dipendente = **NEW**.Dipendente **and** IP.postazione = **NEW**.Postazione **and** IP.DataInizio < **DATE**(DATE\_sub(NOW(), **INTERVAL** 2 **YEAR**)))

**AND** Disponibile **is** **TRUE**) **THEN**

signal **sqlstate** '45000'

**SET** MESSAGE\_TEXT = 'Il dipendente è stato già assegnato a questa postazione nei precedenti due anni!';

**END** **IF**;

**SET** **NEW**.DataInizio = NOW();

**INSERT** **INTO** ImpiegoPassato(DataInizio, DataFine, Postazione, Dipendente) **VALUES** (**OLD**.dataInizio, NOW(), **OLD**.Postazione, **OLD**.Dipendente);

**UPDATE** PostazioneAttiva **set** Disponibile = **false** **WHERE** idPostazioneAttiva = **NEW**.Postazione;

**UPDATE** PostazioneAttiva **set** Disponibile = **true** **WHERE** idPostazioneAttiva = **OLD**.Postazione;

**UPDATE** Dipendente **set** eMailUfficio = (**SELECT** ufficio.Email **FROM** PostazioneAttiva **JOIN** Ufficio **as** ufficio **on** PostazioneAttiva.Ufficio = ufficio.idUfficio **WHERE** PostazioneAttiva.idPostazioneAttiva = **NEW**.Postazione), isDaTrasferire = **FALSE** **WHERE** idDipendente = **NEW**.Dipendente;

**END** **IF**;

**IF** (**NEW**.Dipendente != **OLD**.Dipendente) **THEN**

**IF** **NOT** **EXISTS** (**SELECT** 1 **FROM** Dipendente **WHERE** idDipendente = **NEW**.Dipendente **and** isDaTrasferire **is** **true**) **THEN**

signal **sqlstate** '45000'

**SET** MESSAGE\_TEXT = 'L\'ID del dipendente non è valido o non disponibile per il trasferimento!';

**END** **IF**;

**END** **IF**;

**END**

**PostazioneAttiva (Before Insert):** necessario per controllare che la postazione indicata per l’attivazione sia effettivamente attiva

**CREATE** **DEFINER**=`root`@`localhost` **TRIGGER** `mydb`.`PostazioneAttiva\_BEFORE\_INSERT` **BEFORE** **INSERT** **ON** `PostazioneAttiva` **FOR** **EACH** **ROW**

**BEGIN**

**IF** **NOT** **EXISTS** (**SELECT** 1 **FROM** Postazione **WHERE** idPostazione = **NEW**.idPostazioneAttiva **and** Ufficio != 0) **THEN**

signal **sqlstate** '45000'

**SET** MESSAGE\_TEXT = 'Nessuna postazione associata all\'ID indicato!';

**END** **IF**;

**END**

**PostazioneAttiva (After Insert):** necessario per la ridondanza dei dati, per aggiornare il numero di postazioni libere all’interno del singolo ufficio

**CREATE** **DEFINER**=`root`@`localhost` **TRIGGER** `mydb`.`PostazioneAttiva\_AFTER\_INSERT` **AFTER** **INSERT** **ON** `PostazioneAttiva` **FOR** **EACH** **ROW**

**BEGIN**

**IF** **NEW**.Disponibile **is** **true** **THEN**

**UPDATE** Ufficio **set** NumeroPostazioniLibere = NumeroPostazioniLibere + 1 **WHERE** idUfficio = **NEW**.Ufficio;

**ELSE**

**UPDATE** Ufficio **set** NumeroPostazioniLibere = NumeroPostazioniLibere - 1 **WHERE** idUfficio = **NEW**.Ufficio;

**END** **IF**;

**END**

**PostazioneAttiva (After Update):**

**CREATE** **DEFINER**=`root`@`localhost` **TRIGGER** `mydb`.`PostazioneAttiva\_AFTER\_UPDATE\_1` **AFTER** **UPDATE** **ON** `PostazioneAttiva` **FOR** **EACH** **ROW**

**BEGIN**

**IF** (**OLD**.Disponibile != **NEW**.Disponibile **and** **NEW**.Disponibile = **false**) **then**

**UPDATE** Ufficio **set** NumeroPostazioniLibere = NumeroPostazioniLibere - 1 **WHERE** idUfficio = **NEW**.Ufficio;

**END** **IF**;

**IF** (**OLD**.Disponibile != **NEW**.Disponibile **and** **NEW**.Disponibile = **true**) **then**

**UPDATE** Ufficio **set** NumeroPostazioniLibere = numeroPostazioniLibere + 1 **Where** idUfficio = **NEW**.Ufficio;

**END** **IF;**

**END**

**PostazioneAttiva (Before Delete):** necessario per controllare, prima dell’eliminazione di una postazione attiva, che non ci sia nessun dipendente associato ad essa

**CREATE** **DEFINER**=`root`@`localhost` **TRIGGER** `mydb`.`PostazioneAttiva\_BEFORE\_DELETE` **BEFORE** **Delete** **ON** `PostazioneAttiva` **FOR** **EACH** **ROW**

**BEGIN**

**IF** (**OLD**.Disponibile **is** **false**) **THEN**

signal **sqlstate** '45000'

**SET** MESSAGE\_TEXT = 'Non è possibile disabilitare una postazione attualmente assegnata ad un dipendente!';

**END** **IF**;

**UPDATE** Ufficio **SET** NumeroPostazioniLibere = NumeroPostazioniLibere - 1 **WHERE** idUfficio = **OLD**.Ufficio;

**END**

**Postazione (Before Update):** necessario per controllare, prima della modifica dell’assegnazione di una postazione ad un ufficio, che l’ufficio di destinazione esita effettivamente e che la postazione non sia attualmente attiva

**CREATE** **DEFINER**=`root`@`localhost` **TRIGGER** `mydb`.`Postazione\_BEFORE\_UPDATE` **BEFORE** **UPDATE** **ON** `Postazione` **FOR** **EACH** **ROW**

**BEGINxc**

**IF** (**NEW**.Ufficio != **OLD**.Ufficio) **THEN**

**IF** **EXISTS** (**SELECT** 1 **FROM** PostazioneAttiva **WHERE** idPostazioneAttiva = **OLD**.idPostazione **and** Disponibile **is** **false**) **THEN**

signal **sqlstate** '45000'

**SET** MESSAGE\_TEXT = 'Non è assegnare spostare una postazione attualmente occupata!';

**END** **IF**;

**IF** **NOT** **EXISTS** (**SELECT** 1 **FROM** Ufficio **WHERE** idUfficio = **NEW**.Ufficio) **THEN**

signal **sqlstate** '45000'

**SET** MESSAGE\_TEXT = 'L\'ID dell\'ufficio indicato non è corretto!';

**END** **IF**;

**END** **IF**;

**END**

## Viste

**GetPostazioni:** la vista permette di ottenere la lista delle postazioni attualmente presenti nella base di dati.

**CREATE** **OR** **REPLACE**

**VIEW** `getPostazioni` **AS** **select**

`Postazione`.`idPostazione` **AS** `idPostazione`,

`Postazione`.`Ufficio` **AS** `Ufficio`

**from**

`Postazione`

**GetPostazioniAttive:** la vista permette di ottenere la lista delle postazioni attualmente attive presenti nelal base di dati.

**CREATE** **OR** **REPLACE**

**VIEW** `getPostazioniAttive` **AS** **select**

`PostazioneAttiva`.`idPostazioneAttiva` **AS** `idPostazioneAttiva`,

`PostazioneAttiva`.`Disponibile` **AS** `Disponibile`,

`PostazioneAttiva`.`Ufficio` **AS** `Ufficio`

**from**

`PostazioneAttiva`

**getToTransfer:** la vista permette di ottenere la lista dei dipendenti, presenti nella base di dati, che devono essere trasferiti.

**CREATE** **OR** **REPLACE**

**VIEW** `getToTransfer` **AS** **select**

`Dipendente`.`idDipendente` **AS** `idDipendente`

**from**

`Dipendente`

**where**

`mydb`.`Dipendente`.`daTrasferire` **is** **true**

## Stored Procedures e transazioni

**MakeScambioUser:** viene utilizzata un’operazione transazionale complessa per assicurare la consistenza delle informazioni all’interno del database in modo, come in questo caso, di non “lasciare senza nessuna mansione I dipendenti”

**CREATE** **DEFINER**=`root`@`localhost` **PROCEDURE** `mydb`.`makeScambioUser`(user1 **int**, user2 **int**)

**BEGIN**

**DECLARE** postazione1 **int**;

**DECLARE** postazione2 **int**;

**DECLARE** validUser1 **int**;

**DECLARE** validUser2 **int**;

**DECLARE** **EXIT** HANDLER **FOR** **SQLEXCEPTION**

**BEGIN**

**ROLLBACK**;

**END**;

**SET** validUser1 = checkValidUser(user1);

**SET** validUser2 = checkValidUser(user2);

**SET** postazione1 = getCurrentPostazione(user1);

**SET** postazione2 = getCurrentPostazione(user2);

**IF** (postazione1 **is** **not** **null** **and** postazione2 **is** **not** **null**) **THEN**

**UPDATE** impiegoCorrente **SET** impiegoCorrente.dummyValue = 0 **WHERE** impiegoCorrente.Dipendente = user1;

**UPDATE** impiegoCorrente **SET** impiegoCorrente.Postazione = postazione2 **WHERE** impiegoCorrente.Dipendente = user1;

**UPDATE** impiegoCorrente **SET** impiegoCorrente.dummyValue = 0 **WHERE** impiegoCorrente.Dipendente = user2;

**UPDATE** impiegoCorrente **SET** impiegoCorrente.Postazione = postazione1 **WHERE** impiegoCorrente.Dipendente = user2;

**ELSE**

**ROLLBACK**;

**END** **IF**;

**COMMIT**;

**END**

**AssignPostazioneToUfficio:** viene utilizzata un’operazione transazionale complessa in quanto è necessario l’aggiornamento dell’informazione relativa alla stessa postazione ma di due tabelle differenti

**CREATE** **DEFINER**=`root`@`localhost` **PROCEDURE** `mydb`.`assignPostazioneToUfficio`(**IN** postazione **Int**, **IN** ufficioID **Int**)

**BEGIN**

**DECLARE** validPostazione **INT**;

**DECLARE** **EXIT** HANDLER **FOR** **SQLEXCEPTION**

**BEGIN**

**ROLLBACK**;

**END**;

**IF** **EXISTS** (**SELECT** 1 **FROM** Postazione **WHERE** Postazione.ID = postazione **and** Postazione.Ufficio = ufficioID) **then**

signal **sqlstate** '45000'

**SET** MESSAGE\_TEXT = 'Si sta assegnando al dipendente la sua mansione attuale!';

**END** **IF**;

**SET** validPostazione = checkValidPostazione(postazione);

**SET** FOREIGN\_KEY\_CHECKS = 0;

**UPDATE** PostazioneAttiva **SET** PostazioneAttiva.Ufficio = ufficioID **WHERE** PostazioneAttiva.idPostazione = postazione;

**UPDATE** Postazione **SET** Postazione.Ufficio = ufficioID **WHERE** Postazione.idPostazione = postazione;

**SET** FOREIGN\_KEY\_CHECKS = 1;

**COMMIT**;

**END**

**AssignToPostazioneFree:** viene utilizzata un’operazione transazionale complessa in quanto è necessario l’aggiornamento dell’informazione relativa alla stessa postazione ma di due tabelle differenti

**CREATE** **DEFINER**=`root`@`localhost` **PROCEDURE** `mydb`.`assignToPostazioneFree`(**IN** dipendente **Int**, **IN** newPostazione **Int**)

**BEGIN**

**DECLARE** mansioneDipendente **Int**;

**DECLARE** littleResult **INT**;

**DECLARE** postazioneDipendente **INT**;

**DECLARE** validUser **INT**;

**DECLARE** **EXIT** HANDLER **FOR** **SQLEXCEPTION**

**BEGIN**

**ROLLBACK**;

**END**;

**SET** postazioneDipendente = getCurrentPostazione(dipendente);

**SET** mansioneDipendente = getMansioneUtente(dipendente);

**SET** littleResult = checkPostazione(newPostazione);

**SET** validUser = checkValidUser(dipendente);

**IF** postazioneDipendente **is** **null** **then**

**INSERT** **INTO** impiegoCorrente(DataInizio, Postazione, Dipendente) **VALUES** (NOW(), newPostazione, dipendente);

**UPDATE** Dipendente **SET** isDaTrasferire = **FALSE** **WHERE** idDipendente = dipendente;

**END** **IF**;

**IF** postazioneDipendente **is** **not** **null** **THEN**

**UPDATE** impiegoCorrente **SET** impiegoCorrente.Postazione = newPostazione **WHERE** impiegoCorrente.Dipendente = dipendente;

**UPDATE** Dipendente **SET** isDaTrasferire = **FALSE** **WHERE** idDipendente = dipendente;

**END** **IF**;

**COMMIT**;

**END**

**CheckPostazioniDisponibili:** viene utilizzata per ottenere la lista delle postazioni disponibili per l’assegnazione del dipendente indicato

**CREATE** **DEFINER**=`root`@`localhost` **PROCEDURE** `mydb`.`checkPostazioniDisponibili`(**IN** newDipendente **Int**)

**BEGIN**

**DECLARE** postazioneDipendente **Int**;

**DECLARE** mansioneDipendente **Int**;

**DECLARE** counter **int**;

**DECLARE** postazioneExists **int**;

/\* Controllo che l'utenta esista e che sia indicato come da trasferire \*/

**SET** counter = checkValidUser(newDipendente);

**SET** mansioneDipendente = getMansioneUtente(newDipendente);

**SET** postazioneDipendente = getCurrentPostazione(newDipendente);

**SELECT** Postazione **INTO** postazioneDipendente **from** impiegoCorrente **where** Dipendente = newDipendente;

**IF** postazioneDipendente **is** **null** **then**

**SELECT** idPostazioneAttiva **FROM** PostazioneAttiva

**JOIN** Ufficio **on** PostazioneAttiva.Ufficio = Ufficio.idUfficio

**JOIN** Mansione **on** Ufficio.Mansione = mansioneDipendente

**WHERE** (

**NOT** **EXISTS** (**SELECT** \* **FROM** ImpiegoPassato IP **WHERE** IP.dipendente = newDipendente **and** IP.postazione = idPostazioneAttva)

**or**

**EXISTS** (**SELECT** \* **from** ImpiegoPassato IP **WHERE** IP.dipendente = newDipendente **and** IP.postazione = idPostazioneAttiva **and** IP.DataInizio < **DATE**(DATE\_sub(NOW(), **INTERVAL** 2 **YEAR**)))

)

**AND** Disponibile **is** **TRUE**;

**ELSE**

**SELECT** idPostazioneAttiva **FROM** PostazioneAttiva

**JOIN** Ufficio **on** PostazioneAttiva.Ufficio = Ufficio.idUfficio

**JOIN** Mansione **on** Ufficio.Mansione = mansioneDipendente

**WHERE** idPostazioneAttiva != postazioneDipendente

**AND** (

**NOT** **EXISTS** (**SELECT** \* **FROM** ImpiegoPassato IP **WHERE** IP.dipendente = newDipendente **and** IP.postazione = idPostazioneAttiva)

**or**

**EXISTS** (**SELECT** \* **from** ImpiegoPassato IP **WHERE** IP.dipendente = newDipendente **and** IP.postazione = idPostazioneAttiva **and** IP.DataInizio < **DATE**(DATE\_sub(NOW(), **INTERVAL** 2 **YEAR**)))

)

**AND** Disponibile **is** **TRUE**;

**END** **IF**;

**END**

**DisablePostazione:** viene utilizzata per disabilitare una postazione

**CREATE** **DEFINER**=`root`@`localhost` **PROCEDURE** `mydb`.`disablePostazione`(**IN** postazione **Int**)

**BEGIN**

**DECLARE** validPostazione **INT**;

/\* Aggiungere per il controllo se la posizione poteva essere disabilitata \*/

/\* Anche qui aggiungere il check \*/

**SET** validPostazione = checkValidPostazione(postazione);

**DELETE** **FROM** PostazioneAttiva **WHERE** idPostazioneAttiva = postazione;

**END**

**EditMansione:** viene utilizzata per modificare la mansione di un dipendente

**CREATE** **DEFINER**=`root`@`localhost` **PROCEDURE** `mydb`.`editMansione`(**IN** userID **int**, **IN** idMansione **int**)

**BEGIN**

**DECLARE** validUser **int**;

**DECLARE** test **int**;

**set** validUser = checkUserExists(userID);

/\* Al momento è implementato nel trigger, ed è Ok.

\* Il controllo del dipendente però va fatto prima, lui comunque non mi da errore

\* Però, se ho il check di una foreign key, lo potrei evitare \*/

**SET** test = checkValidMansione(idMansione);

**IF** **EXISTS** (**SELECT** 1 **FROM** Dipendente **WHERE** idDipendente = userID **and** Mansione = idMansione) **then**

signal **sqlstate** '45000'

**SET** MESSAGE\_TEXT = 'Si sta assegnando al dipendente la sua mansione attuale!';

**END** **IF**;

**UPDATE** Dipendente **SET** Mansione = idMansione **WHERE** idDipendente = userID;

**END**

**EnablePostazione:** viene utilizzata per abilitare una postazione

**CREATE** **DEFINER**=`root`@`localhost` **PROCEDURE** `mydb`.`enablePostazione`(**IN** postazione **Int**)

**BEGIN**

**DECLARE** validPostazione **INT**;

**DECLARE** idUfficio **int**;

**SET** validPostazione = checkValidPostazione(postazione);

**IF** **EXISTS** (**SELECT** 1 **FROM** PostazioneAttiva **WHERE** idPostazioneAttiva = postazione) **then**

signal **sqlstate** '45000'

**SET** MESSAGE\_TEXT = 'La postazione risulta già abilitata!';

**END** **IF**;

**SELECT** Ufficio **into** idUfficio **from** Postazione **where** idPostazione = postazione;

**INSERT** **INTO** PostazioneAttiva(Disponibile, idPostazioneAttiva, Ufficio, idPostazione)

**VALUES** (**true**, postazione, idUfficio, postazione);

**END**

**GetAssegnazioniPassate:** viene utilizzata per ottenere le assegnazioni passate relative ad un dipendente

**CREATE** **DEFINER**=`root`@`localhost` **PROCEDURE** `mydb`.`getAssegnazioniPassate`(**IN** idUtente **int**)

**BEGIN**

/\* Controllo l'esistenza del dipendente \*/

**DECLARE** validUtente **int**;

**SET** validUtente = checkUserExists(idUtente);

**SELECT** idImpiegoPassato, DataInizio, DataFine, Postazione **FROM** ImpiegoPassato **WHERE** Dipendente = idUtente;

**END**

**GetPostazioni:** viene utilizzata per mostrare tutte le postazioni, attive e non

**CREATE** **DEFINER**=`root`@`localhost` **PROCEDURE** `mydb`.`getPostazioni`()

**BEGIN**

**SELECT** \* **from** getPostazioni;

**END**

**GetPostazioniAttive:** viene utilizzata per mostrare le postazioni attive

**CREATE** **DEFINER**=`root`@`localhost` **PROCEDURE** `mydb`.`getPostazioniAttive`()

**BEGIN**

**SELECT** \* **from** getPostazioniAttive;

**END**

**GetUserDaTrasferire:** viene utilizzata per mostrare gli utenti attualmente da trasferire

**CREATE** **DEFINER**=`root`@`localhost` **PROCEDURE** `mydb`.`getUserDaTrasferire`()

**BEGIN**

**SELECT** \* **from** getToTransfer;

**END**

**GetUserNumeroInterno:** viene utilizzata per mostrare le informazioni relative all’ufficio ed ai dipendenti attualmente assegnati

**CREATE** **DEFINER**=`root`@`localhost` **PROCEDURE** `mydb`.`getUserNumeroInterno`(**IN** numeroInterno **int**)

**BEGIN**

/\* Devo prima mostrare le informazioni relative all'ufficio \*/

**SELECT** Ufficio.idUfficio, Ufficio.Piano, Ufficio.Edificio, Ufficio.Mansione

**FROM** Ufficio

**WHERE** Ufficio.TelefonoInterno = numeroInterno;

/\* Successivamente devo mostrare le informazioni relative ai dipendenti che stanno dentro

\* l'ufficio

\* \*/

**SELECT** Dipendente.idDipendente, Dipendente.isDaTrasferire

**FROM** impiegoCorrente

**JOIN** PostazioneAttiva **on** impiegoCorrente.Postazione = PostazioneAttiva.idPostazioneAttiva

**JOIN** Ufficio **on** Ufficio.idUfficio = PostazioneAttiva.Ufficio

**JOIN** Dipendente **on** Dipendente.idDipendente = impiegoCorrente.Dipendente

**WHERE** Ufficio.TelefonoInterno = numeroInterno;

**END**

**GetUserScambiabili:** viene utilizzata per ottenere la lista dei dipendenti che sono compatibili con lo scambio di postazione del dipendente indicato

**CREATE** **DEFINER**=`root`@`localhost` **PROCEDURE** `mydb`.`getUserScambiabili`(`newUser` **int**)

**BEGIN**

**DECLARE** `newMansione` **int**;

**DECLARE** `newPostazione` **int**;

/\* Effettuo il controllo sull'esistenza ed il trasferire dell'utente \*/

**DECLARE** `validUser` **int**;

**SET** validUser = checkValidUser(newUser);

**SET** newPostazione = getCurrentPostazione(newUser);

**SET** newMansione = getMansioneUtente(newUser);

**SELECT** I2.dipendente **as** idImpiegato, I2.postazione **as** idPostazione

**from** impiegoCorrente I2

**JOIN** Dipendente **on** Dipendente.idDipendente = I2.Dipendente

**WHERE**

((

**NOT** **EXISTS** (**SELECT** \* **FROM** ImpiegoPassato IP **WHERE** IP.dipendente = newUser **and** IP.postazione = I2.postazione)

**or**

**EXISTS** (**SELECT** \* **from** ImpiegoPassato IP **WHERE** IP.dipendente = newUser **and** IP.postazione = I2.postazione **and** IP.DataInizio < **DATE**(DATE\_sub(NOW(), **INTERVAL** 2 **YEAR**)))

)

**and**

(

**NOT** **EXISTS**(**SELECT** \* **FROM** ImpiegoPassato IP **WHERE** IP.Dipendente = I2.Dipendente **and** IP.postazione = newPostazione)

**or**

**EXISTS**(**SELECT** \* **from** ImpiegoPassato IP **WHERE** IP.Dipendente = I2.Dipendente **and** IP.postazione = newPostazione **and** IP.DataInizio < **DATE**(DATE\_sub(NOW(), **INTERVAL** 2 **YEAR**)))

)

**and** I2.dipendente != newUser **and** Dipendente.Mansione = newMansione

**and** Dipendente.isDaTrasferire = **true**

**and** **EXISTS** (**SELECT** Dipendente **FROM** impiegoCorrente **WHERE** Dipendente = I2.Dipendente)

);

**END**

**InsertNewPostazione:** viene utilzzata per inserire una nuova postazione all’interno della base di dati

**CREATE** **DEFINER**=`root`@`localhost` **PROCEDURE** `mydb`.`insertNewPostazione`(**IN** ufficio **INT**, **OUT** idPostazione **INT**)

**BEGIN**

**DECLARE** validPostazione **INT**;

**INSERT** **INTO** Postazione(Ufficio) **VALUES** (ufficio);

**SET** idPostazione = **LAST\_INSERT\_ID**();

**END**

**LoginUser:** viene utilizzata per effettuare il login all’interno del sistema

**CREATE** **DEFINER**=`root`@`localhost` **PROCEDURE** `mydb`.`loginUser`(**IN** matricolaDipendente **int**, **IN** passwordDipendente **Varchar**(256), **OUT** tipoDipendente **INT**)

**BEGIN**

**SELECT** userType **INTO** tipoDipendente **FROM** Dipendente **WHERE** idDipendente = matricolaDipendente **AND** password = SHA2(passwordDipendente, 256);

**IF** tipoDipendente **is** **NULL** **THEN**

signal **sqlstate** '45000'

**SET** MESSAGE\_TEXT = 'Nome utente e/o password errati!';

**END** **IF**;

**END**

**SetUserDaTrasferire:** viene utilizzata per indicare un dipendente come da trasferire

**CREATE** **DEFINER**=`root`@`localhost` **PROCEDURE** `mydb`.`setUserDaTrasferire`(**IN** utente **INT**)

**BEGIN**

**UPDATE** Dipendente **SET** isDaTrasferire = **TRUE** **Where** idDipendente = utente;

**END**

**UserNameSurname:** viene utilizzata per ottenere le informazioni relative ad un dipendente ed al suo ufficio utilizzando il suo nome e/o cognome

**CREATE** **DEFINER**=`root`@`localhost` **PROCEDURE** `mydb`.`userNameSurname`(name **VARCHAR**(255), surname **VARCHAR**(255))

**BEGIN**

**IF** (name = '' **and** surname = '') **THEN**

signal **sqlstate** '45000'

**SET** MESSAGE\_TEXT = 'Inserire nome e/o cognome del dipendente!';

**ELSE**

**IF** (surname = '') **THEN**

**SELECT** Dipendente.EMailUfficio, Dipendente.EMailPersonale, Ufficio.idUfficio, Ufficio.Piano, Ufficio.Edificio **FROM** Dipendente

**JOIN** impiegoCorrente **ON** impiegoCorrente.Dipendente = Dipendente.idDipendente

**JOIN** PostazioneAttiva **on** impiegoCorrente.Postazione = PostazioneAttiva.idPostazioneAttiva

**JOIN** Ufficio **on** PostazioneAttiva.Ufficio = Ufficio.idUfficio

**WHERE** Dipendente.Nome = name;

**END** **IF**;

**IF** (name = '') **THEN**

**SELECT** Dipendente.EMailUfficio, Dipendente.EMailPersonale, Ufficio.idUfficio, Ufficio.Piano, Ufficio.Edificio **FROM** Dipendente

**JOIN** impiegoCorrente **ON** impiegoCorrente.Dipendente = Dipendente.idDipendente

**JOIN** PostazioneAttiva **on** impiegoCorrente.Postazione = PostazioneAttiva.idPostazioneAttiva

**JOIN** Ufficio **on** PostazioneAttiva.Ufficio = Ufficio.idUfficio

**WHERE** Dipendente.Cognome = surname;

**END** **IF**;

**IF** (name != '' **and** surname != '') **THEN**

**SELECT** Dipendente.EMailUfficio, Dipendente.EMailPersonale, Ufficio.idUfficio, Ufficio.Piano, Ufficio.Edificio **FROM** Dipendente

**JOIN** impiegoCorrente **ON** impiegoCorrente.Dipendente = Dipendente.idDipendente

**JOIN** PostazioneAttiva **on** impiegoCorrente.Postazione = PostazioneAttiva.idPostazioneAttiva

**JOIN** Ufficio **on** PostazioneAttiva.Ufficio = Ufficio.idUfficio

**WHERE** Dipendente.Cognome = surname **AND** Dipendente.Nome = name;

**END** **IF**;

**END** **IF**;

**END**

## Funzioni

**CheckDaTrasferire**

**CREATE** **DEFINER**=`root`@`localhost` **FUNCTION** `mydb`.`checkDaTrasferire`(newDipendente **INT**) **RETURNS** **int**(11)

**DETERMINISTIC**

**BEGIN**

**DECLARE** counter **int**;

**IF** **NOT** **EXISTS** (**SELECT** **COUNT**(\*) **FROM** Dipendente **WHERE** idDipendente = newDipendente **and** daTrasferire **is** **true**) **THEN**

signal **sqlstate** '45000'

**SET** MESSAGE\_TEXT = 'Utente non indicato come da trasferire!';

**END** **IF**;

**RETURN** 1;

**END**

**CheckPostazione**

**CREATE** **DEFINER**=`root`@`localhost` **FUNCTION** `mydb`.`checkPostazione`(ID **int**) **RETURNS** **int**(11)

**DETERMINISTIC**

**BEGIN**

**DECLARE** counter **int**;

**IF** **NOT** **EXISTS** (**SELECT** 1 **FROM** PostazioneAttiva **WHERE** idPostazioneAttiva = ID **AND** Disponibile = **true**) **THEN**

signal **sqlstate** '45000'

**SET** MESSAGE\_TEXT = 'Postazione non esistente o attualmente già occupata!';

**END** **IF**;

**RETURN** 1;

**END**

**CheckUserExists**

**CREATE** **DEFINER**=`root`@`localhost` **FUNCTION** `mydb`.`checkUserExists`(userID **int**) **RETURNS** **int**(11)

**DETERMINISTIC**

**BEGIN**

**DECLARE** counter **int**;

**IF** **NOT** **EXISTS** (**SELECT** 1 **FROM** Dipendente **WHERE** idDipendente=userID) **then**

signal **sqlstate** '45000'

**SET** MESSAGE\_TEXT = 'Il dipendente indicato non esiste!';

**END** **IF**;

**RETURN** counter;

**END**

**CheckValidMansione**

**CREATE** **DEFINER**=`root`@`localhost` **FUNCTION** `mydb`.`checkValidMansione`(ID **int**) **RETURNS** **int**(11)

**DETERMINISTIC**

**BEGIN**

**DECLARE** counter **int**;

**IF** **NOT** **EXISTS** (**SELECT** 1 **FROM** Mansione **WHERE** idMansione = ID) **THEN**

signal **sqlstate** '45000'

**SET** MESSAGE\_TEXT = 'Mansione non valida!';

**END** **IF**;

**RETURN** 1;

**END**

**CheckValidPostazione**

**CREATE** **DEFINER**=`root`@`localhost` **FUNCTION** `mydb`.`checkValidPostazione`(postazione **int**) **RETURNS** **int**(11)

**DETERMINISTIC**

**BEGIN**

**IF** **NOT** **EXISTS** (**SELECT** 1 **FROM** Postazione **WHERE** idPostazione = postazione) **then**

signal **sqlstate** '45000'

**SET** MESSAGE\_TEXT = 'Nessuna postazione associata all\'ID indicato!';

**END** **IF**;

**RETURN** 1;

**END**

**CheckValidUser**

**CREATE** **DEFINER**=`root`@`localhost` **FUNCTION** `mydb`.`checkValidUser`(userID **int**) **RETURNS** **int**(11)

**DETERMINISTIC**

**BEGIN**

**DECLARE** counter **int**;

**IF** **NOT** **EXISTS** (**SELECT** 1 **FROM** Dipendente **WHERE** idDipendente=userID **and** isDaTrasferire **is** **true**) **then**

signal **sqlstate** '45000'

**SET** MESSAGE\_TEXT = 'Dipendente non esiste e/o non attualmente da trasferire!';

**END** **IF**;

**RETURN** 1;

**END**

**GetCurrentPostazione**

**CREATE** **DEFINER**=`root`@`localhost` **FUNCTION** `mydb`.`getCurrentPostazione`(idDipendente **INT**) **RETURNS** **int**(11)

**DETERMINISTIC**

**BEGIN**

**DECLARE** idPostazione **int**;

**SELECT** Postazione **INTO** idPostazione **from** impiegoCorrente **where** Dipendente = idDipendente **LIMIT** 1;

**RETURN** idPostazione;

**END**

**GetMansioneUtente**

**CREATE** **DEFINER**=`root`@`localhost` **FUNCTION** `mydb`.`getMansioneUtente`(dipendenteNum **Int**) **RETURNS** **int**(11)

**DETERMINISTIC**

**BEGIN**

**DECLARE** idMansione **int**;

**SELECT** Mansione **INTO** idMansione **FROM** Dipendente **WHERE** idDipendente = dipendenteNum **Limit** 1;

**IF** idMansione **is** **null** **then**

signal **sqlstate** '45000'

**SET** MESSAGE\_TEXT = 'Nessuna mansione associata al dipendente!';

**END** **IF**;

**RETURN** idMansione;

**END**

# Appendice: Implementazione

## Codice SQL per instanziare il database

Riportare il codice SQL necessario ad istanziare lo schema del DB.

## /\*!40101 SET @saved\_cs\_client = @@character\_set\_client \*/ ;

/\*!40101 SET character\_set\_client = utf8 \*/

;

CREATE TABLE `Dipendente`(

`idDipendente` int(11) NOT NULL AUTO\_INCREMENT,

`Mansione` int(11) NOT NULL DEFAULT 0,

`Nome` varchar(45) NOT NULL,

`IndirizoResidenza` varchar(45) NOT NULL,

`EMailUfficio` varchar(45) NOT NULL DEFAULT 'dummyEmail@office.com',

`EMailPersonale` varchar(45) NOT NULL,

`DataNascita` date NOT NULL,

`LuogoNascita` varchar(45) NOT NULL,

`password` varchar(256) NOT NULL,

`userType` int(11) NOT NULL,

`Cognome` varchar(100) NOT NULL,

`isDaTrasferire` tinyint(1) NOT NULL,

PRIMARY KEY(`idDipendente`),

KEY `fk\_Dipendente\_1`(`Mansione`),

CONSTRAINT `fk\_Dipendente\_1`

FOREIGN KEY(`Mansione`) REFERENCES `Mansione`(`idMansione`) ON DELETE NO ACTION ON UPDATE NO ACTION

) ENGINE = InnoDB AUTO\_INCREMENT = 4 DEFAULT CHARSET = utf8;

/\*!40101 SET character\_set\_client = @saved\_cs\_client \*/

;

INSERT INTO `Dipendente`

VALUES(1, 2, 'Giuseppe', 'Via Argada, 2', 'ufficio2@test.com', 'giuseppe@gmail.com', '1995-04-10', 'Catanzaro', '688787d8ff144c502c7f5cffaafe2cc588d86079f9de88304c26b0cb99ce91c6', 1, 'A', 0), (2, 2, 'Francesco', 'Via Argada, 3', 'ufficio1@test.com', 'francesco@gmail.com', '1996-04-11', 'Girifalco', '1b1d74f95b17f8241afa453ad24ed005101a4c12e699ad6ba83a3d5d28627914', 2, 'B', 0), (3, 2, 'Salvatore', 'Via Argada, 4', 'ufficio2@test.com', 'salvatore@gmail.com', '1995-04-12', 'Catanzaro', 'bbh', 3, 'C', 0);

/\*!50003 SET @saved\_cs\_client = @@character\_set\_client \*/;

/\*!50003 SET @saved\_cs\_results = @@character\_set\_results \*/;

/\*!50003 SET @saved\_col\_connection = @@collation\_connection \*/;

/\*!50003 SET character\_set\_client = utf8 \*/;

/\*!50003 SET character\_set\_results = utf8 \*/;

/\*!50003 SET collation\_connection = utf8\_general\_ci \*/;

/\*!50003 SET @saved\_sql\_mode = @@sql\_mode \*/;

/\*!50003 SET sql\_mode = 'IGNORE\_SPACE,STRICT\_TRANS\_TABLES,ERROR\_FOR\_DIVISION\_BY\_ZERO,NO\_AUTO\_CREATE\_USER,NO\_ENGINE\_SUBSTITUTION' \*/;

DELIMITER;;

/\*!50003 CREATE\*/

/\*!50017 DEFINER=`root`@`localhost`\*/

/\*!50003 TRIGGER `mydb`.`Dipendente\_BEFORE\_UPDATE` BEFORE UPDATE ON `Dipendente` FOR EACH ROW

BEGIN

IF (NEW.Mansione != OLD.Mansione) THEN

SET NEW.isDaTrasferire = TRUE;

END IF;

END \*/;;

DELIMITER;

/\*!50003 SET sql\_mode = @saved\_sql\_mode \*/;

/\*!50003 SET character\_set\_client = @saved\_cs\_client \*/;

/\*!50003 SET character\_set\_results = @saved\_cs\_results \*/;

/\*!50003 SET collation\_connection = @saved\_col\_connection \*/;

/\*!40101 SET @saved\_cs\_client = @@character\_set\_client \*/;

/\*!40101 SET character\_set\_client = utf8 \*/;

CREATE TABLE `ImpiegoPassato`(

`idImpiegoPassato`

int(11) NOT NULL AUTO\_INCREMENT,

`DataInizio`

date NOT NULL,

`DataFine`

date NOT NULL,

`Postazione`

int(11) NOT NULL,

`Dipendente`

int(11) NOT NULL,

PRIMARY KEY(`idImpiegoPassato`, `Postazione`, `Dipendente`),

KEY `fk\_ImpiegoPassato\_1`(`Postazione`),

KEY `fk\_ImpiegoPassato\_2`(`Dipendente`),

CONSTRAINT `fk\_ImpiegoPassato\_1`

FOREIGN KEY(`Postazione`) REFERENCES `Postazione`(`idPostazione`) ON DELETE NO ACTION ON UPDATE NO ACTION,

CONSTRAINT `fk\_ImpiegoPassato\_2`

FOREIGN KEY(`Dipendente`) REFERENCES `Dipendente`(`idDipendente`) ON DELETE NO ACTION ON UPDATE NO ACTION

) ENGINE = InnoDB AUTO\_INCREMENT = 23 DEFAULT CHARSET = utf8;

/\*!40101 SET character\_set\_client = @saved\_cs\_client \*/

;

INSERT INTO `ImpiegoPassato`

VALUES(19, '2019-05-03', '2019-05-03', 2, 3), (20, '2019-05-03', '2019-05-03', 1, 2), (21, '2019-05-03', '2019-05-13', 5, 1), (22, '2019-05-13', '2019-05-17', 1, 1);

/\*!40101 SET @saved\_cs\_client = @@character\_set\_client \*/;

/\*!40101 SET character\_set\_client = utf8 \*/;

CREATE TABLE `Mansione`(

`idMansione` int(11) NOT NULL AUTO\_INCREMENT,

`NomeMansione` varchar(45) NOT NULL,

PRIMARY KEY(`idMansione`)

) ENGINE = InnoDB AUTO\_INCREMENT = 3 DEFAULT CHARSET = utf8;

/\*!40101 SET character\_set\_client = @saved\_cs\_client \*/

;

INSERT INTO `Mansione`

VALUES(1, 'Amministrazione'), (2, 'MansioneProva');

/\*!40101 SET @saved\_cs\_client = @@character\_set\_client \*/;

/\*!40101 SET character\_set\_client = utf8 \*/;

CREATE TABLE `Postazione`(

`idPostazione`

int(11) NOT NULL AUTO\_INCREMENT,

`Ufficio`

int(11) NOT NULL,

PRIMARY KEY(`idPostazione`, `Ufficio`),

KEY `fk\_Postazione\_1`(`Ufficio`),

CONSTRAINT `fk\_Postazione\_1`

FOREIGN KEY(`Ufficio`) REFERENCES `Ufficio`(`idUfficio`) ON DELETE NO ACTION ON UPDATE NO ACTION

) ENGINE = InnoDB AUTO\_INCREMENT = 9 DEFAULT CHARSET = utf8;

/\*!40101 SET character\_set\_client = @saved\_cs\_client \*/;

INSERT INTO `Postazione`

VALUES(1, 1), (2, 2), (3, 1), (4, 2), (5, 2), (6, 1), (7, 1);

/\*!50003 SET @saved\_cs\_client = @@character\_set\_client \*/;

/\*!50003 SET @saved\_cs\_results = @@character\_set\_results \*/;

/\*!50003 SET @saved\_col\_connection = @@collation\_connection \*/;

/\*!50003 SET character\_set\_client = utf8 \*/;

/\*!50003 SET character\_set\_results = utf8 \*/;

/\*!50003 SET collation\_connection = utf8\_general\_ci \*/;

/\*!50003 SET @saved\_sql\_mode = @@sql\_mode \*/;

/\*!50003 SET sql\_mode = 'IGNORE\_SPACE,STRICT\_TRANS\_TABLES,ERROR\_FOR\_DIVISION\_BY\_ZERO,NO\_AUTO\_CREATE\_USER,NO\_ENGINE\_SUBSTITUTION' \*/;

DELIMITER;;

/\*!50003 CREATE\*/

/\*!50017 DEFINER=`root`@`localhost`\*/

/\*!50003 TRIGGER `mydb`.`Postazione\_BEFORE\_UPDATE` BEFORE UPDATE ON `Postazione` FOR EACH ROW

BEGIN

IF (NEW.Ufficio != OLD.Ufficio) THEN

IF EXISTS (SELECT 1 FROM PostazioneAttiva WHERE idPostazioneAttiva = OLD.idPostazione and Disponibile is false) THEN

signal sqlstate '45000'

SET MESSAGE\_TEXT = 'Non è assegnare spostare una postazione attualmente occupata!';

END IF;

IF NOT EXISTS (SELECT 1 FROM Ufficio WHERE idUfficio = NEW.Ufficio) THEN

signal sqlstate '45000'

SET MESSAGE\_TEXT = 'L\'ID dell\'ufficio indicato non è corretto!';

END IF;

END IF;

END \*/

;;

DELIMITER;

/\*!50003 SET sql\_mode = @saved\_sql\_mode \*/;

/\*!50003 SET character\_set\_client = @saved\_cs\_client \*/;

/\*!50003 SET character\_set\_results = @saved\_cs\_results \*/;

/\*!50003 SET collation\_connection = @saved\_col\_connection \*/;

/\*!40101 SET @saved\_cs\_client = @@character\_set\_client \*/;

/\*!40101 SET character\_set\_client = utf8 \*/;

CREATE TABLE `PostazioneAttiva`(

`idPostazioneAttiva`

int(11) NOT NULL AUTO\_INCREMENT COMMENT ' ',

`Disponibile`

tinyint(4) NOT NULL,

`Ufficio`

int(11) NOT NULL,

`idPostazione`

int(11) NOT NULL,

PRIMARY KEY(`idPostazioneAttiva`, `Ufficio`, `idPostazione`),

KEY `fk\_table1\_1`(`Ufficio`),

KEY `PostazioneAttiva\_Postazione\_FK`(`idPostazione`, `Ufficio`),

CONSTRAINT `fk\_table1\_1`

FOREIGN KEY(`Ufficio`) REFERENCES `Ufficio`(`idUfficio`) ON DELETE NO ACTION ON UPDATE NO ACTION

) ENGINE = InnoDB AUTO\_INCREMENT = 9 DEFAULT CHARSET = utf8;

/\*!40101 SET character\_set\_client = @saved\_cs\_client \*/;

INSERT INTO `PostazioneAttiva`

VALUES(1, 1, 1, 1), (2, 0, 2, 2), (3, 0, 1, 3), (4, 0, 2, 4), (5, 1, 2, 5);

/\*!50003 SET @saved\_cs\_client = @@character\_set\_client \*/;

/\*!50003 SET @saved\_cs\_results = @@character\_set\_results \*/;

/\*!50003 SET @saved\_col\_connection = @@collation\_connection \*/;

/\*!50003 SET character\_set\_client = utf8 \*/;

/\*!50003 SET character\_set\_results = utf8 \*/;

/\*!50003 SET collation\_connection = utf8\_general\_ci \*/;

/\*!50003 SET @saved\_sql\_mode = @@sql\_mode \*/;

/\*!50003 SET sql\_mode = 'IGNORE\_SPACE,STRICT\_TRANS\_TABLES,ERROR\_FOR\_DIVISION\_BY\_ZERO,NO\_AUTO\_CREATE\_USER,NO\_ENGINE\_SUBSTITUTION' \*/;

DELIMITER;;

/\*!50003 CREATE\*/

/\*!50017 DEFINER=`root`@`localhost`\*/

/\*!50003 TRIGGER `mydb`.`PostazioneAttiva\_BEFORE\_INSERT` BEFORE INSERT ON `PostazioneAttiva` FOR EACH ROW

BEGIN

IF NOT EXISTS (SELECT 1 FROM Postazione WHERE idPostazione = NEW.idPostazioneAttiva and Ufficio != 0) THEN

signal sqlstate '45000'

SET MESSAGE\_TEXT = 'Nessuna postazione associata all\'ID indicato!';

END IF;

END \*/;;

DELIMITER;

/\*!50003 SET sql\_mode = @saved\_sql\_mode \*/;

/\*!50003 SET character\_set\_client = @saved\_cs\_client \*/;

/\*!50003 SET character\_set\_results = @saved\_cs\_results \*/;

/\*!50003 SET collation\_connection = @saved\_col\_connection \*/;

/\*!50003 SET @saved\_cs\_client = @@character\_set\_client \*/;

/\*!50003 SET @saved\_cs\_results = @@character\_set\_results \*/;

/\*!50003 SET @saved\_col\_connection = @@collation\_connection \*/;

/\*!50003 SET character\_set\_client = utf8 \*/;

/\*!50003 SET character\_set\_results = utf8 \*/;

/\*!50003 SET collation\_connection = utf8\_general\_ci \*/;

/\*!50003 SET @saved\_sql\_mode = @@sql\_mode \*/;

/\*!50003 SET sql\_mode = 'IGNORE\_SPACE,STRICT\_TRANS\_TABLES,ERROR\_FOR\_DIVISION\_BY\_ZERO,NO\_AUTO\_CREATE\_USER,NO\_ENGINE\_SUBSTITUTION' \*/;

DELIMITER;;

/\*!50003 CREATE\*/

/\*!50017 DEFINER=`root`@`localhost`\*/

/\*!50003 TRIGGER `mydb`.`PostazioneAttiva\_AFTER\_INSERT` AFTER INSERT ON `PostazioneAttiva` FOR EACH ROW

BEGIN

IF NEW.Disponibile is true THEN

UPDATE Ufficio set NumeroPostazioniLibere = NumeroPostazioniLibere + 1 WHERE idUfficio = NEW.Ufficio;

END IF;

END \*/

;;

DELIMITER;

/\*!50003 SET sql\_mode = @saved\_sql\_mode \*/

;

/\*!50003 SET character\_set\_client = @saved\_cs\_client \*/

;

/\*!50003 SET character\_set\_results = @saved\_cs\_results \*/

;

/\*!50003 SET collation\_connection = @saved\_col\_connection \*/

;

/\*!50003 SET @saved\_cs\_client = @@character\_set\_client \*/

;

/\*!50003 SET @saved\_cs\_results = @@character\_set\_results \*/

;

/\*!50003 SET @saved\_col\_connection = @@collation\_connection \*/

;

/\*!50003 SET character\_set\_client = utf8 \*/

;

/\*!50003 SET character\_set\_results = utf8 \*/

;

/\*!50003 SET collation\_connection = utf8\_general\_ci \*/

;

/\*!50003 SET @saved\_sql\_mode = @@sql\_mode \*/

;

/\*!50003 SET sql\_mode = 'IGNORE\_SPACE,STRICT\_TRANS\_TABLES,ERROR\_FOR\_DIVISION\_BY\_ZERO,NO\_AUTO\_CREATE\_USER,NO\_ENGINE\_SUBSTITUTION' \*/

;

DELIMITER;;

/\*!50003 CREATE\*/

/\*!50017 DEFINER=`root`@`localhost`\*/

/\*!50003 TRIGGER `mydb`.`PostazioneAttiva\_AFTER\_UPDATE\_1` AFTER UPDATE ON `PostazioneAttiva` FOR EACH ROW

BEGIN

IF (OLD.Disponibile != NEW.Disponibile and NEW.Disponibile = false) then

UPDATE Ufficio set NumeroPostazioniLibere = NumeroPostazioniLibere - 1 WHERE idUfficio = NEW.Ufficio;

END IF;

IF (OLD.Disponibile != NEW.Disponibile and NEW.Disponibile = true) then

UPDATE Ufficio set NumeroPostazioniLibere = numeroPostazioniLibere + 1 Where idUfficio = NEW.Ufficio;

END IF;

END \*/

;;

DELIMITER;

/\*!50003 SET sql\_mode = @saved\_sql\_mode \*/

;

/\*!50003 SET character\_set\_client = @saved\_cs\_client \*/

;

/\*!50003 SET character\_set\_results = @saved\_cs\_results \*/

;

/\*!50003 SET collation\_connection = @saved\_col\_connection \*/

;

/\*!50003 SET @saved\_cs\_client = @@character\_set\_client \*/

;

/\*!50003 SET @saved\_cs\_results = @@character\_set\_results \*/

;

/\*!50003 SET @saved\_col\_connection = @@collation\_connection \*/

;

/\*!50003 SET character\_set\_client = utf8 \*/

;

/\*!50003 SET character\_set\_results = utf8 \*/

;

/\*!50003 SET collation\_connection = utf8\_general\_ci \*/

;

/\*!50003 SET @saved\_sql\_mode = @@sql\_mode \*/

;

/\*!50003 SET sql\_mode = 'IGNORE\_SPACE,STRICT\_TRANS\_TABLES,ERROR\_FOR\_DIVISION\_BY\_ZERO,NO\_AUTO\_CREATE\_USER,NO\_ENGINE\_SUBSTITUTION' \*/

;

DELIMITER;;

/\*!50003 CREATE\*/

/\*!50017 DEFINER=`root`@`localhost`\*/

/\*!50003 TRIGGER `mydb`.`PostazioneAttiva\_BEFORE\_DELETE` BEFORE Delete ON `PostazioneAttiva` FOR EACH ROW

BEGIN

IF (OLD.Disponibile is false) THEN

signal sqlstate '45000'

SET MESSAGE\_TEXT = 'Non è possibile disabilitare una postazione attualmente assegnata ad un dipendente!';

END IF;

UPDATE Ufficio SET NumeroPostazioniLibere = NumeroPostazioniLibere - 1 WHERE idUfficio = OLD.Ufficio;

END \*/

;;

DELIMITER;

/\*!50003 SET sql\_mode = @saved\_sql\_mode \*/

;

/\*!50003 SET character\_set\_client = @saved\_cs\_client \*/

;

/\*!50003 SET character\_set\_results = @saved\_cs\_results \*/

;

/\*!50003 SET collation\_connection = @saved\_col\_connection \*/

;

/\*!40101 SET @saved\_cs\_client = @@character\_set\_client \*/

;

/\*!40101 SET character\_set\_client = utf8 \*/

;

CREATE TABLE `Ufficio`(

`idUfficio`

int(11) NOT NULL AUTO\_INCREMENT,

`Piano`

int(11) NOT NULL,

`Edificio`

varchar(10) NOT NULL,

`TelefonoInterno`

varchar(45) NOT NULL,

`TelefonoEsterno`

varchar(45) NOT NULL,

`NumeroPostazioniLibere`

int(11) NOT NULL,

`Mansione`

int(11) NOT NULL,

`Email`

varchar(100) NOT NULL,

PRIMARY KEY(`idUfficio`, `Mansione`),

KEY `fk\_Ufficio\_1`(`Mansione`),

CONSTRAINT `fk\_Ufficio\_1`

FOREIGN KEY(`Mansione`) REFERENCES `Mansione`(`idMansione`) ON DELETE NO ACTION ON UPDATE NO ACTION

) ENGINE = InnoDB AUTO\_INCREMENT = 3 DEFAULT CHARSET = utf8;

/\*!40101 SET character\_set\_client = @saved\_cs\_client \*/

;

INSERT INTO `Ufficio`

VALUES(1, 1, '1', '000001', '010101', 1, 1, 'ufficio1@test.com'), (2, 2, '1', '000002', '020202', 2, 2, 'ufficio2@test.com');

SET @saved\_cs\_client = @ @character\_set\_client;

SET character\_set\_client = utf8;

/\*!50001 CREATE TABLE `getPostazioni` (

`idPostazione` tinyint NOT NULL,

`Ufficio` tinyint NOT NULL

) ENGINE=MyISAM \*/

;

SET character\_set\_client = @saved\_cs\_client;

SET @saved\_cs\_client = @ @character\_set\_client;

SET character\_set\_client = utf8;

/\*!50001 CREATE TABLE `getPostazioniAttive` (

`idPostazioneAttiva` tinyint NOT NULL,

`Disponibile` tinyint NOT NULL,

`Ufficio` tinyint NOT NULL

) ENGINE=MyISAM \*/

;

SET character\_set\_client = @saved\_cs\_client;

SET @saved\_cs\_client = @ @character\_set\_client;

SET character\_set\_client = utf8;

/\*!50001 CREATE TABLE `getToTransfer` (

`idDipendente` tinyint NOT NULL

) ENGINE=MyISAM \*/

;

SET character\_set\_client = @saved\_cs\_client;

/\*!40101 SET @saved\_cs\_client = @@character\_set\_client \*/

;

/\*!40101 SET character\_set\_client = utf8 \*/

;

CREATE TABLE `impiegoCorrente`(

`idImpiegoCorrente`

int(11) NOT NULL AUTO\_INCREMENT,

`DataInizio`

date NOT NULL,

`Postazione`

int(11) NOT NULL,

`Dipendente`

int(11) NOT NULL,

`dummyValue`

tinyint(4) NOT NULL DEFAULT 1,

PRIMARY KEY(`idImpiegoCorrente`, `Postazione`, `Dipendente`),

KEY `fk\_ImpiegoCorrente\_1`(`Postazione`),

KEY `fk\_ImpiegoCorrente\_2`(`Dipendente`),

CONSTRAINT `fk\_ImpiegoCorrente\_1`

FOREIGN KEY(`Postazione`) REFERENCES `PostazioneAttiva`(`idPostazioneAttiva`) ON DELETE NO ACTION ON UPDATE NO ACTION,

CONSTRAINT `fk\_ImpiegoCorrente\_2`

FOREIGN KEY(`Dipendente`) REFERENCES `Dipendente`(`idDipendente`) ON DELETE NO ACTION ON UPDATE NO ACTION

) ENGINE = InnoDB AUTO\_INCREMENT = 6 DEFAULT CHARSET = utf8;

/\*!40101 SET character\_set\_client = @saved\_cs\_client \*/

;

INSERT INTO `impiegoCorrente`

VALUES(1, '2019-05-17', 2, 1, 1), (2, '2019-05-03', 3, 2, 1), (5, '2019-05-03', 4, 3, 1);

/\*!50003 SET @saved\_cs\_client = @@character\_set\_client \*/

;

/\*!50003 SET @saved\_cs\_results = @@character\_set\_results \*/

;

/\*!50003 SET @saved\_col\_connection = @@collation\_connection \*/

;

/\*!50003 SET character\_set\_client = utf8 \*/

;

/\*!50003 SET character\_set\_results = utf8 \*/

;

/\*!50003 SET collation\_connection = utf8\_general\_ci \*/

;

/\*!50003 SET @saved\_sql\_mode = @@sql\_mode \*/

;

/\*!50003 SET sql\_mode = 'IGNORE\_SPACE,STRICT\_TRANS\_TABLES,ERROR\_FOR\_DIVISION\_BY\_ZERO,NO\_AUTO\_CREATE\_USER,NO\_ENGINE\_SUBSTITUTION' \*/

;

DELIMITER;;

/\*!50003 CREATE\*/

/\*!50017 DEFINER=`root`@`localhost`\*/

/\*!50003 TRIGGER `mydb`.`impiegoCorrente\_BEFORE\_INSERT` BEFORE INSERT ON `impiegoCorrente` FOR EACH ROW

BEGIN

IF NOT EXISTS (SELECT 1 FROM Dipendente WHERE idDipendente = NEW.Dipendente AND isDaTrasferire is true) THEN

signal sqlstate '45000'

SET MESSAGE\_TEXT = 'L\'ID del dipendente non è valido o non disponibile per il trasferimento!';

END IF;

END \*/

;;

DELIMITER;

/\*!50003 SET sql\_mode = @saved\_sql\_mode \*/

;

/\*!50003 SET character\_set\_client = @saved\_cs\_client \*/

;

/\*!50003 SET character\_set\_results = @saved\_cs\_results \*/

;

/\*!50003 SET collation\_connection = @saved\_col\_connection \*/

;

/\*!50003 SET @saved\_cs\_client = @@character\_set\_client \*/

;

/\*!50003 SET @saved\_cs\_results = @@character\_set\_results \*/

;

/\*!50003 SET @saved\_col\_connection = @@collation\_connection \*/

;

/\*!50003 SET character\_set\_client = utf8 \*/

;

/\*!50003 SET character\_set\_results = utf8 \*/

;

/\*!50003 SET collation\_connection = utf8\_general\_ci \*/

;

/\*!50003 SET @saved\_sql\_mode = @@sql\_mode \*/

;

/\*!50003 SET sql\_mode = 'IGNORE\_SPACE,STRICT\_TRANS\_TABLES,ERROR\_FOR\_DIVISION\_BY\_ZERO,NO\_AUTO\_CREATE\_USER,NO\_ENGINE\_SUBSTITUTION' \*/

;

DELIMITER;;

/\*!50003 CREATE\*/

/\*!50017 DEFINER=`root`@`localhost`\*/

/\*!50003 TRIGGER `mydb`.`impiegoCorrente\_AFTER\_INSERT` AFTER INSERT ON `impiegoCorrente` FOR EACH ROW

BEGIN

UPDATE PostazioneAttiva set Disponibile = false WHERE idPostazioneAttiva = NEW.Postazione;

UPDATE Dipendente set eMailUfficio = (SELECT ufficio.Email FROM PostazioneAttiva JOIN Ufficio as ufficio on PostazioneAttiva.Ufficio = ufficio.idUfficio WHERE PostazioneAttiva.idPostazioneAttiva = NEW.Postazione), isDaTrasferire = FALSE WHERE idDipendente = NEW.Dipendente;

END \*/

;;

DELIMITER;

/\*!50003 SET sql\_mode = @saved\_sql\_mode \*/

;

/\*!50003 SET character\_set\_client = @saved\_cs\_client \*/

;

/\*!50003 SET character\_set\_results = @saved\_cs\_results \*/

;

/\*!50003 SET collation\_connection = @saved\_col\_connection \*/

;

/\*!50003 SET @saved\_cs\_client = @@character\_set\_client \*/

;

/\*!50003 SET @saved\_cs\_results = @@character\_set\_results \*/

;

/\*!50003 SET @saved\_col\_connection = @@collation\_connection \*/

;

/\*!50003 SET character\_set\_client = utf8 \*/

;

/\*!50003 SET character\_set\_results = utf8 \*/

;

/\*!50003 SET collation\_connection = utf8\_general\_ci \*/

;

/\*!50003 SET @saved\_sql\_mode = @@sql\_mode \*/

;

/\*!50003 SET sql\_mode = 'IGNORE\_SPACE,STRICT\_TRANS\_TABLES,ERROR\_FOR\_DIVISION\_BY\_ZERO,NO\_AUTO\_CREATE\_USER,NO\_ENGINE\_SUBSTITUTION' \*/

;

DELIMITER;;

/\*!50003 CREATE\*/

/\*!50017 DEFINER=`root`@`localhost`\*/

/\*!50003 TRIGGER `mydb`.`impiegoCorrente\_BEFORE\_UPDATE` BEFORE UPDATE ON `impiegoCorrente` FOR EACH ROW

BEGIN

DECLARE mansioneDipendente int;

SET mansioneDipendente = getMansioneUtente(NEW.Dipendente);

IF (NEW.Postazione != OLD.Postazione AND OLD.dummyValue = 0) THEN

IF NOT EXISTS(SELECT idPostazioneAttiva FROM PostazioneAttiva

JOIN Ufficio on PostazioneAttiva.Ufficio = Ufficio.idUfficio

JOIN Mansione on Ufficio.Mansione = mansioneDipendente

WHERE (

NOT EXISTS (SELECT \* FROM ImpiegoPassato IP WHERE IP.dipendente = NEW.Dipendente and IP.postazione = NEW.Postazione)

or

EXISTS (SELECT \* from ImpiegoPassato IP WHERE IP.dipendente = NEW.Dipendente and IP.postazione = NEW.Postazione and IP.DataInizio < DATE(DATE\_sub(NOW(), INTERVAL 2 YEAR)))

)) THEN

signal sqlstate '45000'

SET MESSAGE\_TEXT = 'Il dipendente è stato già assegnato a questa postazione nei precedenti due anni!';

END IF;

SET NEW.DataInizio = NOW();

INSERT INTO ImpiegoPassato(DataInizio, DataFine, Postazione, Dipendente) VALUES (OLD.dataInizio, NOW(), OLD.Postazione, OLD.Dipendente);

UPDATE Dipendente set eMailUfficio = (SELECT ufficio.Email FROM PostazioneAttiva JOIN Ufficio as ufficio on PostazioneAttiva.Ufficio = ufficio.idUfficio WHERE PostazioneAttiva.idPostazioneAttiva = NEW.Postazione), isDaTrasferire = FALSE WHERE idDipendente = NEW.Dipendente;

SET NEW.dummyValue = 1;

END IF;

IF (NEW.Postazione != OLD.Postazione AND OLD.dummyValue != 0) THEN

IF NOT EXISTS(SELECT idPostazioneAttiva FROM PostazioneAttiva

JOIN Ufficio on PostazioneAttiva.Ufficio = Ufficio.idUfficio

JOIN Mansione on Ufficio.Mansione = mansioneDipendente

WHERE (

NOT EXISTS (SELECT \* FROM ImpiegoPassato IP WHERE IP.dipendente = NEW.Dipendente and IP.postazione = NEW.Postazione)

or

EXISTS (SELECT \* from ImpiegoPassato IP WHERE IP.dipendente = NEW.Dipendente and IP.postazione = NEW.Postazione and IP.DataInizio < DATE(DATE\_sub(NOW(), INTERVAL 2 YEAR)))

)

AND Disponibile is TRUE) THEN

signal sqlstate '45000'

SET MESSAGE\_TEXT = 'Il dipendente è stato già assegnato a questa postazione nei precedenti due anni!';

END IF;

SET NEW.DataInizio = NOW();

INSERT INTO ImpiegoPassato(DataInizio, DataFine, Postazione, Dipendente) VALUES (OLD.dataInizio, NOW(), OLD.Postazione, OLD.Dipendente);

UPDATE PostazioneAttiva set Disponibile = false WHERE idPostazioneAttiva = NEW.Postazione;

UPDATE PostazioneAttiva set Disponibile = true WHERE idPostazioneAttiva = OLD.Postazione;

UPDATE Dipendente set eMailUfficio = (SELECT ufficio.Email FROM PostazioneAttiva JOIN Ufficio as ufficio on PostazioneAttiva.Ufficio = ufficio.idUfficio WHERE PostazioneAttiva.idPostazioneAttiva = NEW.Postazione), isDaTrasferire = FALSE WHERE idDipendente = NEW.Dipendente;

END IF;

IF (NEW.Dipendente != OLD.Dipendente) THEN

IF NOT EXISTS (SELECT 1 FROM Dipendente WHERE idDipendente = NEW.Dipendente and isDaTrasferire is true) THEN

signal sqlstate '45000'

SET MESSAGE\_TEXT = 'L\'ID del dipendente non è valido o non disponibile per il trasferimento!';

END IF;

END IF;

END \*/

;;

DELIMITER;

/\*!50003 SET sql\_mode = @saved\_sql\_mode \*/

;

/\*!50003 SET character\_set\_client = @saved\_cs\_client \*/

;

/\*!50003 SET character\_set\_results = @saved\_cs\_results \*/

;

/\*!50003 SET collation\_connection = @saved\_col\_connection \*/

;

/\*!50001 DROP TABLE IF EXISTS `getPostazioni`\*/

;

/\*!50001 SET @saved\_cs\_client = @@character\_set\_client \*/

;

/\*!50001 SET @saved\_cs\_results = @@character\_set\_results \*/

;

/\*!50001 SET @saved\_col\_connection = @@collation\_connection \*/

;

/\*!50001 SET character\_set\_client = utf8 \*/

;

/\*!50001 SET character\_set\_results = utf8 \*/

;

/\*!50001 SET collation\_connection = utf8\_general\_ci \*/

;

/\*!50001 CREATE ALGORITHM=UNDEFINED \*/

/\*!50013 DEFINER=`root`@`localhost` SQL SECURITY DEFINER \*/

/\*!50001 VIEW `getPostazioni` AS select `Postazione`.`idPostazione` AS `idPostazione`,`Postazione`.`Ufficio` AS `Ufficio` from `Postazione` \*/

;

/\*!50001 SET character\_set\_client = @saved\_cs\_client \*/

;

/\*!50001 SET character\_set\_results = @saved\_cs\_results \*/

;

/\*!50001 SET collation\_connection = @saved\_col\_connection \*/

;

/\*!50001 DROP TABLE IF EXISTS `getPostazioniAttive`\*/

;

/\*!50001 SET @saved\_cs\_client = @@character\_set\_client \*/

;

/\*!50001 SET @saved\_cs\_results = @@character\_set\_results \*/

;

/\*!50001 SET @saved\_col\_connection = @@collation\_connection \*/

;

/\*!50001 SET character\_set\_client = utf8 \*/

;

/\*!50001 SET character\_set\_results = utf8 \*/

;

/\*!50001 SET collation\_connection = utf8\_general\_ci \*/

;

/\*!50001 CREATE ALGORITHM=UNDEFINED \*/

/\*!50013 DEFINER=`root`@`localhost` SQL SECURITY DEFINER \*/

/\*!50001 VIEW `getPostazioniAttive` AS select `PostazioneAttiva`.`idPostazioneAttiva` AS `idPostazioneAttiva`,`PostazioneAttiva`.`Disponibile` AS `Disponibile`,`PostazioneAttiva`.`Ufficio` AS `Ufficio` from `PostazioneAttiva` \*/

;

/\*!50001 SET character\_set\_client = @saved\_cs\_client \*/

;

/\*!50001 SET character\_set\_results = @saved\_cs\_results \*/

;

/\*!50001 SET collation\_connection = @saved\_col\_connection \*/

;

/\*!50001 DROP TABLE IF EXISTS `getToTransfer`\*/

;

/\*!50001 SET @saved\_cs\_client = @@character\_set\_client \*/

;

/\*!50001 SET @saved\_cs\_results = @@character\_set\_results \*/

;

/\*!50001 SET @saved\_col\_connection = @@collation\_connection \*/

;

/\*!50001 SET character\_set\_client = utf8 \*/

;

/\*!50001 SET character\_set\_results = utf8 \*/

;

/\*!50001 SET collation\_connection = utf8\_general\_ci \*/

;

/\*!50001 CREATE ALGORITHM=UNDEFINED \*/

/\*!50013 DEFINER=`root`@`localhost` SQL SECURITY DEFINER \*/

/\*!50001 VIEW `getToTransfer` AS select `Dipendente`.`idDipendente` AS `idDipendente` from `Dipendente` where `Dipendente`.`isDaTrasferire` is true \*/

;

/\*!50001 SET character\_set\_client = @saved\_cs\_client \*/

;

/\*!50001 SET character\_set\_results = @saved\_cs\_results \*/

;

/\*!50001 SET collation\_connection = @saved\_col\_connection \*/

;

CREATE USER IF NOT EXISTS 'tempUser' IDENTIFIED BY 'tempUser';

GRANT EXECUTE ON PROCEDURE `mydb`.`loginUser` TO 'tempUser';

GRANT SELECT ON TABLE `mydb`.`Dipendente` TO 'tempUser';

CREATE USER IF NOT EXISTS 'other'IDENTIFIED BY 'otherUser';

GRANT EXECUTE ON PROCEDURE `mydb`.`userNameSurname` TO 'other';

GRANT SELECT ON TABLE `mydb`.`Dipendente` TO 'other';

GRANT EXECUTE ON PROCEDURE `mydb`.`getUserNumeroInterno` TO 'other';

GRANT SELECT ON TABLE `mydb`.`impiegoCorrente` TO 'other';

GRANT SELECT ON TABLE `mydb`.`PostazioneAttiva` TO 'other';

GRANT SELECT ON TABLE `mydb`.`Ufficio` TO 'other';

CREATE USER IF NOT EXISTS 'amministrativo' IDENTIFIED BY 'settoreAmministrativo';

GRANT EXECUTE ON PROCEDURE `mydb`.`editMansione` TO 'amministrativo';

GRANT EXECUTE ON PROCEDURE `mydb`.`getAssegnazioniPassate` TO 'amministrativo';

GRANT EXECUTE ON PROCEDURE `mydb`.`setUserDaTrasferire` TO 'amministrativo';

GRANT SELECT ON TABLE `mydb`.`ImpiegoPassato` TO 'amministrativo';

GRANT EXECUTE ON PROCEDURE `mydb`.`userNameSurname` TO 'amministrativo';

GRANT SELECT, INSERT ON TABLE `mydb`.`Dipendente` TO 'amministrativo';

GRANT EXECUTE ON PROCEDURE `mydb`.`getUserNumeroInterno` TO 'amministrativo';

GRANT SELECT ON TABLE `mydb`.`impiegoCorrente` TO 'amministrativo';

GRANT SELECT ON TABLE `mydb`.`PostazioneAttiva` TO 'amministrativo';

GRANT SELECT ON TABLE `mydb`.`Ufficio` TO 'amministrativo';

CREATE USER IF NOT EXISTS 'spazi'IDENTIFIED BY 'settoreSpazi';

GRANT SELECT ON `mydb`.\*TO 'spazi';

GRANT EXECUTE ON PROCEDURE `mydb`.`assignPostazioneToUfficio` TO 'spazi';

GRANT EXECUTE ON PROCEDURE `mydb`.`assignToPostazioneFree` TO 'spazi';

GRANT EXECUTE ON PROCEDURE `mydb`.`checkPostazioniDisponibili` TO 'spazi';

GRANT EXECUTE ON PROCEDURE `mydb`.`disablePostazione` TO 'spazi';

GRANT EXECUTE ON PROCEDURE `mydb`.`enablePostazione` TO 'spazi';

GRANT EXECUTE ON PROCEDURE `mydb`.`getUserDaTrasferire` TO 'spazi';

GRANT EXECUTE ON PROCEDURE `mydb`.`getUserScambiabili` TO 'spazi';

GRANT EXECUTE ON PROCEDURE `mydb`.`insertNewPostazione` TO 'spazi';

GRANT EXECUTE ON PROCEDURE `mydb`.`makeScambioUser` TO 'spazi';

GRANT UPDATE, INSERT, DELETE ON TABLE `mydb`.`Postazione` TO 'spazi';

GRANT UPDATE, INSERT, DELETE ON TABLE `mydb`.`PostazioneAttiva` TO 'spazi';

GRANT UPDATE TABLE `mydb`.`impiegoCorrente` TO 'spazi';

GRANT UPDATE, INSERT ON TABLE `mydb`.`ImpiegoPassato` TO 'spazi';

GRANT UPDATE TABLE `mydb`.`Ufficio` TO 'spazi';

GRANT EXECUTE ON PROCEDURE `mydb`.`userNameSurname` TO 'spazi';

GRANT EXECUTE ON PROCEDURE `mydb`.`getUserNumeroInterno` TO 'spazi';

FLUSH PRIVILEGES;

## Codice del Front-End

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

**Main.c**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <mysql.h>

#include "program.h"

struct configuration conf;

char nome[64];

void

test\_error (MYSQL \* con, int status)

{

if (status)

{

fprintf (stderr, "Error: %s (errno: %d)\n", mysql\_error (con),

mysql\_errno (con));

}

}

void

test\_stmt\_error (MYSQL\_STMT \* stmt, int status)

{

if (status)

{

fprintf (stderr, "\nErrore: %s\n", mysql\_stmt\_error (stmt));

}

}

int

main (int argc, char \*\*argv)

{

flush\_terminal\_no\_input MYSQL \* con = mysql\_init (NULL);

load\_file (&config, "tempUser.json");

parse\_config ();

if (con == NULL)

{

fprintf (stderr, "Initilization error: %s\n", mysql\_error (con));

exit (1);

}

if (mysql\_real\_connect

(con, conf.host, conf.username, conf.password, conf.database, conf.port,

NULL, CLIENT\_MULTI\_STATEMENTS) == NULL)

{

fprintf (stderr, "Connection error: %s\n", mysql\_error (con));

exit (1);

}

int scelta;

char scelta\_utente[10];

while (1)

{

printf

("\n1 - Effettua l'accesso al sistema\n2 - Termina programma\n\nScegli un opzione: ");

fgets (scelta\_utente, 32, stdin);

printf ("\n\n");

scelta = atoi (scelta\_utente);

switch (scelta)

{

case 1:

flush\_terminal\_no\_input int result = getUserType (con);

if (result == 1)

{

load\_file (&config, "settoreAmministrativo.json");

parse\_config ();

if (mysql\_change\_user

(con, conf.username, conf.password, conf.database) != 0)

{

printf ("Failed to change user. Error: %s\n",

mysql\_error (con));

flushTerminal exit (-1);

}

menuSettoreAmministrativo (con);

}

else if (result == 2)

{

load\_file (&config, "settoreSpazi.json");

parse\_config ();

if (!mysql\_change\_user

(con, conf.username, conf.password, conf.database))

{

fprintf (stderr, "Failed to change user. Error: %s\n",

mysql\_error (con));

}

menuSettoreSpazi (con);

}

else if (result == 3)

{

load\_file (&config, "otherUser.json");

parse\_config ();

if (!mysql\_change\_user

(con, conf.username, conf.password, conf.database))

{

fprintf (stderr, "Failed to change user. Error: %s\n",

mysql\_error (con));

}

menuAltroDipendente (con);

}

else

{

break;

}

case 2:

flush\_terminal\_no\_input mysql\_close (con);

exit (1);

break;

default:

flush\_terminal\_no\_input

printf ("\nInserisci un numero corretto per continuare!\n");

flushTerminal break;

}

}

return 1;

}

void

printResults (MYSQL\_STMT \* statement, MYSQL \* connessione, char \*\*resultName)

{

MYSQL \*con = connessione;

MYSQL\_RES \*result;

MYSQL\_ROW row;

MYSQL\_FIELD \*field;

MYSQL\_RES \*rs\_metadata;

MYSQL\_STMT \*stmt = statement;

bool is\_null[4];

MYSQL\_FIELD \*fields;

int i, num\_fields, status;

MYSQL\_BIND \*rs\_bind;

MYSQL\_TIME \*date;

int resultSet = 0;

int lastResultSet = 0;

i = 0;

do

{

resultSet = i;

printf ("\n");

num\_fields = mysql\_stmt\_field\_count (stmt);

if (num\_fields > 0)

{

rs\_metadata = mysql\_stmt\_result\_metadata (stmt);

test\_stmt\_error (stmt, rs\_metadata == NULL);

fields = mysql\_fetch\_fields (rs\_metadata);

rs\_bind = (MYSQL\_BIND \*) malloc (sizeof (MYSQL\_BIND) \* num\_fields);

if (!rs\_bind)

{

printf ("Cannot allocate output buffers\n");

exit (1);

}

memset (rs\_bind, 0, sizeof (MYSQL\_BIND) \* num\_fields);

for (i = 0; i < num\_fields; ++i)

{

rs\_bind[i].buffer\_type = fields[i].type;

rs\_bind[i].is\_null = &is\_null[i];

rs\_bind[i].buffer = malloc (fields[i].length);

rs\_bind[i].buffer\_length = fields[i].length;

}

status = mysql\_stmt\_bind\_result (stmt, rs\_bind);

test\_stmt\_error (stmt, status);

int counter = 0;

while (1)

{

status = mysql\_stmt\_fetch (stmt);

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

{

break;

}

resultSet = lastResultSet;

for (i = 0; i < num\_fields; ++i)

{

switch (rs\_bind[i].buffer\_type)

{

case MYSQL\_TYPE\_VAR\_STRING:

if (\*rs\_bind[i].is\_null)

printf ("%s = NULL; ", resultName[resultSet]);

else

printf ("%s = %s; ", resultName[resultSet],

(char \*) rs\_bind[i].buffer);

break;

case MYSQL\_TYPE\_TINY:

if (\*rs\_bind[i].is\_null)

printf ("%s = NULL; ", resultName[resultSet]);

else

printf ("%s = %d; ", resultName[resultSet],

\*(bool \*) rs\_bind[i].buffer);

break;

case MYSQL\_TYPE\_LONG:

if (\*rs\_bind[i].is\_null)

printf ("%s = NULL; ", resultName[resultSet]);

else

printf ("%s = %d; ", resultName[resultSet],

\*(int \*) rs\_bind[i].buffer);

break;

case MYSQL\_TYPE\_DATE:

if (\*rs\_bind[i].is\_null)

printf ("%s = NULL; ", resultName[resultSet]);

else

date = rs\_bind[i].buffer;

printf ("%s = %d/%d/%d; ", resultName[resultSet],

date - >day, date - >month, date - >year);

break;

default:

printf ("ERROR: unexpected type (%d)\n",

rs\_bind[resultSet].buffer\_type);

}

resultSet += 1;

}

counter = counter + 1;

printf ("\n");

}

if (counter == 0)

{

printf ("\nNessun risultato disponibile!\n");

}

mysql\_free\_result (rs\_metadata); // free metadata

free (rs\_bind); // free output buffers

}

else

{

}

status = mysql\_stmt\_next\_result (stmt);

lastResultSet = i;

if (status > 0)

test\_stmt\_error (stmt, status);

}

while (status == 0);

for (i = 0; i < num\_fields; ++i)

{

free (rs\_bind[i].buffer);

}

mysql\_stmt\_close (stmt);

flushTerminal return;

}

int

getUserType (MYSQL \* connessione)

{

/\* Questa funzione permette di effettuare il Login, e restituisce il tipo di utente

se il login va a buon fine. \*/

printf ("\n \*\*\*\*\* Login dipendente \*\*\*\*\*\n\n");

MYSQL \*con = connessione;

MYSQL\_STMT \*stmt;

MYSQL\_BIND ps\_params[3];

unsigned long length[3];

MYSQL\_BIND \*rs\_bind;

int status, userType;

bool is\_null[2];

MYSQL\_FIELD \*fields;

MYSQL\_RES \*rs\_metadata;

stmt = mysql\_stmt\_init (con);

if (!stmt)

{

printf ("Could not initialize statement\n");

exit (1);

}

status =

mysql\_stmt\_prepare (stmt, "CALL loginUser(?, ?, ?)",

strlen ("CALL loginUser(?, ?, ?)"));

test\_stmt\_error (stmt, status);

memset (ps\_params, 0, sizeof (ps\_params));

char idDipendenteChar[20];

printf ("Matricola dipendente: ");

getInput (20, idDipendenteChar, false);

int idDipendente = atoi (idDipendenteChar);

length[0] = sizeof (int);

printf ("Password dipendente: ");

char passwordDipendente[20];

getInput (20, passwordDipendente, true);

length[1] = strlen (passwordDipendente);

ps\_params[0].buffer\_type = MYSQL\_TYPE\_LONG;

ps\_params[0].buffer = &idDipendente;

ps\_params[0].buffer\_length = sizeof (int);

ps\_params[0].length = &length[0];

ps\_params[0].is\_null = 0;

ps\_params[1].buffer\_type = MYSQL\_TYPE\_VAR\_STRING;

ps\_params[1].buffer = passwordDipendente;

ps\_params[1].buffer\_length = strlen (passwordDipendente);

ps\_params[1].length = &length[1];

ps\_params[1].is\_null = 0;

status = mysql\_stmt\_bind\_param (stmt, ps\_params);

test\_stmt\_error (stmt, status);

status = mysql\_stmt\_execute (stmt);

test\_stmt\_error (stmt, status);

if (status)

{

flushTerminal return 0;

}

if (mysql\_stmt\_field\_count (stmt) == 0)

{

fprintf (stderr,

"Error while retrieving the stored procedure output\n");

exit (1);

}

rs\_metadata = mysql\_stmt\_result\_metadata (stmt);

fields = mysql\_fetch\_fields (rs\_metadata);

rs\_bind = (MYSQL\_BIND \*) malloc (sizeof (MYSQL\_BIND) \* 1); // We know the number of parameters beforehand

memset (rs\_bind, 0, sizeof (MYSQL\_BIND) \* 1);

int \*idDipendente2 = malloc (sizeof (int));

rs\_bind[0].buffer\_type = MYSQL\_TYPE\_LONG;

rs\_bind[0].is\_null = &is\_null[0];

rs\_bind[0].buffer = idDipendente2;

rs\_bind[0].buffer\_length = sizeof (int);

status = mysql\_stmt\_bind\_result (stmt, rs\_bind);

test\_stmt\_error (stmt, status);

status = mysql\_stmt\_fetch (stmt);

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

{

printf ("Unable to retrieve the information\n");

}

mysql\_free\_result (rs\_metadata); // free metadata

free (rs\_bind); // free output buffers

mysql\_stmt\_close (stmt);

flushTerminal return \*idDipendente2;

}

**Settorespazi.c**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <mysql.h>

#include "program.h"

void getPostazioni(MYSQL \* connessione) {

MYSQL \* con = connessione;

MYSQL\_STMT \* stmt;

int status;

stmt = mysql\_stmt\_init(con);

if (!stmt) {

printf("Could not initialize statement\n");

exit(1);

}

status = mysql\_stmt\_prepare(stmt, "CALL getPostazioni", strlen("CALL getPostazioni"));

test\_stmt\_error(stmt, status);

status = mysql\_stmt\_execute(stmt);

test\_stmt\_error(stmt, status);

if (status) {

flushTerminal

return;

}

char \* toPrint[2] = {

"ID Postazione",

"ID Ufficio"

};

printResults(stmt, con, toPrint);

}

void getPostazioniAttive(MYSQL \* connessione) {

MYSQL \* con = connessione;

MYSQL\_STMT \* stmt;

int status;

stmt = mysql\_stmt\_init(con);

if (!stmt) {

printf("Could not initialize statement\n");

exit(1);

}

status = mysql\_stmt\_prepare(stmt, "CALL getPostazioniAttive", strlen("CALL getPostazioniAttive"));

test\_stmt\_error(stmt, status);

status = mysql\_stmt\_execute(stmt);

test\_stmt\_error(stmt, status);

if (status) {

flushTerminal

return;

}

char \* toPrint[3] = {

"ID Postazione Attiva",

"Disponibile?",

"ID Ufficio"

};

printResults(stmt, con, toPrint);

}

void getUserDaTrasferire(MYSQL \* connessione) {

MYSQL \* con = connessione;

MYSQL\_STMT \* stmt;

int status;

stmt = mysql\_stmt\_init(con);

if (!stmt) {

printf("Could not initialize statement\n");

exit(1);

}

status = mysql\_stmt\_prepare(stmt, "CALL getUserDaTrasferire", strlen("CALL getUserDaTrasferire"));

test\_stmt\_error(stmt, status);

status = mysql\_stmt\_execute(stmt);

test\_stmt\_error(stmt, status);

if (status) {

flushTerminal

return;

}

char \* toPrint[1] = {

"ID DIpendente"

};

printResults(stmt, con, toPrint);

}

void makeScambioUser(MYSQL \* connessione) {

MYSQL \* con = connessione;

MYSQL\_STMT \* stmt;

MYSQL\_BIND ps\_params[2];

unsigned long length[2];

int status;

char nome[64];

printf("ID Dipendente 1: ");

getInput(64, nome, false);

length[0] = sizeof(int);

int idDipendente = atoi(nome);

char nome1[64];

printf("ID Dipendente 2: ");

getInput(64, nome1, false);

length[1] = sizeof(int);

int idDipendente1 = atoi(nome1);

stmt = mysql\_stmt\_init(con);

if (!stmt) {

printf("Could not initialize statement\n");

exit(1);

}

status = mysql\_stmt\_prepare(stmt, "CALL makeScambioUser(?, ?)", strlen("CALL makeScambioUser(?, ?)"));

test\_stmt\_error(stmt, status);

memset(ps\_params, 0, sizeof(ps\_params));

ps\_params[0].buffer\_type = MYSQL\_TYPE\_LONG;

ps\_params[0].buffer = & idDipendente;

ps\_params[0].buffer\_length = sizeof(int);

ps\_params[0].length = & length[0];

ps\_params[0].is\_null = 0;

ps\_params[1].buffer\_type = MYSQL\_TYPE\_LONG;

ps\_params[1].buffer = & idDipendente1;

ps\_params[1].buffer\_length = sizeof(int);

ps\_params[1].length = & length[1];

ps\_params[1].is\_null = 0;

int idRisultato = 0;

status = mysql\_stmt\_bind\_param(stmt, ps\_params);

test\_stmt\_error(stmt, status);

status = mysql\_stmt\_execute(stmt);

test\_stmt\_error(stmt, status);

if (status) {

flushTerminal

return;

}

printf("\n\n ---> Scambio utenti effettuato correttamente <--- \n\n");

flushTerminal

}

void checkUserScambiabili(MYSQL \* connessione) {

MYSQL \* con = connessione;

MYSQL\_STMT \* stmt;

MYSQL\_BIND ps\_params[2]; // input parameter buffers

unsigned long length[2]; // Can do like that because all IN parameters have the same length

int status;

char nome[64];

printf("ID Dipendente: ");

getInput(64, nome, false);

length[0] = sizeof(int);

int idDipendente = atoi(nome);

stmt = mysql\_stmt\_init(con);

if (!stmt) {

printf("Could not initialize statement\n");

exit(1);

}

status = mysql\_stmt\_prepare(stmt, "CALL getUserScambiabili(?)", strlen("CALL getUserScambiabili(?)"));

test\_stmt\_error(stmt, status);

memset(ps\_params, 0, sizeof(ps\_params));

ps\_params[0].buffer\_type = MYSQL\_TYPE\_LONG;

ps\_params[0].buffer = & idDipendente;

ps\_params[0].buffer\_length = sizeof(int);

ps\_params[0].length = & length[0];

ps\_params[0].is\_null = 0;

int idRisultato = 0;

status = mysql\_stmt\_bind\_param(stmt, ps\_params);

test\_stmt\_error(stmt, status);

status = mysql\_stmt\_execute(stmt);

test\_stmt\_error(stmt, status);

if (status) {

flushTerminal

return;

}

char \* toPrint[2] = {

"ID Dipendente",

"ID Postazione"

};

printResults(stmt, con, toPrint);

}

void inserisciPostazione(MYSQL \* connessione) {

MYSQL \* con = connessione;

MYSQL\_STMT \* stmt;

MYSQL\_BIND \* rs\_bind;

MYSQL\_RES \* rs\_metadata;

bool is\_null[2];

MYSQL\_FIELD \* fields;

MYSQL\_BIND ps\_params[2]; // input parameter buffers

unsigned long length[2]; // Can do like that because all IN parameters have the same length

int status;

int idPostazione;

char nome[64];

printf("ID Ufficio: ");

getInput(64, nome, false);

length[0] = sizeof(int);

int idDipendente = atoi(nome);

stmt = mysql\_stmt\_init(con);

if (!stmt) {

printf("Could not initialize statement\n");

exit(1);

}

status = mysql\_stmt\_prepare(stmt, "CALL insertNewPostazione(?, ?)", strlen("CALL insertNewPostazione(?, ?)"));

test\_stmt\_error(stmt, status);

memset(ps\_params, 0, sizeof(ps\_params));

ps\_params[0].buffer\_type = MYSQL\_TYPE\_LONG;

ps\_params[0].buffer = & idDipendente;

ps\_params[0].buffer\_length = sizeof(int);

ps\_params[0].length = & length[0];

ps\_params[0].is\_null = 0;

int idRisultato = 0;

status = mysql\_stmt\_bind\_param(stmt, ps\_params);

test\_stmt\_error(stmt, status);

status = mysql\_stmt\_execute(stmt);

test\_stmt\_error(stmt, status);

if (status) {

flushTerminal

return;

}

rs\_metadata = mysql\_stmt\_result\_metadata(stmt);

fields = mysql\_fetch\_fields(rs\_metadata);

rs\_bind = (MYSQL\_BIND \* ) malloc(sizeof(MYSQL\_BIND) \* 1); // We know the number of parameters beforehand

memset(rs\_bind, 0, sizeof(MYSQL\_BIND) \* 1);

int \* idDipendente2 = malloc(sizeof(int));

rs\_bind[0].buffer\_type = MYSQL\_TYPE\_LONG;

rs\_bind[0].is\_null = & is\_null[0];

rs\_bind[0].buffer = & idPostazione;

rs\_bind[0].buffer\_length = sizeof(int);

status = mysql\_stmt\_bind\_result(stmt, rs\_bind);

test\_stmt\_error(stmt, status);

status = mysql\_stmt\_fetch(stmt);

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

printf("Unable to retrieve the information\n");

}

printf("\n\n ---> Postazione inserita correttamente <--- \n ID Della postazione inserita: %d\n", idPostazione);

mysql\_free\_result(rs\_metadata); // free metadata

free(rs\_bind); // free output buffers

mysql\_stmt\_close(stmt);

flushTerminal

return;

}

void disabilitaPostazione(MYSQL \* connessione) {

MYSQL \* con = connessione;

MYSQL\_STMT \* stmt;

MYSQL\_BIND ps\_params[2]; // input parameter buffers

unsigned long length[2]; // Can do like that because all IN parameters have the same length

int status;

char nome[64];

printf("ID Postazione: ");

getInput(64, nome, false);

length[0] = sizeof(int);

int idDipendente = atoi(nome);

stmt = mysql\_stmt\_init(con);

if (!stmt) {

printf("Could not initialize statement\n");

exit(1);

}

status = mysql\_stmt\_prepare(stmt, "CALL disablePostazione(?)", strlen("CALL disablePostazione(?)"));

test\_stmt\_error(stmt, status);

memset(ps\_params, 0, sizeof(ps\_params));

ps\_params[0].buffer\_type = MYSQL\_TYPE\_LONG;

ps\_params[0].buffer = & idDipendente;

ps\_params[0].buffer\_length = sizeof(int);

ps\_params[0].length = & length[0];

ps\_params[0].is\_null = 0;

int idRisultato = 0;

status = mysql\_stmt\_bind\_param(stmt, ps\_params);

test\_stmt\_error(stmt, status);

status = mysql\_stmt\_execute(stmt);

test\_stmt\_error(stmt, status);

if (status) {

flushTerminal

return;

}

printf("\n\n ---> Postazione disabilitata correttamente <--- \n\n");

flushTerminal

return;

}

void abilitaPostazione(MYSQL \* connessione) {

MYSQL \* con = connessione;

MYSQL\_STMT \* stmt;

MYSQL\_BIND ps\_params[2];

unsigned long length[2];

int status;

char nome[64];

printf("ID Postazione: ");

getInput(64, nome, false);

length[0] = sizeof(int);

int idDipendente = atoi(nome);

stmt = mysql\_stmt\_init(con);

if (!stmt) {

printf("Could not initialize statement\n");

exit(1);

}

status = mysql\_stmt\_prepare(stmt, "CALL enablePostazione(?)", strlen("CALL enablePostazione(?)"));

test\_stmt\_error(stmt, status);

memset(ps\_params, 0, sizeof(ps\_params));

ps\_params[0].buffer\_type = MYSQL\_TYPE\_LONG;

ps\_params[0].buffer = & idDipendente;

ps\_params[0].buffer\_length = sizeof(int);

ps\_params[0].length = & length[0];

ps\_params[0].is\_null = 0;

int idRisultato = 0;

status = mysql\_stmt\_bind\_param(stmt, ps\_params);

test\_stmt\_error(stmt, status);

status = mysql\_stmt\_execute(stmt);

test\_stmt\_error(stmt, status);

if (status) {

flushTerminal

return;

}

printf("\n\n ---> Postazione abilitata correttamente <--- \n\n");

flushTerminal

return;

}

void checkPostazioniDisponibili(MYSQL \* connessione) {

MYSQL \* con = connessione;

MYSQL\_STMT \* stmt;

MYSQL\_BIND ps\_params[2]; // input parameter buffers

unsigned long length[2]; // Can do like that because all IN parameters have the same length

int status;

char nome[64];

printf("ID Dipendente: ");

getInput(64, nome, false);

length[0] = sizeof(int);

int idDipendente = atoi(nome);

stmt = mysql\_stmt\_init(con);

if (!stmt) {

printf("Could not initialize statement\n");

exit(1);

}

status = mysql\_stmt\_prepare(stmt, "CALL checkPostazioniDisponibili(?)", strlen("CALL checkPostazioniDisponibili(?)"));

test\_stmt\_error(stmt, status);

memset(ps\_params, 0, sizeof(ps\_params));

ps\_params[0].buffer\_type = MYSQL\_TYPE\_LONG;

ps\_params[0].buffer = & idDipendente;

ps\_params[0].buffer\_length = sizeof(int);

ps\_params[0].length = & length[0];

ps\_params[0].is\_null = 0;

int idRisultato = 0;

status = mysql\_stmt\_bind\_param(stmt, ps\_params);

test\_stmt\_error(stmt, status);

status = mysql\_stmt\_execute(stmt);

test\_stmt\_error(stmt, status);

if (status) {

flushTerminal

return;

}

char \* toPrint[1] = {

"ID Postazione Attiva"

};

printResults(stmt, con, toPrint);

}

void assignToPostazioneFree(MYSQL \* connessione) {

MYSQL \* con = connessione;

MYSQL\_STMT \* stmt;

MYSQL\_BIND ps\_params[2]; // input parameter buffers

unsigned long length[2]; // Can do like that because all IN parameters have the same length

int status;

char nome[64];

printf("ID Dipendente: ");

getInput(64, nome, false);

length[0] = sizeof(int);

int idDipendente = atoi(nome);

char mansione[64];

printf("ID Postazione: ");

getInput(64, mansione, false);

length[1] = sizeof(int);

int idMansione = atoi(mansione);

stmt = mysql\_stmt\_init(con);

if (!stmt) {

printf("Could not initialize statement\n");

exit(1);

}

status = mysql\_stmt\_prepare(stmt, "CALL assignToPostazioneFree(?,?)", strlen("CALL assignToPostazioneFree(?,?)"));

test\_stmt\_error(stmt, status);

memset(ps\_params, 0, sizeof(ps\_params));

ps\_params[0].buffer\_type = MYSQL\_TYPE\_LONG;

ps\_params[0].buffer = & idDipendente;;

ps\_params[0].buffer\_length = sizeof(int);

ps\_params[0].length = & length[0];

ps\_params[0].is\_null = 0;

ps\_params[1].buffer\_type = MYSQL\_TYPE\_LONG;

ps\_params[1].buffer = & idMansione;;

ps\_params[1].buffer\_length = sizeof(int);

ps\_params[1].length = & length[1];

ps\_params[1].is\_null = 0;

status = mysql\_stmt\_bind\_param(stmt, ps\_params);

test\_stmt\_error(stmt, status);

status = mysql\_stmt\_execute(stmt);

test\_stmt\_error(stmt, status);

if (status) {

flushTerminal

return;

}

printf("\nUtente assegnato correttamente!\n");

flushTerminal

mysql\_stmt\_close(stmt);

return;

}

void assegnaPostazioneToUfficio(MYSQL \* connessione) {

MYSQL \* con = connessione;

MYSQL\_STMT \* stmt;

MYSQL\_BIND ps\_params[2]; // input parameter buffers

unsigned long length[2]; // Can do like that because all IN parameters have the same length

int status;

char nome[64];

printf("ID Postazione: ");

getInput(64, nome, false);

length[0] = sizeof(int);

int idDipendente = atoi(nome);

char mansione[64];

printf("ID Ufficio: ");

getInput(64, mansione, false);

length[1] = sizeof(int);

int idMansione = atoi(mansione);

stmt = mysql\_stmt\_init(con);

if (!stmt) {

printf("Could not initialize statement\n");

exit(1);

}

status = mysql\_stmt\_prepare(stmt, "CALL assignPostazioneToUfficio(?,?)", strlen("CALL assignPostazioneToUfficio(?,?)"));

test\_stmt\_error(stmt, status);

memset(ps\_params, 0, sizeof(ps\_params));

ps\_params[0].buffer\_type = MYSQL\_TYPE\_LONG;

ps\_params[0].buffer = & idDipendente;;

ps\_params[0].buffer\_length = sizeof(int);

ps\_params[0].length = & length[0];

ps\_params[0].is\_null = 0;

ps\_params[1].buffer\_type = MYSQL\_TYPE\_LONG;

ps\_params[1].buffer = & idMansione;;

ps\_params[1].buffer\_length = sizeof(int);

ps\_params[1].length = & length[1];

ps\_params[1].is\_null = 0;

status = mysql\_stmt\_bind\_param(stmt, ps\_params);

test\_stmt\_error(stmt, status);

status = mysql\_stmt\_execute(stmt);

test\_stmt\_error(stmt, status);

if (status) {

flushTerminal

return;

}

printf("\nPostazione assegnata correttamente!\n");

flushTerminal

mysql\_stmt\_close(stmt);

return;

}

int menuSettoreSpazi(MYSQL \* connessione) {

MYSQL \* con = connessione;

int scelta;

char scelta\_utente[10];

while (1) {

printf("\n \*\*\*\*\* Tipo dipendente: Settore spazi \*\*\*\*\*\n");

printf("\n1 - Postazioni disponibili per l'utente\n2 - Assegna dipendente ad una postazioni libera\n3 - checkUserScambiabili\n4 - makeScambioUser\n5 - getPostazioni\n6 - getPostazioniAttive\n7 - getUserDaTrasferire\n8 - disabilitaPostazione\n9 - abilitaPostazione\n10 - inserisciPostazione\n11 - Assegna postazione ad ufficio\n12 - getUserNumeroInterno\n13 - getNameSurname\n14 - Cerca dipendente numero intero\n15 - Cerca dipendente nome e/o cognome\n16 - Termina esecuzione\n\nScegli un opzione: ");

fgets(scelta\_utente, 32, stdin);

scelta = atoi(scelta\_utente);

switch (scelta) {

case 1:

flush\_terminal\_no\_input

checkPostazioniDisponibili(con);

break;

case 2:

flush\_terminal\_no\_input

assignToPostazioneFree(con);

break;

case 3:

flush\_terminal\_no\_input

checkUserScambiabili(con);

break;

case 4:

flush\_terminal\_no\_input

makeScambioUser(con);

break;

case 5:

flush\_terminal\_no\_input

getPostazioni(con);

break;

case 6:

flush\_terminal\_no\_input

getPostazioniAttive(con);

break;

case 7:

flush\_terminal\_no\_input

getUserDaTrasferire(con);

break;

case 8:

flush\_terminal\_no\_input

disabilitaPostazione(con);

break;

case 9:

flush\_terminal\_no\_input

abilitaPostazione(con);

break;

case 10:

flush\_terminal\_no\_input

inserisciPostazione(con);

break;

case 11:

flush\_terminal\_no\_input

assegnaPostazioneToUfficio(con);

break;

case 12:

flush\_terminal\_no\_input

getUserNumeroInterno(con);

break;

case 13:

flush\_terminal\_no\_input

getNameSurname(con);

break;

case 14:

flush\_terminal\_no\_input

getUserNumeroInterno(con);

break;

case 15:

flush\_terminal\_no\_input

getNameSurname(con);

break;

case 16:

flush\_terminal\_no\_input

mysql\_close(con);

exit(1);

break;

default:

flush\_terminal\_no\_input

printf("\nInserisci un numero corretto per continuare!\n");

flushTerminal

break;

}

}

}

**otheruser.c**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <mysql.h>

#include "program.h"

int menuAltroDipendente(MYSQL \* connessione) {

MYSQL \* con = connessione;

int scelta;

char scelta\_utente[10];

while (1) {

printf("\n \*\*\*\*\* Tipo dipendente: Altro dipendente \*\*\*\*\*\n");

printf("\n1 - Cerca dipendente numero intero\n2 - Cerca dipendente nome e/o cognome\n3 - Termina esecuzione\n\nScegli un opzione: ");

fgets(scelta\_utente, 32, stdin);

scelta = atoi(scelta\_utente);

switch (scelta) {

case 1:

flush\_terminal\_no\_input

getUserNumeroInterno(connessione);

break;

case 2:

flush\_terminal\_no\_input

getNameSurname(connessione);

break;

case 3:

flush\_terminal\_no\_input

mysql\_close(con);

exit(1);

break;

default:

flush\_terminal\_no\_input

printf("\nInserisci un numero corretto per continuare!\n");

flushTerminal

break;

}

}

}

void getUserNumeroInterno(MYSQL \* connessione) {

printf("\n \*\*\*\*\* getUserNumeroInterno \*\*\*\*\*\n\n");

MYSQL \* con = connessione;

MYSQL\_STMT \* stmt;

MYSQL\_BIND ps\_params[1]; // input parameter buffers

unsigned long length[1]; // Can do like that because all IN parameters have the same length

int status;

char nome[64];

printf("Numero interno: ");

getInput(64, nome, false);

length[0] = sizeof(int);

int idDipendente = atoi(nome);

stmt = mysql\_stmt\_init(con);

if (!stmt) {

printf("Could not initialize statement\n");

exit(1);

}

status = mysql\_stmt\_prepare(stmt, "CALL getUserNumeroInterno(?)", strlen("CALL getUserNumeroInterno(?)"));

test\_stmt\_error(stmt, status);

memset(ps\_params, 0, sizeof(ps\_params));

ps\_params[0].buffer\_type = MYSQL\_TYPE\_LONG;

ps\_params[0].buffer = & idDipendente;

ps\_params[0].buffer\_length = sizeof(int);

ps\_params[0].length = & length[0];

ps\_params[0].is\_null = 0;

status = mysql\_stmt\_bind\_param(stmt, ps\_params);

test\_stmt\_error(stmt, status);

status = mysql\_stmt\_execute(stmt);

test\_stmt\_error(stmt, status);

if (status) {

flushTerminal

return;

}

char \* toPrint[6] = {

"ID Ufficio",

"Piano",

"Edificio",

"Mansione",

"ID DIpendente",

"Is da trasferire?"

};

printResults(stmt, con, toPrint);

}

void getNameSurname(MYSQL \* connessione) {

printf("\n \*\*\*\*\* getUserNumeroInterno \*\*\*\*\*\n\n");

MYSQL \* con = connessione;

MYSQL\_STMT \* stmt;

MYSQL\_BIND ps\_params[2]; // input parameter buffers

unsigned long length[2]; // Can do like that because all IN parameters have the same length

int status;

char nome[64];

printf("Nome dipendente: ");

getInput(64, nome, false);

length[0] = strlen(nome);

char cognome[64];

printf("Cognome dipendente: ");

getInput(64, cognome, false);

length[1] = strlen(cognome);

stmt = mysql\_stmt\_init(con);

if (!stmt) {

printf("Could not initialize statement\n");

exit(1);

}

status = mysql\_stmt\_prepare(stmt, "CALL userNameSurname(?, ?)", strlen("CALL userNameSurname(?, ?)"));

test\_stmt\_error(stmt, status);

memset(ps\_params, 0, sizeof(ps\_params));

ps\_params[0].buffer\_type = MYSQL\_TYPE\_VAR\_STRING;

ps\_params[0].buffer = nome;

ps\_params[0].buffer\_length = strlen(nome);

ps\_params[0].length = & length[0];

ps\_params[0].is\_null = 0;

ps\_params[1].buffer\_type = MYSQL\_TYPE\_VAR\_STRING;

ps\_params[1].buffer = cognome;

ps\_params[1].buffer\_length = strlen(cognome);

ps\_params[1].length = & length[1];

ps\_params[1].is\_null = 0;

status = mysql\_stmt\_bind\_param(stmt, ps\_params);

test\_stmt\_error(stmt, status);

status = mysql\_stmt\_execute(stmt);

test\_stmt\_error(stmt, status);

if (status) {

flushTerminal

return;

}

char \* toPrint[5] = {

"E-Mail Ufficio",

"E-Mail Personale",

"ID Ufficio",

"Piano",

"Edificio"

};

printResults(stmt, con, toPrint);

}

**settoreamministrativo.c**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <mysql.h>

#include "program.h"

int menuSettoreAmministrativo(MYSQL \* connessione) {

MYSQL \* con = connessione;

int scelta;

char scelta\_utente[10];

while (1) {

printf("\n \*\*\*\*\* Tipo dipendente: Settore amministrativo \*\*\*\*\*\n");

printf("\n1 - Visualizza assegnazioni passate dipendente\n2 - Modifica mansione dipendente\n3 - Indica utente come da trasferire\n4 - getUserNumeroInterno\n5 - getNameSurname\n6 - Termina esecuzione\n\nScegli un opzione: ");

fgets(scelta\_utente, 32, stdin);

scelta = atoi(scelta\_utente);

switch (scelta) {

case 1: // Login

flush\_terminal\_no\_input

getAssegnazioniPassate(con);

break;

case 2: // Leggi tutti i messaggi presenti

flush\_terminal\_no\_input

editMansione(con);

break;

case 3: // Inserimento nuovo messaggio

flush\_terminal\_no\_input

setDaTrasferire(con);

break;

case 4:

flush\_terminal\_no\_input

getUserNumeroInterno(con);

break;

case 5:

flush\_terminal\_no\_input

getNameSurname(con);

break;

case 6:

exit(1);

break;

default:

flush\_terminal\_no\_input

printf("\nInserisci un numero corretto per continuare!\n");

flushTerminal

break;

}

}

}

void getAssegnazioniPassate(MYSQL \* connessione) {

printf("\n \*\*\*\*\* getAssegnazioniPassate \*\*\*\*\*\n\n");

MYSQL \* con = connessione;

MYSQL\_STMT \* stmt;

MYSQL\_BIND ps\_params[1];

unsigned long length[1];

int status;

char nome[64];

printf("ID Dipendente: ");

getInput(64, nome, false);

length[0] = sizeof(int);

int idDipendente = atoi(nome);

stmt = mysql\_stmt\_init(con);

if (!stmt) {

printf("Could not initialize statement\n");

exit(1);

}

status = mysql\_stmt\_prepare(stmt, "CALL getAssegnazioniPassate(?)", strlen("CALL getAssegnazioniPassate(?)"));

test\_stmt\_error(stmt, status);

memset(ps\_params, 0, sizeof(ps\_params));

ps\_params[0].buffer\_type = MYSQL\_TYPE\_LONG;

ps\_params[0].buffer = & idDipendente;

ps\_params[0].buffer\_length = sizeof(int);

ps\_params[0].length = & length[0];

ps\_params[0].is\_null = 0;

status = mysql\_stmt\_bind\_param(stmt, ps\_params);

test\_stmt\_error(stmt, status);

status = mysql\_stmt\_execute(stmt);

test\_stmt\_error(stmt, status);

if (status) {

flushTerminal

return;

}

char \* toPrint[4] = {

"ID Assegnazione",

"Data inizio",

"Data fine",

"ID Postazione"

};

printResults(stmt, con, toPrint);

}

void setDaTrasferire(MYSQL \* connessione) {

MYSQL \* con = connessione;

MYSQL\_STMT \* stmt;

MYSQL\_BIND ps\_params[1]; // input parameter buffers

unsigned long length[1]; // Can do like that because all IN parameters have the same length

int status;

char nome[64];

printf("ID Dipendente: ");

getInput(64, nome, false);

length[0] = sizeof(int);

int idDipendente = atoi(nome);

stmt = mysql\_stmt\_init(con);

if (!stmt) {

printf("Could not initialize statement\n");

exit(1);

}

status = mysql\_stmt\_prepare(stmt, "CALL setUserDaTrasferire(?)", strlen("CALL setUserDaTrasferire(?)"));

test\_stmt\_error(stmt, status);

memset(ps\_params, 0, sizeof(ps\_params));

ps\_params[0].buffer\_type = MYSQL\_TYPE\_LONG;

ps\_params[0].buffer = & idDipendente;

ps\_params[0].buffer\_length = sizeof(int);

ps\_params[0].length = & length[0];

ps\_params[0].is\_null = 0;

status = mysql\_stmt\_bind\_param(stmt, ps\_params);

test\_stmt\_error(stmt, status);

status = mysql\_stmt\_execute(stmt);

test\_stmt\_error(stmt, status);

if (status) {

flushTerminal

return;

}

printf("\nUtente indicato come da trasferire correttamente!\n");

flushTerminal

}

void editMansione(MYSQL \* connessione) {

MYSQL \* con = connessione;

MYSQL\_STMT \* stmt;

MYSQL\_BIND ps\_params[2]; // input parameter buffers

unsigned long length[2]; // Can do like that because all IN parameters have the same length

int status;

char nome[64];

printf("ID Dipendente: ");

getInput(64, nome, false);

length[0] = sizeof(int);

int idDipendente = atoi(nome);

char mansione[64];

printf("\nID Mansione: ");

getInput(64, mansione, false);

length[1] = sizeof(int);

int idMansione = atoi(mansione);

stmt = mysql\_stmt\_init(con);

if (!stmt) {

printf("Could not initialize statement\n");

exit(1);

}

status = mysql\_stmt\_prepare(stmt, "CALL editMansione(?, ?)", strlen("CALL editMansione(?, ?)"));

test\_stmt\_error(stmt, status);

memset(ps\_params, 0, sizeof(ps\_params));

ps\_params[0].buffer\_type = MYSQL\_TYPE\_LONG;

ps\_params[0].buffer = & idDipendente;

ps\_params[0].buffer\_length = sizeof(int);

ps\_params[0].length = & length[0];

ps\_params[0].is\_null = 0;

ps\_params[1].buffer\_type = MYSQL\_TYPE\_LONG;

ps\_params[1].buffer = & idMansione;

ps\_params[1].buffer\_length = sizeof(int);

ps\_params[1].length = & length[1];

ps\_params[1].is\_null = 0;

status = mysql\_stmt\_bind\_param(stmt, ps\_params);

test\_stmt\_error(stmt, status);

status = mysql\_stmt\_execute(stmt);

test\_stmt\_error(stmt, status);

if (status) {

flushTerminal

return;

}

printf("\nMansione modificata correttamente!\n");

flushTerminal

}

1. Indicare con E le entità, con R le relazioni [↑](#footnote-ref-2)
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-3)
3. PK = primary key, NN = not null, UQ = unique, UN = unsigned, AI = auto increment. È ovviamente possibile specificare più di un attributo per ciascuna colonna. [↑](#footnote-ref-4)
4. PK = primary key, NN = not null, UQ = unique, UN = unsigned, AI = auto increment. È ovviamente possibile specificare più di un attributo per ciascuna colonna. [↑](#footnote-ref-5)
5. PK = primary key, NN = not null, UQ = unique, UN = unsigned, AI = auto increment. È ovviamente possibile specificare più di un attributo per ciascuna colonna. [↑](#footnote-ref-6)
6. PK = primary key, NN = not null, UQ = unique, UN = unsigned, AI = auto increment. È ovviamente possibile specificare più di un attributo per ciascuna colonna. [↑](#footnote-ref-7)
7. PK = primary key, NN = not null, UQ = unique, UN = unsigned, AI = auto increment. È ovviamente possibile specificare più di un attributo per ciascuna colonna. [↑](#footnote-ref-8)
8. PK = primary key, NN = not null, UQ = unique, UN = unsigned, AI = auto increment. È ovviamente possibile specificare più di un attributo per ciascuna colonna. [↑](#footnote-ref-9)
9. IDX = index, UQ = unique, FT = full text, PR = primary. [↑](#footnote-ref-10)
10. IDX = index, UQ = unique, FT = full text, PR = primary. [↑](#footnote-ref-11)
11. IDX = index, UQ = unique, FT = full text, PR = primary. [↑](#footnote-ref-12)
12. IDX = index, UQ = unique, FT = full text, PR = primary. [↑](#footnote-ref-13)
13. IDX = index, UQ = unique, FT = full text, PR = primary. [↑](#footnote-ref-14)
14. IDX = index, UQ = unique, FT = full text, PR = primary. [↑](#footnote-ref-15)