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

**Basi di Dati**

**Progetto A.A. 2021/2022**

TITOLO DEL PROGETTO

0268364

Filippo Muscherà

**Indice**

[1. Descrizione del Minimondo 3](#_Toc606296459)

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

[3. Progettazione concettuale 5](#_Toc2081466291)

[4. Progettazione logica 6](#_Toc2147004904)

[5. Progettazione fisica 8](#_Toc518560220)

[Appendice: Implementazione 9](#_Toc403811585)

# Descrizione del Minimondo

|  |  |  |
| --- | --- | --- |
|  | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34 | 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. 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 ubicazione di un qualsiasi altro dipendente, ricercandolo per nome, per cognome o utilizzando entrambi. Inoltre, un dipendente può ricercare un certo numero di telefono 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** |
| 9 | Dipendenti | Dipendenti del settore spazi | Disambiguare i dipendenti “base” da quelli che lavorano nella Gestione Spazi, perché questi ultimi possono effettuare trasferimenti dei dipendenti, mentre i dipendenti “base” no. |
| 10 | Assegnati a particolari uffici | Assegnati ai rispettivi uffici, e sono anch’essi soggetti a turnazione | Rendere esplicito il fatto che i dipendenti del settore spazi, come tutti gli altri dipendenti, sono soggetti a turnazione. |
| 15 | Spostati | Trasferiti | Rendere esplicito il riferimento all’operazione di trasferimento di un dipendente |
| 16 | Ufficio Spazi | Settore Spazi | Mantenere consistenti i nomi utilizzati. |
| 17 | Tra le mansioni e gli uffici | Tra la mansione assegnata al dipendente e quella associata all’ufficio | Rendere esplicita la differenza tra la mansione che viene assegnata a un dipendente e quella che invece è associata ad un ufficio: un dipendente è assegnato a una mansione, che deve svolgere in una postazione che fa parte di un ufficio che sia associato alla stessa mansione. La mansione del dipendente però può essere modificata nel corso del tempo dal Settore Amministrativo, mentre quella di un ufficio rimane fissa. È quindi importante marcare la differenza tra le due. |
| 22 | Settore Amministrativo | I dipendenti del Settore Amministrativo | Disambiguare il fatto che il Settore Amministrativo non è un’entità esterna ma è anch’essa parte dell'azienda ed è composta da dipendenti. |
| 25 | Nel tempo | Negli ultimi tre anni | Specificare il lasso temporale. Si sceglono tre anni poichè è quello il periodo di tempo di interesse per la directory aziendale e i trasferimenti. |
| 28 | Numero di telefono | Numero di telefono esterno | Esistono numeri di telefono interno ed esterno. In questo caso è quello esterno perché deve essere univoco, e quello interno potrebbe essere lo stesso per postazioni di uffici diversi, mentre quello esterno è sicuramente univoco. |

### Specifica disambiguata

|  |
| --- |
| 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 composto da 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 e edifici è demandata al Settore Spazi dell’azienda, composto da dipendenti del settore spazi che sono assegnati ai rispettivi uffici, e sono anch’essi soggetti a turnazione.  L’assegnazione di un dipendente ad un ufficio avviene sulla 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 . Tale trasferimento deve essere effettuato rispettando l’associazione tra la mansione assegnata al dipendente e quella associata all’ufficio. 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.  I dipendenti del Settore Amministrativo possono 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, un dipendente del settore amministrativo può generare, per ciascun dipendente, un report indicante a quali uffici esso è stato assegnato negli ultimi tre anni.  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 esterno 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 e-mail personale ed uno associato all’ufficio a cui fa capo. |

## Glossario dei Termini

|  |  |  |  |
| --- | --- | --- | --- |
| **Termine** | **Descrizione** | **Sinonimi** | **Collegamenti** |
| Dipendente | Lavoratore generico dell’azienda | Lavoratore, Impiegato | Mansione, Ufficio |
| Mansione | Lavoro svolto da ogni dipendente | Compito, Lavoro | Dipendente, Ufficio |
| Ufficio | Locale destinato a impiegati che svolgono una determinata mansione |  | Dipendente, Mansione, Piano |
| Edificio | Struttura che contiene uno o più piani |  | Piano |
| Piano | Uno dei livelli che costituiscono un edificio. Contiene uno o più uffici | Livello | Edificio, Ufficio |
| Postazione | Collocazione all’interno di un ufficio a cui può essere assegnato un impiegato |  | Ufficio, Dipendente, Telefono Interno, Telefono Esterno |
| Telefono Interno | Numero di telefono valido per chiamate tra postazioni dello stesso ufficio |  | Postazione |
| Telefono Esterno | Numero di telefono valido per chiamate tra postazioni di uffici differenti |  | Postazione |
| Settore Spazi | Insieme degli uffici e dei dipendenti che sono demandati alla gestione delle turnazioni di tutti i dipendenti |  | Dipendente, Postazione |
| Data Ultimo Trasferimento | Data in cui il dipendente è stato assegnato alla sua corrente postazione. |  | Dipendente |
| Trasferimento | Spostamento di un dipendente da una postazione ad un’altra. Viene effettuato dal Settore Spazi |  | Dipendente, Settore Spazi |
| Settore Amministrativo | Insieme degli uffici e dei dipendenti che hanno la possibilità di cambiare la mansione a cui è assegnato un dipendente |  | Dipendente, Mansione |

## Raggruppamento dei requisiti in insiemi omogenei

|  |
| --- |
| **Frasi relative a “Dipendente”** |
| * identificato da codice fiscale, nome, cognome, data di nascita, luogo di nascita, indirizzo di residenza svolge una particolare mansione all’interno dell’azienda. * L’assegnazione di un dipendente ad un ufficio avviene sulla base di turnazioni periodiche. Ogni dipendente, pertanto, è associato alla data dell’ultimo trasferimento. * 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 esterno 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. * Un dipendente ha un indirizzo e-mail personale ed uno associato all’ufficio a cui fa capo. |

|  |
| --- |
| **Frasi relative a “Ufficio”** |
| * L’azienda ha a disposizione un numero arbitrario di edifici, ciascuno composto da 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. |

|  |
| --- |
| **Frasi relative a “Settore Spazi”** |
| * La gestione degli uffici, piani, edifici è demandata al Settore Spazi dell’azienda, composto da dipendenti del settore spazi che sono assegnati ai rispettivi uffici, e sono anch’essi soggetti a turnazione. * 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. * 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. |

|  |
| --- |
| **Frasi relative a “Settore Amministrativo”** |
| * I dipendenti del Settore Amministrativo possono decidere in qualsiasi momento di modificare la mansione di un dipendente. * Inoltre, un dipendente del settore amministrativo può generare, per ciascun dipendente, un report indicante a quali uffici esso è stato assegnato nel tempo. |

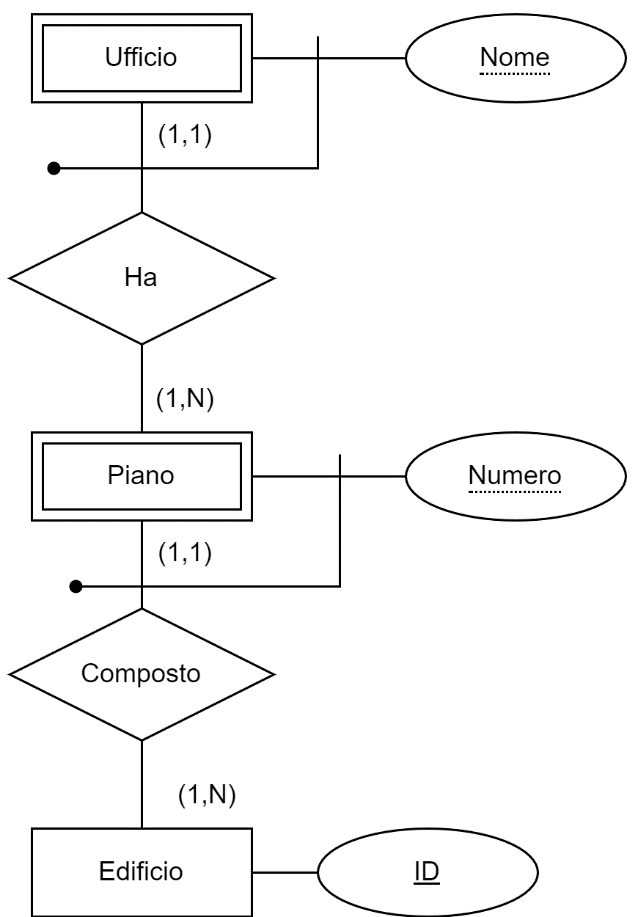
# Progettazione concettuale

## Costruzione dello schema E-R

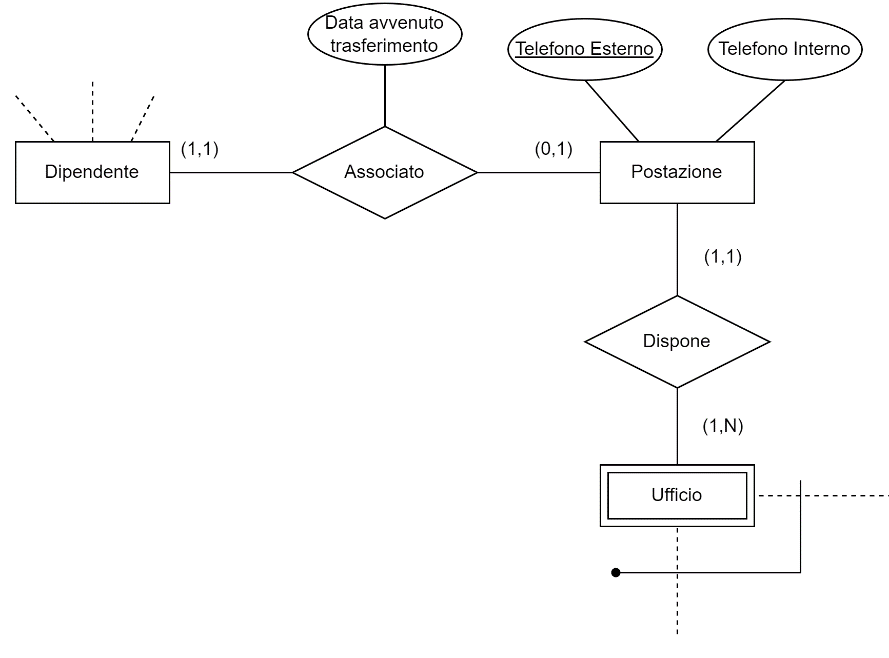
### Schema E-R dell’entità “Dipendente”, che ha come possibili identificatori il Codice Fiscale o il Nome e il Cognome.

Al dipendente sono associati tutti gli attributi richiesti dalla specifica

L’attributo “Indirizzo” è un attributo composto perché aggrega dati che sono correlati tra di loro.



La composizione degli uffici viene rappresentata a partire dagli edifici, identificati per mezzo di un ID, ovvero un codice alfanumerico (il nome o un numero che li identifichi) assegnato ad ogni edificio, in modo da poter distinguere anche edifici che si trovano nello stesso complesso (e che quindi avrebbero lo stesso indirizzo). Ogni edificio ha collegati ad esso un numero variabile di Piani, che a sua volta ospita al suo interno un numero arbitrario di uffici. “Ufficio” e “Piano” sono entità deboli perché per essere identificati hanno bisogno di sapere a quale edificio appartengono, e da questo derivano le due entità deboli a cascata. Il piano poi è identificato anche dal suo numero, mentre l’ufficio dal nome che lo identifica rispetto agli altri presenti allo stesso piano.

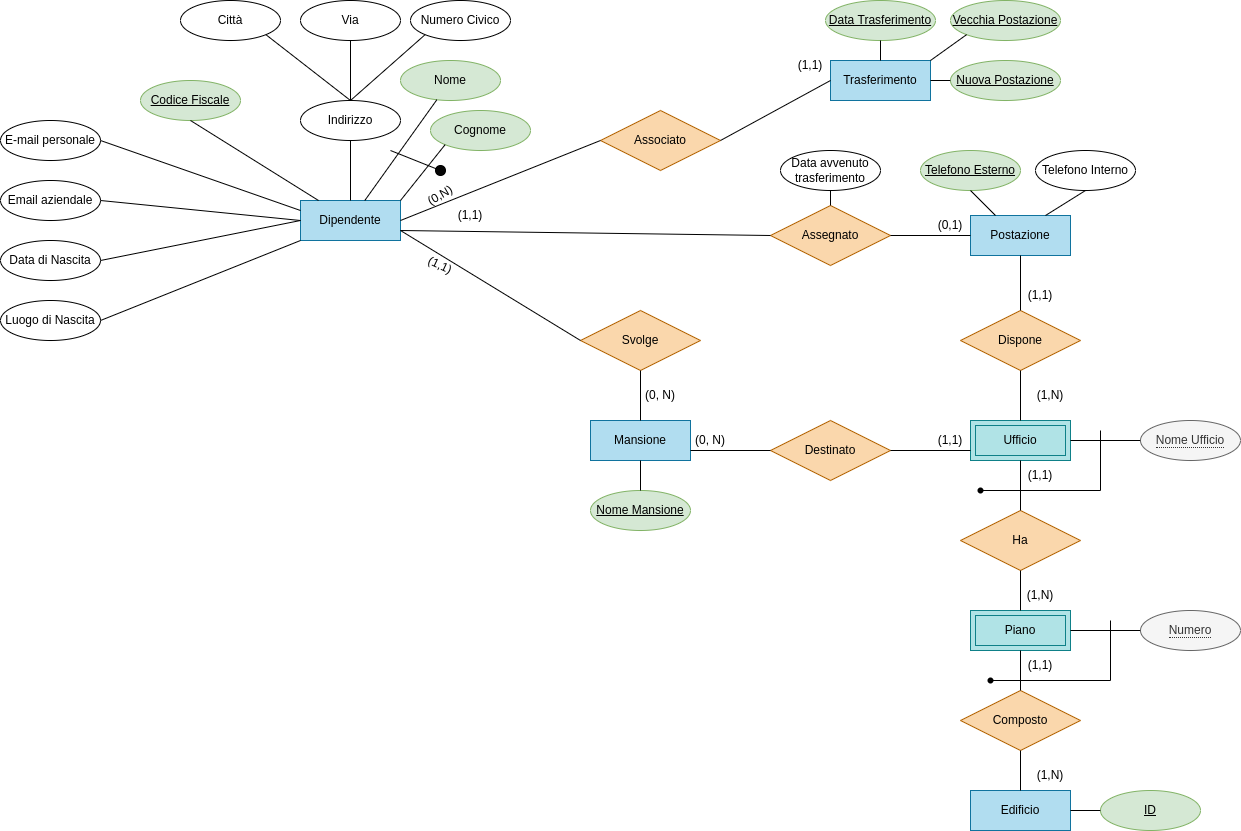
La postazione viene identificata tramite il numero di telefono esterno ad essa associata, perché è univoco dovendo essere il modo per mettersi in contatto da altri uffici con quella specifica postazione, a differenza del numero di telefono interno, usato per le comunicazioni tra postazioni dello stesso ufficio, che potrebbe ripetersi in postazioni di uffici differenti.

A una postazione è associato un Dipendente (di cui per compattezza sono omessi gli attributi) con cardinalità (0,1), perché la postazione può essere associata a un dipendente oppure essere momentaneamente libera. Il dipendente invece deve essere sempre associato a una postazione (poiché se non ne dispone non può lavorare). L’associazione tra di essi ha anche un attributo (“Data avvenuto trasferimento”) che tiene traccia di quando il dipendente è stato trasferito presso la postazione che occupa. La postazione inoltre è anche associata con cardinalità (1,1) a un ufficio, perché una postazione può e deve appartenere ad un singolo ufficio. L’ufficio (di cui sono omessi gli attributi per compattezza) può aggregare a sé una o più postazioni.

### Integrazione finale

Nell’integrazione finale, per tenere traccia dello storico dei trasferimenti di un dipendente, viene inserita l’entità “Trasferimento”, che tramite i suoi attributi tiene traccia, per l’appunto, di un singolo trasferimento di uno specifico dipendente. Ad ogni dipendente sono quindi associate tante entità “Trasferimento” quanti sono stati i suoi trasferimenti da una postazione all’altra all’interno dell’azienda negli ultimi tre anni (questo è il periodo di interesse per la Directory Aziendale).

Inoltre, il dipendente viene anche associato a una specifica mansione, che è quella che gli viene assegnata e che può essere modificata da un dipendente del Settore Amministrativo. La mansione è associata a uno o più uffici, ma un ufficio può essere adibito a una sola mansione, come da specifiche.   
Al dipendente viene anche assegnato l’attributo “email aziendale”, che rappresenta quello che la specifica indica come “indirizzo e-mail […] associato all’ufficio a cui fa capo”, che avrà nel dominio dell’email il nome dell’ufficio a cui è associato attualmente il dipendente.

Per evitare conflitti e ambiguità sugli attributi denominati “Nome” di Dipendente, Mansione e Ufficio, gli ultimi due vengono rinominati in “Nome Mansione” e “Nome Ufficio”.  
Per le stesse ragioni l’associazione tra Dipendente e Postazione viene rinominata in “Assegnato”.

## Regole aziendali

## RA 1) A un Dipendente deve essere associata una nuova entità Trasferimento, ogni volta che viene eseguita l’operazione di trasferimento su di esso, e solo in quel caso.

RA 2) Un’entità “Trasferimento” deve essere eliminata se rappresenta un trasferimento avvenuto più di tre anni fa.

## Dizionario dei dati

|  |  |  |  |
| --- | --- | --- | --- |
| **Entità** | **Descrizione** | **Attributi** | **Identificatori** |
| Dipendente | Dipendente dell’azienda | Codice Fiscale, Indirizzo (Città, Via, Numero Civico), Nome, Cognome, E-mail personale, E-mail aziendale, Data di nascita, Luogo di nascita | Codice Fiscale, Nome e Cognome |
| Trasferimento | Rappresenta un trasferimento di un dipendente, dalla sua vecchia postazione a quella che gli viene assegnata dal settore spazi | Data Trasferimento, Vecchia Postazione, Nuova Postazione | Data Trasferimento, Vecchia Postazione, Nuova Postazione |
| Postazione | Rappresenta una postazione di un determinato ufficio | Telefono Esterno, Telefono Interno | Telefono Esterno |
| Mansione | Rappresenta la mansione svolta da un determinato Dipendente o associata ad un ufficio | Nome Mansione | Nome Mansione |
| Edificio | Rappresenta un edificio utilizzato dall’azienda | ID | ID |
| Piano | Rappresenta il piano di un edificio | Numero | Numero, Edificio |
| Ufficio | Rappresenta un ufficio dell’azienda | Nome Ufficio | Nome Ufficio, Piano |

# Progettazione logica

## Volume dei dati

Si considera un’azienda con 1000 dipendenti. Si considera che la base di dati si stabilizzi una volta raggiunto questo numero di dipendenti. Un dipendente deve essere trasferito dopo aver trascorso più di 6 mesi nella stessa postazione. Inoltre consideriamo stabile anche il numero di entità “Trasferimento”, poiché sebbene ne vengano aggiunte di nuove per ogni trasferimento di un dipendente, quelle che rappresentano un trasferimento più vecchio di tre anni vengono eliminate, e perciò possiamo approssimare il numero di queste entità (dopo che la base di dati sarà a regime) un numero costante.

|  |  |  |
| --- | --- | --- |
| **Concetto nello schema** | **Tipo[[1]](#footnote-2)** | **Volume atteso** |
| Dipendente | E | 1000 |
| Trasferimento | E | 6000 (6\*dipendente) |
| Postazioni | E | 1100 |
| Mansioni | E | 20 |
| Edificio | E | 3 |
| Piano | E | 12 |
| Uffici | E | 120 |
| Associato | R | 6000 |
| Assegnato | R | 1000 |
| Svolge | R | 1000 |
| Dispone | R | 1100 |
| Destinato | R | 120 |
| Ha | R | 120 |
| Composto | R | 12 |

## Tavola delle operazioni

|  |  |  |
| --- | --- | --- |
| **Cod.** | **Descrizione** | **Frequenza attesa** |
| DT | Report del sistema dei dipendenti da trasferire. | 1 volta a settimana |
| DTM | Report dei dipendenti da trasferire, raggruppati per mansione | 1 volta a settimana |
| T | Trasferimento di un dipendente | 20 volte a settimana |
| MM | Modifica mansione di un dipendente | 5 volte al mese |
| SD | Report dello storico degli uffici di un dipendente | 20 volte a settimana |
| RD | Ricerca di un dipendente per nome e/o cognome. | 30 volte al giorno |
| RP | Ricerca postazione tramite telefono esterno. | 30 volte al giorno |

## Costo delle operazioni

* DT (Report dipendenti da trasferire del Sistema)

|  |  |  |  |
| --- | --- | --- | --- |
| Dipendete | E | L | 1000 |
| Assegnato | R | L | 1000 |
| Postazione | E | L | 1000 |
| Dispone | R | L | 1000 |
| Ufficio | E | L | 120 |
| Destinato | R | L | 120 |
| Mansione | E | L | 20 |
| Svolge | R | L | 1000 |

Per capire se un dipendente va trasferito bisogna guardare due fattori: da quanto tempo è associato all’attuale postazione, oppure se la mansione ad esso assegnata coincide ancora con quella associata all’ufficio in cui si trova, nel caso in cui sia stata cambiata dal Settore Amministrativo, il dipendente va marcato come da trasferire. Bisogna quindi prima andare a controllare l’attributo “Data Avvenuto Trasferimento”, e poi confrontare la mansione svolta dal dipendente e quella associata all’ufficio in cui lavora.

Totale = 5260 a settimana

* DTM (Report dei dipendenti da trasferire divisi per mansione)

|  |  |  |  |
| --- | --- | --- | --- |
| Dipendete | E | L | 1000 |
| Assegnato | R | L | 1000 |
| Postazione | E | L | 1000 |
| Dispone | R | L | 1000 |
| Ufficio | E | L | 120 |
| Destinato | R | L | 120 |
| Mansione | E | L | 20 |
| Svolge | R | L | 1000 |

Totale = 5260 accessi a settimana

L’operazione svolge le stesse attività di DT ma mostra il risultato con i dipendenti raggruppati per mansione.

* T (Trasferimento di un dipendente)

|  |  |  |  |
| --- | --- | --- | --- |
| Dipendente | E | L | 1 |
| Associato | R | L | 6 |
| Trasferimento | E | L | 6 |
| Assegnato | R | L | 1 |
| Postazione | E | S | 2 |
| Dispone | R | L | 1 |
| Ufficio | E | L | 1 |
| Destinato | R | L | 1 |
| Mansione | E | L | 1 |
| Svolge | E | L | 1 |

Per effettuare un trasferimento devo leggere lo storico dei trasferimenti associati a quell’utente per controllare che non sta per essere assegnato a una postazione che ha già occupato negli ultimi tre anni, poi devo controllare la mansione associata all’ufficio della nuova postazione, e controllare che coincida con la mansione che è attualmente assoicata al dipendente di cui sto effettuando il trasferimento. A quel punto posso scrivere l’entità postazione con la nuova postazione da associare al dipendente.

Totale accessi = 21 a operazione → 420 a settimana

* MM (Modifica Mansione)

|  |  |  |  |
| --- | --- | --- | --- |
| Dipendente | E | L | 1 |
| Svolge | R | L | 1 |
| Mansione | E | S | 2 |

Per cambiarla basta identificare il dipendente e accedere in scrittura all’entità “Mansione”. Questa operazione comporta che (fino al prossimo trasferimento) la mansione assegnata a un dipendente e quella associata all’ufficio in cui lavora siano differenti, ma questo è consentito perché sta a segnalare che a un dipendente è stata assegnata una nuova mansione da svolgere, e che quindi deve essere trasferito (e infatti comparirà nei report dei dipendenti da trasferire).

Totale = 4 accessi per operazione → 20 accessi al mese (quindi in media 5 a settimana).

* SD (Report storico trasferimenti di un dipendente)

|  |  |  |  |
| --- | --- | --- | --- |
| Dipendente | E | L | 1 |
| Associato | R | L | 6 |
| Trasferimento | E | L | 6 |

Per ogni dipendente basta accedere in lettura alle entità Trasferimento a lui associate (che in media saranno 6, poiché si conservano per non più di tre anni, e si ha (senza contare i cambi di mansione, che però sono poco frequenti) un trasferimento dopo 6 mesi, quindi 2 all’anno, quindi 6 in 3 anni).

Totale = 13 accessi ad operazione, 20 operazioni a settimana → 260 accessi a settimana

* RD (Ricerca Dipendente)

|  |  |  |  |
| --- | --- | --- | --- |
| Dipendente | E | L | 1 |
| Assegnato | R | L | 1 |
| Postazione | E | L | 1 |
| Dispone | R | L | 1 |
| Ufficio | E | L | 1 |
| Ha | R | L | 1 |
| Piano | E | L | 1 |
| Composto | R | L | 1 |
| Edificio | E | L | 1 |

Per cercare un dipendente abbiamo bisogno della sua entità (per accedere ai suoi attributi), e quelli della postazione, del piano e dell’edificio per identificare l’ufficio in cui lavora.

Totale = 9 accessi ad operazione, 30 operazioni al giorno → 270 accessi al giorno → 1890 accessi a settimana. L’operazione da questo punto di vista è la stessa sia che la ricerca sia per nome, sia che sia per cognome, sia che usi entrambi.

* RP (Ricerca Postazione)

|  |  |  |  |
| --- | --- | --- | --- |
| Postazione | E | L | 1 |
| Assegnato | R | L | 1 |
| Dipendente | E | L | 1 |
| Dispone | R | L | 1 |
| Ufficio | E | L | 1 |
| Ha | R | L | 1 |
| Piano | E | L | 1 |
| Composto | R | L | 1 |
| Edificio | E | L | 1 |
| Svolge | R | L | 1 |
| Mansione | E | L | 2 |
| Destinato | R | L | 1 |

La ricerca della postazione accede sia alla postazione, che al dipendente che può essere associato ad essa (nel calcolo degli accessi consideriamo il caso peggiore/più costoso, ovvero quello in cui alla postazione sia effettivamente assegnato un dipendente). Poi deve riportare i dati sulla sua ubicazione, accedendo quindi in lettura al piano e all’edificio a cui è associata, oltre che alla mansione a cui è associato l’ufficio in cui si trova. Inoltre, viene richiesto di mostrare se il dipendente è in procinto di essere trasferito oppure no, quindi bisogna accedere nuovamente alla mansione del dipendente, per controllare che sia la stessa che è associata all’ufficio, e accedere al dato “Data Avvenuto Trasferimento” per controllare da quanto tempo il dipendente occupa quella mansione. Per ricavare quest’ultimo dato non abbiamo bisogno di ulteriori accessi perché lo otteniamo quando accediamo in lettura per la prima volta alla relazione “Assegnato”.

Totale = 13 accesi per operazione, 30 operazioni al giorno → 390 accessi al giorno → 2730 accessi a settimana.

## Ristrutturazione dello schema E-R

1. L’attributo “Data Avvenuto Trasferimento” della relazione “Assegnato” è ridondante all’attributo “Data Trasferimento” dell’ultima entità “Trasferimento” associata al dipendente.

È però conveniente mantenerlo, perché nelle operazioni DT e DTM (le più costose su base settimanale) questo attributo viene sfruttato per determinare da quanto tempo un dipendente è assegnato alla sua postazione, e tramite questo stabilire se va trasferito o meno.

Se non ci fosse, bisognerebbe ricercare per ogni dipendente l’ultimo trasferimento tra le entità trasferimento ad esso associate, aumentando ulteriormente il costo dell’operazione.

Per ogni dipendente, nel caso peggiore, bisognerebbe effettuare 6 ulteriori accessi in lettura per cercare l’ultimo trasferimento. Effettuandolo per tutti i dipendenti otterremo 6000 accessi in lettura in più. Se consideriamo che il tipo di dato che memorizza la data di “Data Avvenuto Trasferimento” occupi 3 bytes, avremmo un carico aggiuntivo di circa 3 Kb. Viene pertanto mantenuta la ridondanza.

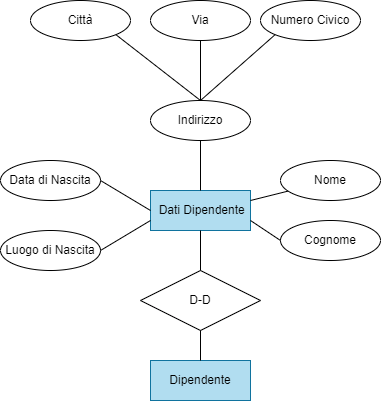
L’entità “Mansione” è raggiungibile sia dal “Dipendente”, sia da “Ufficio”. È però importante notare che non sono legate fra di loro, e questa in realtà non rappresenta una ridondanza, ne una dipendenza della mansione del dipendente da quella dell’ufficio, poiché rappresentano cose concettualmente diverse. In alcune situazioni infatti la mansione assegnata al dipendente e quella associata all’ufficio in cui lavora possono non coincidere (per esempio nel caso in cui la mansione del dipendente sia stata modificata dal Settore Amministrativo, ma il dipendente in questione non sia ancora stato trasferito in una postazione di un ufficio associato alla sua nuova mansione).

3) Per l’identificazione del dipendente viene scelto il codice fiscale, poiché identifica in maniera univoca un dipendente, ed è un attributo singolo, il che permette di avere tabelle più compatte, piuttosto che scegliere la coppia (Nome, Cognome). Per le altre entità vengono mantenuti gli identificatori scelti nel diagramma E-R, poiché non c’erano più opzioni tra le quali scegliere.

|  |  |
| --- | --- |
| **Entità** | **Identificatori** |
| Dipendente | Codice Fiscale |
| Trasferimento | Data Trasferimento, Vecchia Postazione, Nuova Postazione |
| Postazione | Telefono Esterno |
| Mansione | Nome Mansione |
| Edificio | ID |
| Piano | Numero, Edificio |
| Ufficio | Nome Ufficio, Piano |

L’ID dell’edificio viene inteso come una stringa (es.: il nome assegnato all’edificio) che lo identifichi (ovviamente in maniera univoca) rispetto agli altri.

## Trasformazione di attributi e identificatori

Poiché l’entità dipendente ha un numero molto elevato di attributi, che genererebbero poi una tabella con un numero molto alto di colonne, si può optare per un partizionamento verticale dell’entità, aggiungendo un’entità collegata “Dati Dipendente”, che contiene l’indirizzo, il luogo e la data di nascita, il nome e il cognome. (Nella figura sono omessi i rimanenti attributi di dipendente).

(1,1)

(1,1)

## Traduzione di entità e associazioni

DIPENDENTE (Codice Fiscale, Telefono Esterno, Data avvenuto trasferimento, Mansione)

* Vincolo Integrità Relazionale tra Telefono Esterno e Postazione (Telefono Estero)
* Vincolo Integrità Relazionale tra Mansione e Mansione (Nome Mansione)

DATI DIPENDENTE (CF Dipendente, Nome, Cognome, Data di Nascita, Luogo di Nascita, Città, Via, Numero Civico, Email Personale, Email Aziendale)

* Vincolo Integrità Relazionale tra CF Dipendente e Dipendente (Codice Fiscale)

MANSIONE (Nome Mansione)

EDIFICIO (ID)

POSTAZIONE (Telefono Esterno, Telefono Interno, Nome Ufficio, Numero Piano, ID Edificio)

* Vincolo Integrità Relazionale tra Nome Ufficio e Ufficio (Nome Ufficio)
* Vincolo Integrità Relazionale tra Numero Piano e Piano (Numero)
* Vincolo Integrità Relazionale tra ID Edificio e Edificio (ID)

UFFICIO (Nome Ufficio, Numero Piano, ID Edificio, Mansione Assegnata)

* Vincolo Integrità Relazionale tra Numero Piano e Piano (Numero)
* Vincolo Integrità Relazionale tra ID Edificio e Edificio (ID)
* Vincolo Integrità Relazionale tra Mansione Assegnata e Mansione (Nome Mansione)

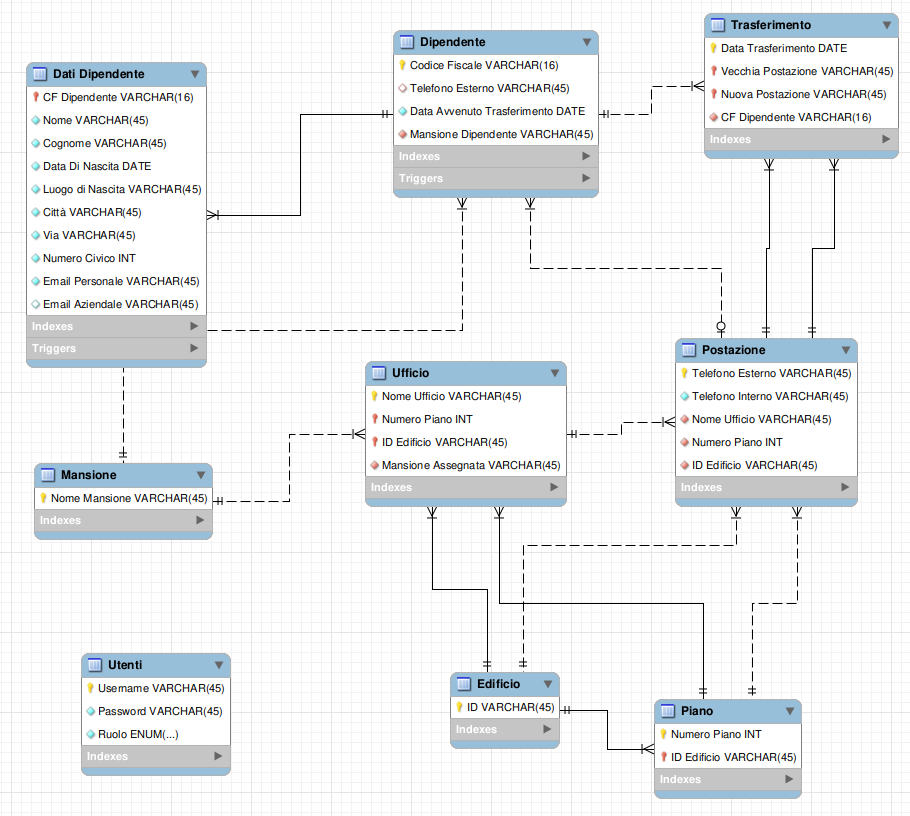
PIANO (Numero Piano, ID Edificio)

* Vincolo Integrità Relazionale tra ID Edificio e Edificio (ID)

TRASFERIMENTO (Data Trasferimento, Vecchia Postazione, Nuova Postazione, CF Dipendente)

* Vincolo Integrità Relazionale tra CF Dipendente e Dipendente (Codice Fiscale)
* Vincolo di Integrità Relazionale tra Vecchia Postazione e Postazione (Telefono Esterno)
* Vincolo di Integrità Relazionale tra Nuova Postazione e Postazione (Telefono Esterno)

Rappresentazione grafica del modello relazionale:



## Normalizzazione del modello relazionale

Effettuare la normalizzazione del modello relazionale precedentemente descritto (in forma grafica) andando a mostrare le forme 1NF, 2NF, 3NF.

* 1NF: Il modello relazionale è in 1NF perché ogni tabella ha una chiave primaria, e di conseguenza non ci possono essere tuple duplicate. Inoltre, non ci sono colonne divisibili (ovvero dove abbiamo un attributo composto), poiché l’indirizzo del dipendente è stato scomposto in tre colonne (città, via e numero civico), mentre l’ID Edificio non rappresenta un edificio, bensì il nome che identidica l’edificio (es.: Edificio Amministrazione, Magazzino, Edificio A, ecc…).
* 2NF: Il modello è in 2NF perché tutti i valori che non sono la chiave dipendono dall’intera chiave: in “Dati Dipendente” tutti i valori dipendono dal dipendente a cui si riferiscono, e quindi dal codice fiscale del dipendente che li identifica. Lo stesso discorso vale per “Dipendente”. Nella tabella “Trasferimento”, il codice fiscale del dipendente dipende da tutti e tre gli attributi della chiave, poiché se dipendesse solamente dalla Data del trasferimento, ci potrebbe essere un solo trasferimento per ogni data disponibile, se dipendesse solamente dalla data e dalla vecchia postazione, e quindi questa fosse la chiave minima, non potremmo avere che due dipendenti diversi vengano trasferiti dalla stessa postazione lo stesso giorno (scenario poco probabile ma non impossibile, nel caso in cui un dipendente venga trasferito presso una postazione, ma gli venga poi cambiata la mansione e necessiti di un nuovo ricollocamento il giorno stesso: non si potrebbe in questo caso inserire il trasferimento se la postazione fosse stata già occupata da un dipendente che il giorno stesso è stato trasferito (o scambiato con il dipendente in questione). Discorso simile se il codice fiscale dipendesse solamente dalla data e dalla nuova postazione. L’unica chiave minima possibile è quindi l’unione di questi tre attributi, e il codice fiscale dipende da tutti e tre.  
  La tabella postazione ha un solo attributo chiave, e tutti gli altri valori dipendono di conseguenza da essa. Per la tabella ufficio, l’attributo “Mansione Assegnata” dipende da tutta la chiave: se infatti dipendesse solo da Nome Ufficio e Piano, non ci potrebbero essere due uffici allo stesso piano (ma in edifici diversi) che svolgono la stessa mansione. Discorso analogo se la mansione dipendesse solo da Piano e ID Edificio, perché non ci potrebbero essere due uffici allo stesso piano che svolgono la stessa mansione. L’ultima combinazione possibile sarebbe Nome Ufficio e ID Edificio, ma così non potremmo avere Uffici a piani diversi dello stesso edificio che svolgono la stessa mansione.  
  Di conseguenza la mansione assegnata a un ufficio dipende da tutta la chiave.
* 3NF: Non ci sono dipendenze di tipo transitivo, poiché nessun attributo delle tabelle dipende da un attributo che non sia la chiave, ma tutte le dipendenze sono nella forma X → Y dove X è sempre parte della chiave primaria. Anche in Dipendente, dove potrebbe sembrare che la mansione dipenda dalla postazione, e non dal dipendente, non è in realtà così. La “Mansione Dipendente” dipende esclusivamente dal dipendente, e non dalla postazione (e quindi dall’ufficio). Perché è vero che un dipendente deve essere assegnato, al momento del trasferimento o dell’inserimento, a una postazione che si trovi in un ufficio adibito alla mansione da cui è assegnato, ma è anche vero che questo attributo, all’interno della tabelloa Dipendente, ha lo scopo di rappresentare la mansione che viene assegnata al dipendente dal Settore Amministrativo. Il Settore Amministrativo inoltre ha anche i privilegi necessari per poter modificare la mansione che viene assegnata al dipendente in ogni momento. È quindi possibile che la mansione assegnata al dipendente differisca da quella dell’ufficio in cui si trova. È quindi evidente che la Mansione Dipendente non dipenda dalla postazione.
* BCNF:   
  Per la tabella “Dati Dipendenti”: CF → Nome, CF → Cognome, CF → Data Di Nascita, CF →Luogo di Nascita, CF →Città, CF →Via, CF →Numero Civico, CF → Email Personale, CF → Email Aziendale\*.  
    
  Per la tabella “Dipendenti” tutti gli attributi dipendono dalla chiave.  
  Per la tabella trasferimento, come già visto l’attributo CF Dipendente dipende dall’intera chiave.  
  Ragionamento analogo per “Ufficio” e per “Postazione”.

\*La mail aziendale dipende anche dalla mansione associata all’ufficio in cui lavora il dipendente, essendo del tipo nome.cognome[numeroRandomico]@nomeufficio.it, ma questa cosa verrà gestita in maniera “trasparente” da un trigger, perciò consideriamo l’email aziendale come un altro degli attributi del dipendente da esso dipendenti.

# Progettazione fisica

## Utenti e privilegi

## All’interno del database sono presenti 4 utenti. Il primo è l’utente di login, che viene “nascosto” all’utente perché utilizzato per consentire il login all’interno degli altri tre utenti. Questo utente infatti ha accesso solamente all’esecuzione della stored procedure che viene utilizzata per il sistema di login.

L’utente “Dipendente” rappresenta l’utente standard, che ha accesso all’esecuzione delle stored procedures RicercaDipendenteNome, RicercaDipendenteCognome, RicercaDipNomeCognome e RicercaPostazione. Queste rappresentano le operazioni base a cui, come richiesto dalla specifica, ogni dipendente della Directory Aziendale deve avere accesso.

L’utente “DipendenteSettoreSpazi” ha accesso a tutte le stored procedure del “Dipendente”, e in più può richiedere l’esecuzione della stored procedure Trasferimento, per effettuare il trasferimento di un dipendente ha un’altra postazione. Può eseguire le stored procedures DipendentiTrasferireMansione e ReportDipendentiTrasferire, che consentono di mostrare quali dipendenti devono essere trasferiti (suddivisi per mansione nella prima, in ordine alfabetico nella seconda). Inoltre, può richiedere l’esecuzione della stored procedure ScambioPostazione (per scambiare la postazione di due dipendenti), e alle stored procedures InserisciDipendente e EliminaDipendente (per aggiungere/rimuovere un dipendente al/dal sistema).

L’utente “DipendenteSettoreAmministrativo”, da specifica, ha la necessità di poter cambiare la mansione di un dipendente. Per questo gli viene consentito l’accesso alla stored procedure ModificaMansione. Un’altra privilegio esclusivo di questo utente è quello di stampare il record degli uffici occupati da un dipendente negli ultimi tre anni, attraverso l’esecuzione della stored procedure StoricoUffici. Anche questo utente ha, inoltre, accesso a tutte le stored procedures del Dipendente.

Tutti questi utenti hanno accesso solamente all’esecuzione di determinate stored procedures, mentre nessuno di questi ha accesso in lettura o scrittura direttamente alle tabelle.

## Strutture di memorizzazione

|  |  |  |
| --- | --- | --- |
| **Tabella Dipendente** | | |
| **Colonna** | **Tipo di dato** | **Attributi[[2]](#footnote-3)** |
| **Codice Fiscale** | VARCHAR(16) | PK, NN |
| **Telefono Esterno** | VARCHAR(45) | UQ |
| **Data Avvenuto Trasferimento** | DATE | NN |
| **Mansione Dipendente** | VARCHAR(45) | NN |

|  |  |  |
| --- | --- | --- |
| **Tabella Dati Dipendente** | | |
| **Colonna** | **Tipo di dato** | **Attributi[[3]](#footnote-4)** |
| **CF Dipendente** | VARCHAR(16) | PK, NN |
| **Nome** | VARCHAR(45) | NN |
| **Cognome** | VARCHAR(45) | NN |
| **Data di Nascita** | DATE | NN |
| **Luogo di Nascita** | VARCHAR(45) | NN |
| **Città** | VARCHAR(45) | NN |
| **Via** | VARCHAR(45) | NN |
| **Numero Civico** | INT | NN |
| **Email Personale** | VARCHAR(45) | NN |
| **Email Aziendale** | VARCHAR(45) | NN |

|  |  |  |
| --- | --- | --- |
| **Tabella Trasferimento** | | |
| **Colonna** | **Tipo di dato** | **Attributi** |
| **Data Trasferimento** | DATE | PK, NN |
| **Vecchia Postazione** | VARCHAR(45) | PK, NN |
| **Nuova Postazione** | VARCHAR(45) | PK, NN |
| **CF Dipendente** | VARCHAR(16) | NN |

|  |  |  |
| --- | --- | --- |
| **Tabella Postazione** | | |
| **Colonna** | **Tipo di dato** | **Attributi** |
| **Telefono Esterno** | VARCHAR(45) | PK, NN |
| **Telefono Interno** | VARCHAR(45) | NN |
| **Nome Ufficio** | VARCHAR(45) | NN |
| **Numero Piano** | INT | NN |
| **ID Edificio** | VARCHAR(45) | NN |

|  |  |  |
| --- | --- | --- |
| **Tabella Ufficio** | | |
| **Colonna** | **Tipo di dato** | **Attributi** |
| **Nome Ufficio** | VARCHAR(45) | PK, NN |
| **Numero Piano** | INT | PK, NN |
| **ID Edificio** | VARCHAR(45) | PK, NN |
| **Mansione Assegnata** | VARCHAR(45) | NN |

|  |  |  |
| --- | --- | --- |
| **Tabella Mansione** | | |
| **Colonna** | **Tipo di dato** | **Attributi** |
| **Nome Mansione** | VARCHAR(45) | PK |

|  |  |  |
| --- | --- | --- |
| **Tabella Piano** | | |
| **Colonna** | **Tipo di dato** | **Attributi** |
| **Numero Piano** | INT | PK |
| **ID Edificio** | VARCHAR(45) | PK |

|  |  |  |
| --- | --- | --- |
| **Tabella Edificio** | | |
| **Colonna** | **Tipo di dato** | **Attributi** |
| **ID Edificio** | VARCHAR(45) | PK |

|  |  |  |
| --- | --- | --- |
| **Tabella Utenti** | | |
| **Colonna** | **Tipo di dato** | **Attributi[[4]](#footnote-5)** |
| **Username** | VARCHAR(45) | PK |
| **Password** | VARCHAR(45) | NN |
| **Ruolo** | ENUM | NN |

## Indici

## Non sono presenti indici aggiuntivi oltre quelli generati autonomamente dal DBMS.

## Trigger

CREATE TRIGGER `Directory\_Aziendale`.`CreaEmailAziendale` BEFORE INSERT ON `Dati Dipendente` FOR EACH ROW

BEGIN

declare var\_dominioEmail varchar(45);

declare var\_email varchar(45);

SELECT P.`Nome Ufficio`

FROM `Dipendente` D join `Postazione` P on P.`Telefono Esterno` = D.`Telefono Esterno`

WHERE D.`Codice Fiscale` = NEW.`CF Dipendente` INTO var\_dominioEmail;

/\* Genera un email del tipo nome.cognome00@nomeufficio.it dove '00' è un numero di due cifre randomico, per gestire

eventuali omonimie \*/

SELECT lower(concat(NEW.`Nome`, '.', NEW.`Cognome`,FLOOR(RAND()\*(99-1)+1),

'@', replace(var\_dominioEmail, ' ', ''), '.it')) into var\_email;

SET NEW.`Email Aziendale` = var\_email;

END

## Il trigger di cui sopra il codice provvede a generare l’email aziendale quando viene inserito un nuovo dipendente. Il numero randomico viene utilizzato per quei casi di omonimia dove due dipendenti dallo stesso nome e che lavorano nello stesso ufficio risulterebbero avere la stessa email. Il numero randomico generato all’interno dell’indirizzo email riduce notevolmente questo rischio.

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

CREATE TRIGGER `Directory\_Aziendale`.`CompatibiltaMansione` BEFORE INSERT ON `Dipendente` FOR EACH ROW

BEGIN

declare var\_MansioneUfficio varchar(45);

select `Mansione Assegnata`

from `Postazione` P join `Ufficio` U on P.`Nome Ufficio` = U.`Nome Ufficio` and P.`Numero Piano` = U.`Numero Piano`

and P.`ID Edificio` = U.`ID Edificio`

where P.`Telefono Esterno` = NEW.`Telefono Esterno`

into var\_MansioneUfficio;

if (var\_MansioneUfficio <> NEW.`Mansione Dipendente`) then

signal sqlstate '45002' set message\_text = 'Mansione nuovo dipendente incompatibile con ufficio';

end if;

END

Questo trigger controlla se la mansione del dipendente che sta per essere inserito è compatibile con quella dell’ufficio a cui sta per essere associato.  
-------------------------------------------------------------------------------------------------------------------------

CREATE TRIGGER `Directory\_Aziendale`.`CheckMansione` BEFORE UPDATE

ON `Dipendente` FOR EACH ROW

BEGIN

if (NEW.`Mansione Dipendente` <> (SELECT `Mansione Assegnata` FROM Postazione P join Ufficio U on

U.`Nome Ufficio` = P.`Nome Ufficio` and U.`Numero Piano` = P.`Numero Piano` and U.`ID Edificio` = P.`ID Edificio`

WHERE P.`Telefono Esterno` = NEW.`Telefono Esterno`) AND NEW.`Telefono Esterno` <> OLD.`Telefono Esterno`) then

signal sqlstate '45001' set message\_text = 'Mansione del nuovo ufficio incompatibile con mansione dipendente';

end if;

END

## Questo trigger controlla se, quando un dipendente viene trasferito, la sua mansione e la nuova postazione sono compatibili.

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

CREATE TRIGGER `Directory\_Aziendale`.`CheckStorico` BEFORE UPDATE

ON `Dipendente` FOR EACH ROW

BEGIN

declare var\_todayDate datetime;

SELECT curdate() into var\_todayDate;

if (NEW.`Telefono Esterno` <> OLD.`Telefono Esterno` AND NEW.`Telefono Esterno` in (SELECT `Vecchia Postazione` FROM Trasferimento T join Dipendente D

on D.`Codice Fiscale` = T.`CF Dipendente`) ) then

signal sqlstate '45002' set message\_text = 'Dipendente già assegnato a questa postazione negli ultimi 3 anni ';

end if;

END

Questo trigger controlla lo storico delle postazione del dipendente per vedere se ha già occupato la postazione a cui sta per essere assegnato.

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

CREATE TRIGGER `Directory\_Aziendale`.`CreaTrasferimento` AFTER UPDATE

ON `Dipendente` FOR EACH ROW

BEGIN

if(NEW.`Telefono Esterno` <> OLD.`Telefono Esterno`) then

INSERT INTO `Trasferimento` (`Data Trasferimento`, `Vecchia Postazione`, `Nuova Postazione`, `CF Dipendente`)

VALUES (curdate(), OLD.`Telefono Esterno` , NEW.`Telefono Esterno`, NEW.`Codice Fiscale`);

end if;

END

Questo trigger, dopo che un dipendente è stato trasferito con successo, genera la tupla relativa al suo trasferimento.

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

CREATE TRIGGER `Directory\_Aziendale`.`RevisioneTrasferimenti` AFTER UPDATE

ON `Dipendente` FOR EACH ROW

BEGIN

/\* Setto momentaneamente i safe updates a 0 perchè così posso eseguire delle delete senza dover avere una "where" con

un riferimento a una primary key, poiché non mi interessa a chi appartengono i dati della tupla, se appartiene a un

trasferimento di più di 3 anni fa, va comunque eliminata. \*/

SET SQL\_SAFE\_UPDATES = 0;

-- Valutare cursore in next version

DELETE FROM `Trasferimento`

WHERE DATEDIFF(CURDATE(), `Trasferimento`.`Data Trasferimento`) > 1095; /\* Se sono passati più di 3 anni

(ovvero 1095 giorni) il record del trasferimento non serve più, e viene eiliminato \*/

SET SQL\_SAFE\_UPDATES = 1;

END

Questo trigger controlla dopo ogni update di una tupla della tabella dipendenti se nello storico dei trasferimenti ce n’è qualcuno che risale a più di tre anni fa. In questo caso preovvede ad eliminarlo perché non è più utile alla Directory Aziendale.

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

CREATE TRIGGER `UpdateEmailAziendale` AFTER UPDATE ON `Dipendente` FOR EACH ROW

BEGIN

declare var\_dominioEmail varchar(45);

declare var\_nome varchar(45);

declare var\_cognome varchar(45);

if (NEW.`Telefono Esterno` <> OLD.`Telefono Esterno`) then

SELECT P.`Nome Ufficio`

FROM `Dipendente` D join `Postazione` P on P.`Telefono Esterno` = D.`Telefono Esterno`

WHERE D.`Codice Fiscale` = NEW.`Codice Fiscale` INTO var\_dominioEmail;

SELECT I.`Nome`, I.`Cognome`

FROM `Dipendente` D join `Dati Dipendente` I on I.`CF Dipendente` = D.`Codice FIscale`

WHERE D.`Codice Fiscale` = NEW.`Codice Fiscale`

INTO var\_nome, var\_cognome;

/\* Genera un email del tipo nome.cognome00@nomeufficio.it dove '00' è un numero di due cifre randomico, per gestire

eventuali omonimie \*/

UPDATE `Directory\_Aziendale`.`Dati Dipendente` SET `Email Aziendale` = lower(concat(var\_nome, '.', var\_cognome,

FLOOR(RAND()\*(99-1)+1), '@', replace(var\_dominioEmail, ' ', ''), '.it'))

WHERE (`CF Dipendente` = NEW.`Codice Fiscale`);

end if;

END

## Questo trigger modifica la mail aziendale dopo che un dipendente è stato trasferito con successo in una nuova postazione.

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

## Eventi

## Non sono presenti eventi.

## Viste

## Non sono presenti viste.

## Stored Procedures e transazioni

Di seguito le stored procedures. Per ogni stored procedure è presente un commento affianco al livello di isolamento scelto che motiva la scelta fatta.

1) INSERISCI DIPENDENTE

CREATE PROCEDURE `InserisciDipendente` (in var\_cf VARCHAR(45), in var\_telPostaz VARCHAR(45), in var\_dataTrasf DATE,

in var\_mansioneAssegnata VARCHAR(45), in var\_nome VARCHAR(45), in var\_cognome VARCHAR(45), in var\_dataNascita DATE,

in var\_luogoNascita VARCHAR(45), in var\_citta VARCHAR(45), in var\_via VARCHAR(45), in var\_civico INT, in var\_emailPersonale varchar(45))

BEGIN

declare var\_email\_aziendale varchar(45);

declare exit handler for sqlexception

begin

rollback;

resignal;

end;

set transaction isolation level repeatable read; /\* Abbiamo dei trigger che fanno dei controlli sui dati inseriti

prima di completare effettivamente l'inserimento. Vogliamo che finchè l'inserimento non sia stato validato le tuple che

ansiamo a leggere non possano essere modificate, per questo scegliamo repeatable read \*/

start transaction;

INSERT INTO `Directory\_Aziendale`.`Dipendente` (`Codice Fiscale`, `Telefono Esterno`, `Data Avvenuto Trasferimento`, `Mansione Dipendente`)

VALUES (var\_cf, var\_telPostaz, var\_dataTrasf, var\_mansioneAssegnata);

INSERT INTO `Directory\_Aziendale`.`Dati Dipendente` (`CF Dipendente`, `Nome`, `Cognome`, `Data Di Nascita`, `Luogo di Nascita`, `Città`, `Via`, `Numero Civico`, `Email Personale`)

VALUES (var\_cf, var\_nome, var\_cognome, var\_dataNascita, var\_luogoNascita, var\_citta, var\_via, var\_civico, var\_emailPersonale);

commit;

END

2) REPORT DIPENDENTI DA TRASFERIRE

CREATE PROCEDURE `ReportDipendentiTrasferire` ()

BEGIN

declare exit handler for sqlexception

begin

rollback;

resignal;

end;

set transaction read only;

set transaction isolation level repeatable read; /\* non vogliamo che qualcuno cambi la mansione di un dipendente finchè non abbiamo

finito di scandire tutte le tuple dei dipendenti, in modo che il report dei dipendenti da trasferire che otteniamo sia esatto

non mettiamo "serializable" perchè supponiamo che se anche anche viene inserito un nuovo dipendente mentre eseguiamo il report dei

dipendenti da trasferire, questo non sia da trasferire immediatamente, perchè sarà sicuramente nella sua postazione da meno di 6 mesi,

e la mansione ad esso assegnata dovrà per forza corrispondere a quella dell'ufficio in cui verrà collocato\*/

start transaction;

SELECT I.`Nome`, I.`Cognome`, D.`Mansione Dipendente`

FROM Dipendente D join Postazione P on D.`Telefono Esterno` = P.`Telefono Esterno` join Ufficio U on

U.`Nome Ufficio` = P.`Nome Ufficio` and U.`Numero Piano` = P.`Numero Piano` and U.`ID Edificio` = P.`ID Edificio` join

`Dati Dipendente` I on I.`CF Dipendente` = D.`Codice Fiscale`

WHERE DATEDIFF(CURRENT\_DATE() , D.`Data Avvenuto Trasferimento`) > 180 /\* L'intervallo dopo il quale un dipendente

deve essere trasferito è di 6 mesi, approssimato per semplicità a 180 giorni \*/

OR U.`Mansione Assegnata` <> D.`Mansione Dipendente`

/\* Se la mansione del dipendente è stata cambiata, come da specifica, questo deve risultare come da trasferire \*/

order by I.`Cognome`;

commit;

END

3) RICERCA DIPENDENTE PER NOME E COGNOME

CREATE PROCEDURE `RicercaDipNomeCognome` (in var\_nome VARCHAR(45), in var\_cognome VARCHAR(45))

BEGIN

declare exit handler for sqlexception

begin

rollback;

resignal;

end;

set transaction read only;

set transaction isolation level read committed; -- abbiamo solamente necessita di leggere una ad una le tuple. Non ci interessa dei

-- phantom read perchè non facciamo operazioni su gruppi di dipendenti.

start transaction;

SELECT `Nome`, `Cognome`, `Codice Fiscale`, `Email Personale`, `Email Aziendale`, P.`Telefono Esterno`, `Nome Ufficio`,

`Mansione Dipendente`, `Numero Piano`, `ID Edificio`

FROM `Dati Dipendente` I join `Dipendente` D on D.`Codice Fiscale` = I.`CF Dipendente`

join Postazione P on P.`Telefono Esterno` = D.`Telefono Esterno`

WHERE I.`Nome` = var\_nome and I.`Cognome` = var\_cognome;

commit;

END

4) STAMPA STORICO UFFICI

CREATE PROCEDURE `StoricoUffici`(in var\_codiceFiscale VARCHAR(16))

BEGIN

declare exit handler for sqlexception

begin

rollback;

resignal;

end;

set transaction read only;

set transaction isolation level read committed; /\* Dopbbiamo solamente leggere le tuple della tabella interessata, perciò

ci basta read committed \*/

start transaction;

SELECT U.`Nome Ufficio` AS 'Uffici Occupati', T.`Data Trasferimento` as 'Fino al Giorno'

FROM `Trasferimento` T join `Postazione` P on T.`Vecchia Postazione` = P.`Telefono Esterno`

join `Ufficio` U on P.`Nome Ufficio` = U.`Nome Ufficio`

WHERE T.`CF Dipendente` = var\_codiceFiscale

UNION

SELECT P.`Nome Ufficio`, 'Ufficio Corrente'

FROM `Dipendente` D join `Postazione` P on D.`Telefono Esterno` = P.`Telefono Esterno`

WHERE D.`Codice Fiscale` = var\_codiceFiscale;

commit;

END

5) DIPENDENTI DA TRASFERIRE DIVISI PER MANSIONE

CREATE PROCEDURE `DipendentiTrasferireMansione` ()

BEGIN

declare exit handler for sqlexception

begin

rollback;

resignal;

end;

set transaction read only;

set transaction isolation level repeatable read; /\* non vogliamo che qualcuno cambi la mansione di un dipendente finchè non abbiamo

finito di scandire tutte le tuple dei dipendenti, in modo che il report dei dipendenti da trasferire che otteniamo sia esatto

non mettiamo "serializable" perchè supponiamo che se anche anche viene inserito un nuovo dipendente mentre eseguiamo il report dei

dipendenti da trasferire, questo non sia da trasferire immediatamente, perchè sarà sicuramente nella sua postazione da meno di 6 mesi,

e la mansione ad esso assegnata dovrà per forza corrispondere a quella dell'ufficio in cui verrà collocato\*/

start transaction;

SELECT I.`Nome`, I.`Cognome`, D.`Mansione Dipendente`

FROM Dipendente D join Postazione P on D.`Telefono Esterno` = P.`Telefono Esterno` join Ufficio U

on U.`Nome Ufficio` = P.`Nome Ufficio` and U.`Numero Piano` = P.`Numero Piano` and U.`ID Edificio` = P.`ID Edificio` join

`Dati Dipendente` I on I.`CF Dipendente` = D.`Codice Fiscale`

WHERE DATEDIFF(CURRENT\_DATE() , D.`Data Avvenuto Trasferimento`) > 180 /\* L'intervallo dopo il quale un dipendente

deve essere trasferito è di 6 mesi, approssimato per semplicita a 180 giorni \*/

OR D.`Mansione Dipendente` <> U.`Mansione Assegnata`

GROUP BY D.`Mansione Dipendente`, I.`Nome`, I.`Cognome`;

commit;

END

6) LOGIN

CREATE PROCEDURE `login` (in var\_username varchar(45), in var\_pass varchar(45), out var\_role INT)

BEGIN

declare var\_user\_role ENUM('dipendente', 'dipendente settore spazi', 'dipendente settore amministrativo');

select `Ruolo` from `Utenti`

where `Username` = var\_username

and `Password` = md5(var\_pass)

into var\_user\_role;

-- See the corresponding enum in the client

if var\_user\_role = 'dipendente' then

set var\_role = 1;

elseif var\_user\_role = 'dipendente settore spazi' then

set var\_role = 2;

elseif var\_user\_role = 'dipendente settore amministrativo' then

set var\_role = 3;

else

set var\_role = 4;

end if;

END

7)RICERCA DIPENDENTE PER COGNOME

CREATE PROCEDURE `RicercaDipendenteCognome` (in var\_cognome VARCHAR(45))

BEGIN

declare exit handler for sqlexception

begin

rollback;

resignal;

end;

set transaction read only;

set transaction isolation level read committed;-- abbiamo solamente necessita di leggere una ad una le tuple. Non ci interessa dei

-- phantom read perchè non facciamo operazioni su gruppi di dipendenti.

start transaction;

SELECT `Nome`, `Cognome`,`Codice Fiscale` , `Email Personale`, `Email Aziendale`, D.`Telefono Esterno`, `Nome Ufficio`,

`Mansione Dipendente`, `Numero Piano`, `ID Edificio`

FROM `Dati Dipendente` I join `Dipendente` D on D.`Codice Fiscale` = I.`CF Dipendente`

join Postazione P on P.`Telefono Esterno` = D.`Telefono Esterno`

WHERE I.`Cognome` = var\_cognome;

commit;

END

8) RICERCA POSTAZIONE

CREATE PROCEDURE `RicercaPostazione` (in var\_telefonoEsterno VARCHAR(45))

BEGIN

declare var\_daTrasferire varchar(2);

declare var\_dipendente varchar(16);

declare exit handler for sqlexception

begin

rollback;

resignal;

end;

set transaction read only;

set transaction isolation level repeatable read; /\* Leggiamo più volte gli stessi dati per discriminare i casi in cui

la postazione è libera e quelli in cui invece è occupata (e in questo caso dobbiamo controllare se il dipendente sia da

trasferire o meno, pertanto vogliamo un lock sui dati che leggiamo per tutta la durata della transazione \*/

start transaction;

if (not exists (select `Codice Fiscale` FROM `Dipendente` WHERE `Telefono Esterno` = var\_telefonoEsterno)) then

SELECT P.`Nome Ufficio`, U.`Mansione Assegnata` as 'Mansione Ufficio', P.`Numero Piano`, P.`ID Edificio`, P.`Telefono Interno`, 'Si' as 'Postazione Libera'

FROM `Postazione` P join `Ufficio` U on P.`Nome Ufficio` = U.`Nome Ufficio` and P.`Numero Piano` = U.`Numero Piano` and

P.`ID Edificio` = U.`ID Edificio`

WHERE `Telefono Esterno` = var\_telefonoEsterno; -- caso postazione non occupata da nessun dipendente

else

select `Codice Fiscale` FROM `Dipendente` WHERE `Telefono Esterno` = var\_telefonoEsterno into var\_dipendente;

if(select var\_dipendente in (SELECT D.`Codice Fiscale`

FROM Dipendente D join Postazione P on D.`Telefono Esterno` = P.`Telefono Esterno` join Ufficio U on

U.`Nome Ufficio` = P.`Nome Ufficio` and U.`Numero Piano` = P.`Numero Piano` and U.`ID Edificio` = P.`ID Edificio`

WHERE DATEDIFF(CURRENT\_DATE() , D.`Data Avvenuto Trasferimento`) > 180 /\* L'intervallo dopo il quale un dipendente

deve essere trasferito è di 6 mesi, approssimato per semplicità a 180 giorni \*/

OR U.`Mansione Assegnata` <> D.`Mansione Dipendente`)) then

select 'Si' into var\_daTrasferire;

else

select 'No' into var\_daTrasferire; -- controllo se il dipendente assegnato alla postazione è da trasferire o meno

end if;

SELECT I.`Nome`, I.`Cognome`, var\_daTrasferire as 'Da Trasferire', I.`Email Personale`, I.`Email Aziendale`, D.`Telefono Esterno`,

P.`Nome Ufficio`, U.`Mansione Assegnata` as 'Mansione Ufficio', P.`Numero Piano`, P.`ID Edificio`, P.`Telefono Interno`

FROM `Dati Dipendente` I join `Dipendente` D on D.`Codice Fiscale` = I.`CF Dipendente`

join Postazione P on P.`Telefono Esterno` = D.`Telefono Esterno` join `Ufficio` U on P.`Nome Ufficio` = U.`Nome Ufficio` and P.

`Numero Piano` = U.`Numero Piano` and P.`ID Edificio` = U.`ID Edificio`

WHERE P.`Telefono Esterno` = var\_telefonoEsterno;

end if;

commit;

END

9) TRASFERIMENTO

CREATE PROCEDURE `Trasferimento` (in var\_cf VARCHAR(16), var\_nuovaPostazione VARCHAR(45))

BEGIN

declare exit handler for sqlexception

begin

rollback;

resignal;

end;

set transaction isolation level repeatable read; /\* Non voliamo che possa venir cambiata la mansione di un dipendente mentre

stiamo effettuando un trasferimento, che risulterebbe quindi poi scorretto. Per questo manteniamo il lock sulla tupla per tutta

la durata della transazione \*/

start transaction;

UPDATE `Dipendente` SET `Dipendente`.`Telefono Esterno` = var\_nuovaPostazione, `Data Avvenuto Trasferimento` = curdate()

WHERE `Codice Fiscale` = var\_cf;

commit;

END

10) EILIMINA DIPENDENTE

CREATE PROCEDURE `EliminaDipendente` (in var\_cf VARCHAR(16))

BEGIN

set transaction isolation level read committed; -- dobbiamo solo eliminare il dipendente, perciò è sufficente un livello più

-- basso di isolamento rispetto a quello standard

start transaction;

DELETE FROM `Dipendente` WHERE (`Codice Fiscale` = var\_cf);

/\* La tupla in `Dati Dipendente` viene eliminata "on cascade" \*/

commit;

END

11) MODIFICA MANSIONE

CREATE PROCEDURE `ModificaMansione` (in var\_cf VARCHAR(16), in var\_nuovaMansione VARCHAR(45))

BEGIN

declare exit handler for sqlexception

begin

rollback;

resignal;

end;

set transaction isolation level repeatable read; /\* Non vogliamo che qualcuno possa aggiornare la postazione di un dipendente mentre

stiamo ancora modificando la sua mansione perchè porterebbe il database in uno stato scorretto, perciò scegliamo una trasazione

repeatable read \*/

start transaction;

UPDATE `Dipendente` SET `Mansione Dipendente` = var\_nuovaMansione

WHERE (`Codice Fiscale` = var\_cf);

commit;

END

12) RICERCA DIPENDENTE PER NOME

CREATE PROCEDURE `RicercaDipendenteNome` (in var\_nome VARCHAR(45))

BEGIN

declare exit handler for sqlexception

begin

rollback;

resignal;

end;

set transaction read only;

set transaction isolation level read committed; -- come per le altre ricerche non dobbiamo rileggere gli stessi dati

-- quindi basta un livello di isolamento read committed

start transaction;

SELECT `Nome`, `Cognome`, `Codice Fiscale`, `Email Personale`, `Email Aziendale`, D.`Telefono Esterno`, `Nome Ufficio`,

`Mansione Dipendente`, `Numero Piano`, `ID Edificio`

FROM `Dati Dipendente` I join `Dipendente` D on D.`Codice Fiscale` = I.`CF Dipendente`

join Postazione P on P.`Telefono Esterno` = D.`Telefono Esterno`

WHERE I.`Nome` = var\_nome;

commit;

END

13) SCAMBIO POSTAZIONI

CREATE PROCEDURE `ScambioPostazione` (in var\_cf1 varchar(16), in var\_cf2 varchar(16))

BEGIN

declare var\_postD1 varchar(45);

declare var\_postD2 varchar (45);

declare exit handler for sqlexception

begin

rollback;

resignal;

end;

set transaction isolation level repeatable read; /\* I trigger devono controllare gli inserimenti, perciò vogliamo mantenere

il lock sulle tuple che leggiamo fino alla fine della transazione \*/

start transaction;

SELECT `Telefono Esterno` FROM `Dipendente` WHERE `Codice Fiscale` = var\_cf1 into var\_postD1;

UPDATE Dipendente SET `Telefono Esterno`= null WHERE `Codice Fiscale` = var\_cf1; -- valore temporaneo

SELECT `Telefono Esterno` FROM `Dipendente` WHERE `Codice Fiscale` = var\_cf2 into var\_postD2;

UPDATE Dipendente SET `Telefono Esterno`= var\_postD1 WHERE `Codice Fiscale` = var\_cf2;

UPDATE Dipendente SET `Telefono Esterno`= var\_postD2 WHERE `Codice Fiscale` = var\_cf1;

/\* Nel caso dello scambio di postazioni, il trigger non può creare la tupla relativa al trasferimento del dipendente

perché l'update avviene passando per il valore NULL, e quindi i valori OLD e NEW nel trigger update non catturano mai

i valori di due postazioni. Per questo la tupla per il dipendente che passa per il valore temporaneo null viene creata

manualmente qui \*/

INSERT INTO `Trasferimento` (`Data Trasferimento`, `Vecchia Postazione`, `Nuova Postazione`, `CF Dipendente`)

VALUES (curdate(), var\_postD1 , var\_postD2, var\_cf1);

commit;

END

# Appendice: Implementazione

## Codice SQL per instanziare il database

## -- MySQL Workbench Forward Engineering

SET @OLD\_UNIQUE\_CHECKS=@@UNIQUE\_CHECKS, UNIQUE\_CHECKS=0;

SET @OLD\_FOREIGN\_KEY\_CHECKS=@@FOREIGN\_KEY\_CHECKS, FOREIGN\_KEY\_CHECKS=0;

SET @OLD\_SQL\_MODE=@@SQL\_MODE, SQL\_MODE='ONLY\_FULL\_GROUP\_BY,STRICT\_TRANS\_TABLES,NO\_ZERO\_IN\_DATE,NO\_ZERO\_DATE,ERROR\_FOR\_DIVISION\_BY\_ZERO,NO\_ENGINE\_SUBSTITUTION';

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

-- Schema Directory\_Aziendale

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

DROP SCHEMA IF EXISTS `Directory\_Aziendale` ;

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

-- Schema Directory\_Aziendale

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

CREATE SCHEMA IF NOT EXISTS `Directory\_Aziendale` ;

USE `Directory\_Aziendale` ;

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

-- Table `Directory\_Aziendale`.`Edificio`

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

DROP TABLE IF EXISTS `Directory\_Aziendale`.`Edificio` ;

CREATE TABLE IF NOT EXISTS `Directory\_Aziendale`.`Edificio` (

`ID` VARCHAR(45) NOT NULL,

PRIMARY KEY (`ID`))

ENGINE = InnoDB

DEFAULT CHARACTER SET = utf8mb4

COLLATE = utf8mb4\_0900\_ai\_ci;

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

-- Table `Directory\_Aziendale`.`Mansione`

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

DROP TABLE IF EXISTS `Directory\_Aziendale`.`Mansione` ;

CREATE TABLE IF NOT EXISTS `Directory\_Aziendale`.`Mansione` (

`Nome Mansione` VARCHAR(45) NOT NULL,

PRIMARY KEY (`Nome Mansione`))

ENGINE = InnoDB

DEFAULT CHARACTER SET = utf8mb4

COLLATE = utf8mb4\_0900\_ai\_ci;

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

-- Table `Directory\_Aziendale`.`Piano`

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

DROP TABLE IF EXISTS `Directory\_Aziendale`.`Piano` ;

CREATE TABLE IF NOT EXISTS `Directory\_Aziendale`.`Piano` (

`Numero Piano` INT NOT NULL,

`ID Edificio` VARCHAR(45) NOT NULL,

PRIMARY KEY (`Numero Piano`, `ID Edificio`),

INDEX `ID\_Edificio\_Piano\_idx` (`ID Edificio` ASC) VISIBLE,

CONSTRAINT `ID\_Edificio\_Piano`

FOREIGN KEY (`ID Edificio`)

REFERENCES `Directory\_Aziendale`.`Edificio` (`ID`)

ON DELETE RESTRICT

ON UPDATE RESTRICT)

ENGINE = InnoDB

DEFAULT CHARACTER SET = utf8mb4

COLLATE = utf8mb4\_0900\_ai\_ci;

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

-- Table `Directory\_Aziendale`.`Ufficio`

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

DROP TABLE IF EXISTS `Directory\_Aziendale`.`Ufficio` ;

CREATE TABLE IF NOT EXISTS `Directory\_Aziendale`.`Ufficio` (

`Nome Ufficio` VARCHAR(45) NOT NULL,

`Numero Piano` INT NOT NULL,

`ID Edificio` VARCHAR(45) NOT NULL,

`Mansione Assegnata` VARCHAR(45) NOT NULL,

PRIMARY KEY (`Nome Ufficio`, `Numero Piano`, `ID Edificio`),

INDEX `ID\_Edificio\_Ufficio\_idx` (`ID Edificio` ASC) VISIBLE,

INDEX `Numero\_Piano\_Ufficio\_idx` (`Numero Piano` ASC) VISIBLE,

INDEX `Mansione\_Ufficio\_idx` (`Mansione Assegnata` ASC) VISIBLE,

CONSTRAINT `ID\_Edificio\_Ufficio`

FOREIGN KEY (`ID Edificio`)

REFERENCES `Directory\_Aziendale`.`Edificio` (`ID`),

CONSTRAINT `Mansione\_Ufficio`

FOREIGN KEY (`Mansione Assegnata`)

REFERENCES `Directory\_Aziendale`.`Mansione` (`Nome Mansione`),

CONSTRAINT `Numero\_Piano\_Ufficio`

FOREIGN KEY (`Numero Piano`)

REFERENCES `Directory\_Aziendale`.`Piano` (`Numero Piano`))

ENGINE = InnoDB

DEFAULT CHARACTER SET = utf8mb4

COLLATE = utf8mb4\_0900\_ai\_ci;

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

-- Table `Directory\_Aziendale`.`Postazione`

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

DROP TABLE IF EXISTS `Directory\_Aziendale`.`Postazione` ;

CREATE TABLE IF NOT EXISTS `Directory\_Aziendale`.`Postazione` (

`Telefono Esterno` VARCHAR(45) NOT NULL,

`Telefono Interno` VARCHAR(45) NOT NULL,

`Nome Ufficio` VARCHAR(45) NOT NULL,

`Numero Piano` INT NOT NULL,

`ID Edificio` VARCHAR(45) NOT NULL,

PRIMARY KEY (`Telefono Esterno`),

INDEX `NomeUfficio\_Postazione\_idx` (`Nome Ufficio` ASC) VISIBLE,

INDEX `NumeroPiano\_Ufficio\_idx` (`Numero Piano` ASC) VISIBLE,

INDEX `IDEdificio\_Postazione\_idx` (`ID Edificio` ASC) VISIBLE,

CONSTRAINT `IDEdificio\_Postazione`

FOREIGN KEY (`ID Edificio`)

REFERENCES `Directory\_Aziendale`.`Edificio` (`ID`)

ON DELETE RESTRICT,

CONSTRAINT `NomeUfficio\_Postazione`

FOREIGN KEY (`Nome Ufficio`)

REFERENCES `Directory\_Aziendale`.`Ufficio` (`Nome Ufficio`),

CONSTRAINT `NumeroPiano\_Postazione`

FOREIGN KEY (`Numero Piano`)

REFERENCES `Directory\_Aziendale`.`Piano` (`Numero Piano`))

ENGINE = InnoDB

DEFAULT CHARACTER SET = utf8mb4

COLLATE = utf8mb4\_0900\_ai\_ci;

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

-- Table `Directory\_Aziendale`.`Dipendente`

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

DROP TABLE IF EXISTS `Directory\_Aziendale`.`Dipendente` ;

CREATE TABLE IF NOT EXISTS `Directory\_Aziendale`.`Dipendente` (

`Codice Fiscale` VARCHAR(16) NOT NULL,

`Telefono Esterno` VARCHAR(45) NULL,

`Data Avvenuto Trasferimento` DATE NOT NULL,

`Mansione Dipendente` VARCHAR(45) NOT NULL COMMENT 'La Mansione assegnata non è in foreign key con il nome mansione della tabella mansione, perchè questo attributo rappresenta la mansione che l\'ufficio amministrativo assegna al dipendente, e in alcune fasi puo\' essere diversa\nda quella dell\'ufficio a cui è assegnato il dipendente. Per esempio se il dipendente è assegnato alla mansione X, e lavora in un ufficio assegnato a tale mansione, il settore amministrativo puo\' arbitrariamente assegnare a questo\ndipendente la mansione Y. A questo punto, e finchè non verrà trasferito, la mansione assegnata al dipendente sarà Y, quella dell\'ufficio in cui si trova sarà ancora X.',

PRIMARY KEY (`Codice Fiscale`),

UNIQUE INDEX `Telefono Esterno\_UNIQUE` (`Telefono Esterno` ASC) VISIBLE,

INDEX `Mansione\_fk\_idx` (`Mansione Dipendente` ASC) VISIBLE,

CONSTRAINT `Telefono\_Postazione`

FOREIGN KEY (`Telefono Esterno`)

REFERENCES `Directory\_Aziendale`.`Postazione` (`Telefono Esterno`)

ON DELETE NO ACTION

ON UPDATE RESTRICT,

CONSTRAINT `Mansione\_fk`

FOREIGN KEY (`Mansione Dipendente`)

REFERENCES `Directory\_Aziendale`.`Mansione` (`Nome Mansione`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB

DEFAULT CHARACTER SET = utf8mb4

COLLATE = utf8mb4\_0900\_ai\_ci

PACK\_KEYS = DEFAULT;

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

-- Table `Directory\_Aziendale`.`Dati Dipendente`

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

DROP TABLE IF EXISTS `Directory\_Aziendale`.`Dati Dipendente` ;

CREATE TABLE IF NOT EXISTS `Directory\_Aziendale`.`Dati Dipendente` (

`CF Dipendente` VARCHAR(16) NOT NULL,

`Nome` VARCHAR(45) NOT NULL,

`Cognome` VARCHAR(45) NOT NULL,

`Data Di Nascita` DATE NOT NULL,

`Luogo di Nascita` VARCHAR(45) NOT NULL,

`Città` VARCHAR(45) NOT NULL,

`Via` VARCHAR(45) NOT NULL,

`Numero Civico` INT NOT NULL,

`Email Personale` VARCHAR(45) NOT NULL,

`Email Aziendale` VARCHAR(45) NOT NULL,

PRIMARY KEY (`CF Dipendente`),

CONSTRAINT `CF\_Dipendente\_Dati`

FOREIGN KEY (`CF Dipendente`)

REFERENCES `Directory\_Aziendale`.`Dipendente` (`Codice Fiscale`)

ON DELETE CASCADE

ON UPDATE RESTRICT)

ENGINE = InnoDB

DEFAULT CHARACTER SET = utf8mb4

COLLATE = utf8mb4\_0900\_ai\_ci;

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

-- Table `Directory\_Aziendale`.`Trasferimento`

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

DROP TABLE IF EXISTS `Directory\_Aziendale`.`Trasferimento` ;

CREATE TABLE IF NOT EXISTS `Directory\_Aziendale`.`Trasferimento` (

`Data Trasferimento` DATE NOT NULL,

`Vecchia Postazione` VARCHAR(45) NOT NULL,

`Nuova Postazione` VARCHAR(45) NOT NULL,

`CF Dipendente` VARCHAR(16) NOT NULL,

PRIMARY KEY (`Data Trasferimento`, `Vecchia Postazione`, `Nuova Postazione`),

INDEX `Vecchia\_Postazione\_idx` (`Vecchia Postazione` ASC) VISIBLE,

INDEX `Nuova\_Postazione\_idx` (`Nuova Postazione` ASC) VISIBLE,

INDEX `CFDipendente\_Trasferito\_idx` (`CF Dipendente` ASC) VISIBLE,

CONSTRAINT `CFDipendente\_Trasferito`

FOREIGN KEY (`CF Dipendente`)

REFERENCES `Directory\_Aziendale`.`Dipendente` (`Codice Fiscale`)

ON DELETE CASCADE

ON UPDATE RESTRICT,

CONSTRAINT `Nuova\_Postazione`

FOREIGN KEY (`Nuova Postazione`)

REFERENCES `Directory\_Aziendale`.`Postazione` (`Telefono Esterno`),

CONSTRAINT `Vecchia\_Postazione`

FOREIGN KEY (`Vecchia Postazione`)

REFERENCES `Directory\_Aziendale`.`Postazione` (`Telefono Esterno`))

ENGINE = InnoDB

DEFAULT CHARACTER SET = utf8mb4

COLLATE = utf8mb4\_0900\_ai\_ci;

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

-- Table `Directory\_Aziendale`.`Utenti`

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

DROP TABLE IF EXISTS `Directory\_Aziendale`.`Utenti` ;

CREATE TABLE IF NOT EXISTS `Directory\_Aziendale`.`Utenti` (

`Username` VARCHAR(45) CHARACTER SET 'utf8mb4' COLLATE 'utf8mb4\_0900\_ai\_ci' NOT NULL,

`Password` VARCHAR(45) CHARACTER SET 'utf8mb4' COLLATE 'utf8mb4\_0900\_ai\_ci' NOT NULL,

`Ruolo` ENUM('dipendente', 'dipendente settore spazi', 'dipendente settore amministrativo') CHARACTER SET 'utf8mb4' COLLATE 'utf8mb4\_0900\_ai\_ci' NOT NULL,

PRIMARY KEY (`Username`))

DEFAULT CHARACTER SET = utf8mb4

COLLATE = utf8mb4\_0900\_ai\_ci

PACK\_KEYS = DEFAULT;

USE `Directory\_Aziendale` ;

USE `Directory\_Aziendale`$$

DROP TRIGGER IF EXISTS `Directory\_Aziendale`.`CheckMansione` $$

USE `Directory\_Aziendale`$$

CREATE TRIGGER `Directory\_Aziendale`.`CheckMansione` BEFORE UPDATE

ON `Dipendente` FOR EACH ROW

BEGIN

if (NEW.`Mansione Dipendente` <> (SELECT `Mansione Assegnata` FROM Postazione P join Ufficio U on

U.`Nome Ufficio` = P.`Nome Ufficio` and U.`Numero Piano` = P.`Numero Piano` and U.`ID Edificio` = P.`ID Edificio`

WHERE P.`Telefono Esterno` = NEW.`Telefono Esterno`) AND NEW.`Telefono Esterno` <> OLD.`Telefono Esterno`) then

signal sqlstate '45001' set message\_text = 'Mansione del nuovo ufficio incompatibile con mansione dipendente';

end if;

END$$

USE `Directory\_Aziendale`$$

DROP TRIGGER IF EXISTS `Directory\_Aziendale`.`CheckStorico` $$

USE `Directory\_Aziendale`$$

CREATE TRIGGER `Directory\_Aziendale`.`CheckStorico` BEFORE UPDATE

ON `Dipendente` FOR EACH ROW

BEGIN

declare var\_todayDate datetime;

SELECT curdate() into var\_todayDate;

if (NEW.`Telefono Esterno` <> OLD.`Telefono Esterno` AND NEW.`Telefono Esterno` in (SELECT `Vecchia Postazione` FROM Trasferimento T join Dipendente D

on D.`Codice Fiscale` = T.`CF Dipendente`) ) then

signal sqlstate '45002' set message\_text = 'Dipendente già assegnato a questa postazione negli ultimi 3 anni ';

end if;

END$$

USE `Directory\_Aziendale`$$

DROP TRIGGER IF EXISTS `Directory\_Aziendale`.`CreaTrasferimento` $$

USE `Directory\_Aziendale`$$

CREATE TRIGGER `Directory\_Aziendale`.`CreaTrasferimento` AFTER UPDATE

ON `Dipendente` FOR EACH ROW

BEGIN

if(NEW.`Telefono Esterno` <> OLD.`Telefono Esterno`) then

INSERT INTO `Trasferimento` (`Data Trasferimento`, `Vecchia Postazione`, `Nuova Postazione`, `CF Dipendente`)

VALUES (curdate(), OLD.`Telefono Esterno` , NEW.`Telefono Esterno`, NEW.`Codice Fiscale`);

end if;

END$$

USE `Directory\_Aziendale`$$

DROP TRIGGER IF EXISTS `Directory\_Aziendale`.`RevisioneTrasferimenti` $$

USE `Directory\_Aziendale`$$

CREATE TRIGGER `Directory\_Aziendale`.`RevisioneTrasferimenti` AFTER UPDATE

ON `Dipendente` FOR EACH ROW

BEGIN

/\* Setto momentaneamente i safe updates a 0 perchè così posso eseguire delle delete senza dover avere una "where" con

un riferimento a una primary key, poiché non mi interessa a chi appartengono i dati della tupla, se appartiene a un

trasferimento di più di 3 anni fa, va comunque eliminata. \*/

SET SQL\_SAFE\_UPDATES = 0;

-- Valutare cursore in next version

DELETE FROM `Trasferimento`

WHERE DATEDIFF(CURDATE(), `Trasferimento`.`Data Trasferimento`) > 1095; /\* Se sono passati più di 3 anni

(ovvero 1095 giorni) il record del trasferimento non serve più, e viene eiliminato \*/

SET SQL\_SAFE\_UPDATES = 1;

END$$

USE `Directory\_Aziendale`$$

DROP TRIGGER IF EXISTS `Directory\_Aziendale`.`UpdateEmailAziendale` $$

USE `Directory\_Aziendale`$$

CREATE TRIGGER `UpdateEmailAziendale` AFTER UPDATE ON `Dipendente` FOR EACH ROW

BEGIN

declare var\_dominioEmail varchar(45);

declare var\_nome varchar(45);

declare var\_cognome varchar(45);

if (NEW.`Telefono Esterno` <> OLD.`Telefono Esterno`) then

SELECT P.`Nome Ufficio`

FROM `Dipendente` D join `Postazione` P on P.`Telefono Esterno` = D.`Telefono Esterno`

WHERE D.`Codice Fiscale` = NEW.`Codice Fiscale` INTO var\_dominioEmail;

SELECT I.`Nome`, I.`Cognome`

FROM `Dipendente` D join `Dati Dipendente` I on I.`CF Dipendente` = D.`Codice FIscale`

WHERE D.`Codice Fiscale` = NEW.`Codice Fiscale`

INTO var\_nome, var\_cognome;

/\* Genera un email del tipo nome.cognome00@nomeufficio.it dove '00' è un numero di due cifre randomico, per gestire

eventuali omonimie \*/

UPDATE `Directory\_Aziendale`.`Dati Dipendente` SET `Email Aziendale` = lower(concat(var\_nome, '.', var\_cognome,

FLOOR(RAND()\*(99-1)+1), '@', replace(var\_dominioEmail, ' ', ''), '.it'))

WHERE (`CF Dipendente` = NEW.`Codice Fiscale`);

end if;

END$$

USE `Directory\_Aziendale`$$

DROP TRIGGER IF EXISTS `Directory\_Aziendale`.`CreaEmailAziendale` $$

USE `Directory\_Aziendale`$$

CREATE TRIGGER `Directory\_Aziendale`.`CreaEmailAziendale` BEFORE INSERT ON `Dati Dipendente` FOR EACH ROW

BEGIN

declare var\_dominioEmail varchar(45);

declare var\_email varchar(45);

SELECT P.`Nome Ufficio`

FROM `Dipendente` D join `Postazione` P on P.`Telefono Esterno` = D.`Telefono Esterno`

WHERE D.`Codice Fiscale` = NEW.`CF Dipendente` INTO var\_dominioEmail;

/\* Genera un email del tipo nome.cognome00@nomeufficio.it dove '00' è un numero di due cifre randomico, per gestire

eventuali omonimie \*/

SELECT lower(concat(NEW.`Nome`, '.', NEW.`Cognome`,FLOOR(RAND()\*(99-1)+1),

'@', replace(var\_dominioEmail, ' ', ''), '.it')) into var\_email;

SET NEW.`Email Aziendale` = var\_email;

END$$

DELIMITER ;

SET SQL\_MODE = '';

DROP USER IF EXISTS DipendenteSettoreSpazi;

SET SQL\_MODE='ONLY\_FULL\_GROUP\_BY,STRICT\_TRANS\_TABLES,NO\_ZERO\_IN\_DATE,NO\_ZERO\_DATE,ERROR\_FOR\_DIVISION\_BY\_ZERO,NO\_ENGINE\_SUBSTITUTION';

CREATE USER 'DipendenteSettoreSpazi' IDENTIFIED BY 'spazi';

GRANT EXECUTE ON procedure `Directory\_Aziendale`.`DipendentiTrasferireMansione` TO 'DipendenteSettoreSpazi';

GRANT EXECUTE ON procedure `Directory\_Aziendale`.`Trasferimento` TO 'DipendenteSettoreSpazi';

GRANT EXECUTE ON procedure `Directory\_Aziendale`.`ScambioPostazione` TO 'DipendenteSettoreSpazi';

GRANT EXECUTE ON procedure `Directory\_Aziendale`.`InserisciDipendente` TO 'DipendenteSettoreSpazi';

GRANT EXECUTE ON procedure `Directory\_Aziendale`.`EliminaDipendente` TO 'DipendenteSettoreSpazi';

GRANT EXECUTE ON procedure `Directory\_Aziendale`.`RicercaDipendenteNome` TO 'DipendenteSettoreSpazi';

GRANT EXECUTE ON procedure `Directory\_Aziendale`.`RicercaPostazione` TO 'DipendenteSettoreSpazi';

GRANT EXECUTE ON procedure `Directory\_Aziendale`.`RicercaDipendenteCognome` TO 'DipendenteSettoreSpazi';

GRANT EXECUTE ON procedure `Directory\_Aziendale`.`RicercaDipNomeCognome` TO 'DipendenteSettoreSpazi';

GRANT EXECUTE ON procedure `Directory\_Aziendale`.`ReportDipendentiTrasferire` TO 'DipendenteSettoreSpazi';

SET SQL\_MODE = '';

DROP USER IF EXISTS DipendenteSettoreAmministrativo;

SET SQL\_MODE='ONLY\_FULL\_GROUP\_BY,STRICT\_TRANS\_TABLES,NO\_ZERO\_IN\_DATE,NO\_ZERO\_DATE,ERROR\_FOR\_DIVISION\_BY\_ZERO,NO\_ENGINE\_SUBSTITUTION';

CREATE USER 'DipendenteSettoreAmministrativo' IDENTIFIED BY 'amministrativo';

GRANT EXECUTE ON procedure `Directory\_Aziendale`.`ModificaMansione` TO 'DipendenteSettoreAmministrativo';

GRANT EXECUTE ON procedure `Directory\_Aziendale`.`StoricoUffici` TO 'DipendenteSettoreAmministrativo';

GRANT EXECUTE ON procedure `Directory\_Aziendale`.`RicercaDipendenteNome` TO 'DipendenteSettoreAmministrativo';

GRANT EXECUTE ON procedure `Directory\_Aziendale`.`RicercaPostazione` TO 'DipendenteSettoreAmministrativo';

GRANT EXECUTE ON procedure `Directory\_Aziendale`.`RicercaDipNomeCognome` TO 'DipendenteSettoreAmministrativo';

GRANT EXECUTE ON procedure `Directory\_Aziendale`.`RicercaDipendenteCognome` TO 'DipendenteSettoreAmministrativo';

SET SQL\_MODE = '';

DROP USER IF EXISTS Dipendente;

SET SQL\_MODE='ONLY\_FULL\_GROUP\_BY,STRICT\_TRANS\_TABLES,NO\_ZERO\_IN\_DATE,NO\_ZERO\_DATE,ERROR\_FOR\_DIVISION\_BY\_ZERO,NO\_ENGINE\_SUBSTITUTION';

CREATE USER 'Dipendente' IDENTIFIED BY 'dipendente';

GRANT EXECUTE ON procedure `Directory\_Aziendale`.`RicercaDipendenteNome` TO 'Dipendente';

GRANT EXECUTE ON procedure `Directory\_Aziendale`.`RicercaPostazione` TO 'Dipendente';

GRANT EXECUTE ON procedure `Directory\_Aziendale`.`RicercaDipNomeCognome` TO 'Dipendente';

GRANT EXECUTE ON procedure `Directory\_Aziendale`.`RicercaDipendenteCognome` TO 'Dipendente';

SET SQL\_MODE = '';

DROP USER IF EXISTS login;

SET SQL\_MODE='ONLY\_FULL\_GROUP\_BY,STRICT\_TRANS\_TABLES,NO\_ZERO\_IN\_DATE,NO\_ZERO\_DATE,ERROR\_FOR\_DIVISION\_BY\_ZERO,NO\_ENGINE\_SUBSTITUTION';

CREATE USER 'login' IDENTIFIED BY 'login';

GRANT EXECUTE ON procedure `Directory\_Aziendale`.`login` TO 'login';

SET SQL\_MODE=@OLD\_SQL\_MODE;

SET FOREIGN\_KEY\_CHECKS=@OLD\_FOREIGN\_KEY\_CHECKS;

SET UNIQUE\_CHECKS=@OLD\_UNIQUE\_CHECKS;

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

-- Data for table `Directory\_Aziendale`.`Edificio`

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

START TRANSACTION;

USE `Directory\_Aziendale`;

INSERT INTO `Directory\_Aziendale`.`Edificio` (`ID`) VALUES ('Edificio A');

INSERT INTO `Directory\_Aziendale`.`Edificio` (`ID`) VALUES ('Edificio B');

INSERT INTO `Directory\_Aziendale`.`Edificio` (`ID`) VALUES ('Edificio Amministrazione');

COMMIT;

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

-- Data for table `Directory\_Aziendale`.`Mansione`

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

START TRANSACTION;

USE `Directory\_Aziendale`;

INSERT INTO `Directory\_Aziendale`.`Mansione` (`Nome Mansione`) VALUES ('Manager');

INSERT INTO `Directory\_Aziendale`.`Mansione` (`Nome Mansione`) VALUES ('Sistemista');

INSERT INTO `Directory\_Aziendale`.`Mansione` (`Nome Mansione`) VALUES ('Advertising');

INSERT INTO `Directory\_Aziendale`.`Mansione` (`Nome Mansione`) VALUES ('Spazi');

INSERT INTO `Directory\_Aziendale`.`Mansione` (`Nome Mansione`) VALUES ('Amministrativo');

COMMIT;

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

-- Data for table `Directory\_Aziendale`.`Piano`

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

START TRANSACTION;

USE `Directory\_Aziendale`;

INSERT INTO `Directory\_Aziendale`.`Piano` (`Numero Piano`, `ID Edificio`) VALUES (1, 'Edificio A');

INSERT INTO `Directory\_Aziendale`.`Piano` (`Numero Piano`, `ID Edificio`) VALUES (2, 'Edificio B');

INSERT INTO `Directory\_Aziendale`.`Piano` (`Numero Piano`, `ID Edificio`) VALUES (1, 'Edificio B');

INSERT INTO `Directory\_Aziendale`.`Piano` (`Numero Piano`, `ID Edificio`) VALUES (2, 'Edificio A');

INSERT INTO `Directory\_Aziendale`.`Piano` (`Numero Piano`, `ID Edificio`) VALUES (3, 'Edificio A');

INSERT INTO `Directory\_Aziendale`.`Piano` (`Numero Piano`, `ID Edificio`) VALUES (1, 'Edificio Amministrazione');

COMMIT;

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

-- Data for table `Directory\_Aziendale`.`Ufficio`

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

START TRANSACTION;

USE `Directory\_Aziendale`;

INSERT INTO `Directory\_Aziendale`.`Ufficio` (`Nome Ufficio`, `Numero Piano`, `ID Edificio`, `Mansione Assegnata`) VALUES ('Sistemi A', 1, 'Edificio A', 'Sistemista');

INSERT INTO `Directory\_Aziendale`.`Ufficio` (`Nome Ufficio`, `Numero Piano`, `ID Edificio`, `Mansione Assegnata`) VALUES ('Adv B', 2, 'Edificio B', 'Advertising');

INSERT INTO `Directory\_Aziendale`.`Ufficio` (`Nome Ufficio`, `Numero Piano`, `ID Edificio`, `Mansione Assegnata`) VALUES ('Managment A', 2, 'Edificio A', 'Manager');

INSERT INTO `Directory\_Aziendale`.`Ufficio` (`Nome Ufficio`, `Numero Piano`, `ID Edificio`, `Mansione Assegnata`) VALUES ('Ufficio Settore Spazi', 3, 'Edificio A', 'Spazi');

INSERT INTO `Directory\_Aziendale`.`Ufficio` (`Nome Ufficio`, `Numero Piano`, `ID Edificio`, `Mansione Assegnata`) VALUES ('Ufficio Amministrativo', 1, 'Edificio Amministrazione', 'Amministrativo');

COMMIT;

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

-- Data for table `Directory\_Aziendale`.`Postazione`

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

START TRANSACTION;

USE `Directory\_Aziendale`;

INSERT INTO `Directory\_Aziendale`.`Postazione` (`Telefono Esterno`, `Telefono Interno`, `Nome Ufficio`, `Numero Piano`, `ID Edificio`) VALUES ('12345', '345', 'Sistemi A', 1, 'Edificio A');

INSERT INTO `Directory\_Aziendale`.`Postazione` (`Telefono Esterno`, `Telefono Interno`, `Nome Ufficio`, `Numero Piano`, `ID Edificio`) VALUES ('23456', '456', 'Adv B', 2, 'Edificio B');

INSERT INTO `Directory\_Aziendale`.`Postazione` (`Telefono Esterno`, `Telefono Interno`, `Nome Ufficio`, `Numero Piano`, `ID Edificio`) VALUES ('34567', '567', 'Adv B', 2, 'Edificio B');

INSERT INTO `Directory\_Aziendale`.`Postazione` (`Telefono Esterno`, `Telefono Interno`, `Nome Ufficio`, `Numero Piano`, `ID Edificio`) VALUES ('0652', '333', 'Ufficio Settore Spazi', 3, 'Edificio A');

INSERT INTO `Directory\_Aziendale`.`Postazione` (`Telefono Esterno`, `Telefono Interno`, `Nome Ufficio`, `Numero Piano`, `ID Edificio`) VALUES ('999', '121', 'Ufficio Amministrativo', 1, 'Edificio Amministrazione');

INSERT INTO `Directory\_Aziendale`.`Postazione` (`Telefono Esterno`, `Telefono Interno`, `Nome Ufficio`, `Numero Piano`, `ID Edificio`) VALUES ('778', '567', 'Managment A', 2, 'Edificio A');

COMMIT;

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

-- Data for table `Directory\_Aziendale`.`Utenti`

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

START TRANSACTION;

USE `Directory\_Aziendale`;

INSERT INTO `Directory\_Aziendale`.`Utenti` (`Username`, `Password`, `Ruolo`) VALUES ('dipendente', 'a17a54e46fb7f594574a9d74b36fe68a', 'dipendente');

INSERT INTO `Directory\_Aziendale`.`Utenti` (`Username`, `Password`, `Ruolo`) VALUES ('spazi', '57f6e04fa9e62b2d7741edcd181f3a47', 'dipendente settore spazi');

INSERT INTO `Directory\_Aziendale`.`Utenti` (`Username`, `Password`, `Ruolo`) VALUES ('amministrativo', '60c1bde27e22fc7a51b783265cb83772', 'dipendente settore amministrativo');

COMMIT;

## Codice del Front-End

- main.c

#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
#include <mysql.h>  
  
#include "defines.h"  
  
  
typedef enum {  
 *DIPENDENTE* = 1,  
 *SPAZI*,  
 *AMMINISTRATIVO*,  
 *FAILED\_LOGIN*  
} role\_t;  
  
  
struct configuration conf;  
  
static MYSQL \*conn;  
  
  
static role\_t attempt\_login(MYSQL \*conn, char \*username, char \*password) {  
 MYSQL\_STMT \*login\_procedure;  
   
 MYSQL\_BIND param[3]; // Used both for input and output  
 int role = 0;  
  
 if(!setup\_prepared\_stmt(&login\_procedure, "call login(?, ?, ?)", conn)) {  
 print\_stmt\_error(login\_procedure, "Unable to initialize login statement\n");  
 goto err2;  
 }  
  
 // Prepare parameters  
 memset(param, 0, sizeof(param));  
   
 param[0].buffer\_type = *MYSQL\_TYPE\_VAR\_STRING*; // IN  
 param[0].buffer = username;  
 param[0].buffer\_length = strlen(username);  
  
 param[1].buffer\_type = *MYSQL\_TYPE\_VAR\_STRING*; // IN  
 param[1].buffer = password;  
 param[1].buffer\_length = strlen(password);  
  
 param[2].buffer\_type = *MYSQL\_TYPE\_LONG*; // OUT  
 param[2].buffer = &role;  
 param[2].buffer\_length = sizeof(role);  
  
 if (mysql\_stmt\_bind\_param(login\_procedure, param) != 0) { // Note \_param  
 print\_stmt\_error(login\_procedure, "Could not bind parameters for login");  
 goto err;  
 }  
  
 // Run procedure  
 if (mysql\_stmt\_execute(login\_procedure) != 0) {  
 print\_stmt\_error(login\_procedure, "Could not execute login procedure");  
 goto err;  
 }  
  
 // Prepare output parameters  
 memset(param, 0, sizeof(param));  
 param[0].buffer\_type = *MYSQL\_TYPE\_LONG*; // OUT  
 param[0].buffer = &role;  
 param[0].buffer\_length = sizeof(role);  
   
 if(mysql\_stmt\_bind\_result(login\_procedure, param)) {  
 print\_stmt\_error(login\_procedure, "Could not retrieve output parameter");  
 goto err;  
 }  
   
 // Retrieve output parameter  
 if(mysql\_stmt\_fetch(login\_procedure)) {  
 print\_stmt\_error(login\_procedure, "Could not buffer results");  
 goto err;  
 }  
  
 mysql\_stmt\_close(login\_procedure);  
 return role;  
  
 err:  
 mysql\_stmt\_close(login\_procedure);  
 err2:  
 return *FAILED\_LOGIN*;  
}  
  
/\* Le funzioni di appoggio per la CLI e le utility per i file json e per sql sono state prese dall'esempio del professor Pellegrini \*/  
  
int main(void) {  
 role\_t role;  
  
 if(!parse\_config("users/login.json", &conf)) {  
 fprintf(stderr, "Unable to load login configuration\n");  
 exit(EXIT\_FAILURE);  
 }  
  
 conn = mysql\_init (NULL);  
 if (conn == NULL) {  
 fprintf (stderr, "mysql\_init() failed (probably out of memory)\n");  
 exit(EXIT\_FAILURE);  
 }  
  
 if (mysql\_real\_connect(conn, conf.host, conf.db\_username, conf.db\_password, conf.database, conf.port, NULL, CLIENT\_MULTI\_STATEMENTS | CLIENT\_MULTI\_RESULTS) == NULL) {  
 fprintf (stderr, "mysql\_real\_connect() failed\n");  
 mysql\_close (conn);  
 exit(EXIT\_FAILURE);  
 }  
  
 printf("Username: ");  
 getInput(128, conf.username, false);  
 printf("Password: ");  
 getInput(128, conf.password, true);  
  
 role = attempt\_login(conn, conf.username, conf.password);  
  
 switch(role) {  
 case *DIPENDENTE*:  
 runAsDipendente(conn);  
 break;  
   
 case *SPAZI*:  
 runAsSetSpazi(conn);  
 break;  
   
 case *AMMINISTRATIVO*:  
 runAsAmministrativo(conn);  
 break;  
   
 case *FAILED\_LOGIN*:  
 fprintf(stderr, "Invalid credentials\n");  
 exit(EXIT\_FAILURE);  
 break;  
  
 default:  
 fprintf(stderr, "Invalid condition at %s:%d\n", \_\_FILE\_\_, \_\_LINE\_\_);  
 abort();  
 }  
  
 printf("Bye!\n");  
  
 mysql\_close (conn);  
 return 0;  
}

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

dipendente.c

#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
  
#include "defines.h"  
  
  
void ricercaDipNomeCognome(MYSQL \*conn)  
{  
 MYSQL\_STMT \*prepared\_stmt;  
 MYSQL\_BIND param[2];  
  
 // Prepare stored procedure call  
 if(!setup\_prepared\_stmt(&prepared\_stmt, "call RicercaDipNomeCognome(?, ?)", conn)) {  
 finish\_with\_stmt\_error(conn, prepared\_stmt, "Unable to initialize 'ricercaDipNomeCognome' statement\n", false);  
 }  
  
 // Prepare parameters  
 memset(param, 0, sizeof(param));  
  
 param[0].buffer\_type = *MYSQL\_TYPE\_VAR\_STRING*;  
 param[1].buffer\_type = *MYSQL\_TYPE\_VAR\_STRING*;  
 char nome[45];  
 char cognome[45];  
 printf("Inserisci il nome del dipendente da cercare: ");  
 getInput(45, nome, false);  
 printf("Inserisci il cognome del dipendente da cercare: ");  
 getInput(45, cognome, false);  
 param[0].buffer = nome;  
 param[1].buffer = cognome;  
 param[0].buffer\_length = strlen((char \*)param[0].buffer);  
 param[1].buffer\_length = strlen((char \*)param[1].buffer);  
  
  
 if (mysql\_stmt\_bind\_param(prepared\_stmt, param) != 0) {  
 finish\_with\_stmt\_error(conn, prepared\_stmt, "Could not bind parameters for ricercaDipNomeCognome\n", true);  
 }  
  
 // Run procedure  
 if (mysql\_stmt\_execute(prepared\_stmt) != 0) {  
 print\_stmt\_error(prepared\_stmt, "An error occurred while retrieving the employees names.");  
 goto out;  
}  
  
  
  
do {  
 dump\_result\_set(conn, prepared\_stmt, "\nDipendenti trovati:");  
} while (mysql\_stmt\_next\_result(prepared\_stmt) == 0);//mi assicuro di consumare tutto il result set, altrimenti potrei ottenere "out of sync"  
  
 out:  
 mysql\_stmt\_close(prepared\_stmt);  
}  
  
  
  
void ricercaDipCognome(MYSQL \*conn) {  
 MYSQL\_STMT \*prepared\_stmt;  
 MYSQL\_BIND param[1];  
  
 // Prepare stored procedure call  
 if(!setup\_prepared\_stmt(&prepared\_stmt, "call RicercaDipendenteCognome(?)", conn)) {  
 finish\_with\_stmt\_error(conn, prepared\_stmt, "Unable to initialize 'ricercaDipNome' statement\n", false);  
 }  
  
 // Prepare parameters  
 memset(param, 0, sizeof(param));  
  
 param[0].buffer\_type = *MYSQL\_TYPE\_VAR\_STRING*;  
 printf("Inserisci il cognome del dipendente da cercare: ");  
 char string[45];  
 getInput(128, string, false);  
 param[0].buffer = string;  
 param[0].buffer\_length = strlen((char \*)param[0].buffer);  
  
 if (mysql\_stmt\_bind\_param(prepared\_stmt, param) != 0) {  
 finish\_with\_stmt\_error(conn, prepared\_stmt, "Could not bind parameters for 'ricerca dipendente per nome'\n", true);  
 }  
  
 // Run procedure  
 if (mysql\_stmt\_execute(prepared\_stmt) != 0) {  
 print\_stmt\_error(prepared\_stmt, "An error occurred while retrieving the employees' names.");  
 goto out;  
 }  
  
  
  
 do {  
 dump\_result\_set(conn, prepared\_stmt, "\nDipendenti trovati:");  
 } while (mysql\_stmt\_next\_result(prepared\_stmt) == 0); //mi assicuro di consumare tutto il result set, altrimenti potrei ottenere "out of sync"  
  
 out:  
 mysql\_stmt\_close(prepared\_stmt);  
}  
  
void ricercaDipNome(MYSQL \*conn) {  
 MYSQL\_STMT \*prepared\_stmt;  
 MYSQL\_BIND param[1];  
  
 // Prepare stored procedure call  
 if(!setup\_prepared\_stmt(&prepared\_stmt, "call RicercaDipendenteNome(?)", conn)) {  
 finish\_with\_stmt\_error(conn, prepared\_stmt, "Unable to initialize 'ricercaDipNome' statement\n", false);  
 }  
  
 // Prepare parameters  
 memset(param, 0, sizeof(param));  
  
 param[0].buffer\_type = *MYSQL\_TYPE\_VAR\_STRING*;  
 printf("Inserisci il nome del dipendente da cercare: ");  
 char string[45];  
 getInput(128, string, false);  
 param[0].buffer = string;  
 param[0].buffer\_length = strlen((char \*)param[0].buffer);  
  
 if (mysql\_stmt\_bind\_param(prepared\_stmt, param) != 0) {  
 finish\_with\_stmt\_error(conn, prepared\_stmt, "Could not bind parameters for 'ricercaDipNome'\n", true);  
 }  
  
 // Run procedure  
 if (mysql\_stmt\_execute(prepared\_stmt) != 0) {  
 print\_stmt\_error(prepared\_stmt, "An error occurred while retrieving the employees' names .");  
 goto out;  
 }  
  
  
  
 do {  
 dump\_result\_set(conn, prepared\_stmt, "\nDipendenti trovati:");  
 } while (mysql\_stmt\_next\_result(prepared\_stmt) == 0);//mi assicuro di consumare tutto il result set, altrimenti potrei ottenere "out of sync"  
  
 out:  
 mysql\_stmt\_close(prepared\_stmt);  
}  
  
void ricercaPostazione(MYSQL \*conn)  
{  
 MYSQL\_STMT \*prepared\_stmt;  
 MYSQL\_BIND param[1];  
  
 // Prepare stored procedure call  
 if(!setup\_prepared\_stmt(&prepared\_stmt, "call RicercaPostazione(?)", conn)) {  
 finish\_with\_stmt\_error(conn, prepared\_stmt, "Unable to initialize ricercaPostazione statement\n", false);  
 }  
  
 // Prepare parameters  
 memset(param, 0, sizeof(param));  
  
 param[0].buffer\_type = *MYSQL\_TYPE\_VAR\_STRING*;  
 printf("Inserisci il numero di telefono ESTERNO della postazione da cercare: ");  
 char postazione[45];  
 getInput(45, postazione, false);  
 param[0].buffer = postazione;  
 param[0].buffer\_length = strlen((char \*)param[0].buffer);  
  
 if (mysql\_stmt\_bind\_param(prepared\_stmt, param) != 0) {  
 finish\_with\_stmt\_error(conn, prepared\_stmt, "Could not bind parameters for ricercaPostazione\n", true);  
 }  
  
 // Run procedure  
 if (mysql\_stmt\_execute(prepared\_stmt) != 0) {  
 print\_stmt\_error(prepared\_stmt, "An error occurred while retrieving post information.");  
 goto out;  
 }  
  
  
  
 do {  
 dump\_result\_set(conn, prepared\_stmt, "\nPostazione trovata:");  
 } while (mysql\_stmt\_next\_result(prepared\_stmt) == 0);//mi assicuro di consumare tutto il result set, altrimenti potrei ottenere "out of sync"  
  
 out:  
 mysql\_stmt\_close(prepared\_stmt);  
}  
  
  
  
void runAsDipendente(MYSQL \*conn)  
{  
 char options[5] = {'1','2', '3', '4', '5'};  
 char op;  
   
 printf("Passaggio a account di tipo dipendente...\n");  
  
 if(!parse\_config("users/dipendente.json", &conf)) {  
 fprintf(stderr, "Unable to load 'dipendente' configuration\n");  
 exit(EXIT\_FAILURE);  
 }  
  
 if(mysql\_change\_user(conn, conf.db\_username, conf.db\_password, conf.database)) {  
 fprintf(stderr, "mysql\_change\_user() failed\n");  
 exit(EXIT\_FAILURE);  
 }  
  
 while(true) {  
 printf("\033[2J\033[H");  
 printf("\*\*\* Quale Operazione vuoi eseguire? \*\*\*\n\n");  
 printf("1) Ricerca dipendente per nome\n");  
 printf("2) Ricerca dipendente per cognome\n");  
 printf("3) Ricerca dipendente per nome e cognome\n");  
 printf("4) Ricerca postazione\n");  
 printf("5) Esci\n");  
  
 op = multiChoice("Selezione un'opzione", options, 5);  
  
 switch(op) {  
 case '1':  
 ricercaDipNome(conn);  
 break;  
   
 case '2':  
 ricercaDipCognome(conn);  
 break;  
 case '3':  
 ricercaDipNomeCognome(conn);  
 break;  
 case '4':  
 ricercaPostazione(conn);  
 break;  
 case '5':  
 return;  
 default:  
 fprintf(stderr, "Invalid condition at %s:%d\n", \_\_FILE\_\_, \_\_LINE\_\_);  
 abort();  
 }  
  
 getchar();  
 }  
}

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

setSpazi.c

#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
  
#include "defines.h"  
  
static void reportDipendenti(MYSQL \*conn)  
{  
 MYSQL\_STMT \*prepared\_stmt;  
  
  
 // Prepare stored procedure call  
 if(!setup\_prepared\_stmt(&prepared\_stmt, "call ReportDipendentiTrasferire()", conn)) {  
 finish\_with\_stmt\_error(conn, prepared\_stmt, "Unable to initialize 'ReportDipendentiTrasferire' statement\n", false);  
 }  
  
 // No parameters to prepare  
 // Run procedure  
 if (mysql\_stmt\_execute(prepared\_stmt) != 0) {  
 print\_stmt\_error(prepared\_stmt, "An error occurred while retrieving employees' report.");  
 goto out;  
 }  
  
 do {  
 dump\_result\_set(conn, prepared\_stmt, "\nDipendenti trovati:");  
 } while (mysql\_stmt\_next\_result(prepared\_stmt) == 0); //mi assicuro di consumare tutto il result set, altrimenti potrei ottenere "out of sync"  
  
 out:  
 mysql\_stmt\_close(prepared\_stmt);  
}  
  
  
static void reportDipendentiMansione(MYSQL \*conn)  
{  
 MYSQL\_STMT \*prepared\_stmt;  
  
  
 // Prepare stored procedure call  
 if(!setup\_prepared\_stmt(&prepared\_stmt, "call DipendentiTrasferireMansione()", conn)) {  
 finish\_with\_stmt\_error(conn, prepared\_stmt, "Unable to initialize dipendentiTrasferireMansione statement\n", false);  
 }  
  
 // No parameters to prepare  
 // Run procedure  
 if (mysql\_stmt\_execute(prepared\_stmt) != 0) {  
 print\_stmt\_error(prepared\_stmt, "An error occurred while retrieving employees' report grouped by job.");  
 goto out;  
 }  
  
 do {  
 dump\_result\_set(conn, prepared\_stmt, "\nDipendenti trovati:");  
 } while (mysql\_stmt\_next\_result(prepared\_stmt) == 0);//mi assicuro di consumare tutto il result set, altrimenti potrei ottenere "out of sync"  
  
 out:  
 mysql\_stmt\_close(prepared\_stmt);  
}  
  
static void trasferisciDipendente(MYSQL \*conn)  
{  
 MYSQL\_STMT \*prepared\_stmt;  
 MYSQL\_BIND param[2];  
  
 // Prepare stored procedure call  
 if(!setup\_prepared\_stmt(&prepared\_stmt, "call Trasferimento(?, ?)", conn)) {  
 finish\_with\_stmt\_error(conn, prepared\_stmt, "Unable to initialize 'Trasferimento' statement\n", false);  
 }  
  
 // Prepare parameters  
 memset(param, 0, sizeof(param));  
  
 param[0].buffer\_type = *MYSQL\_TYPE\_VAR\_STRING*;  
 printf("Inserisci il codice fiscale del dipendente da trasferire: ");  
 char string[16];  
 getInput(16, string, false);  
 param[0].buffer = string;  
 param[0].buffer\_length = strlen((char \*)param[0].buffer);  
  
 param[1].buffer\_type = *MYSQL\_TYPE\_VAR\_STRING*;  
 printf("\nInserisci il numero ESTERNO della postazione in cui trasferire il dipendente\n");  
 printf("[N.B: La postazione deve essere libera e compatibile con la mansione del dipendente scelto]\n");  
 char postazione[45];  
 getInput(45, postazione, false);  
 param[1].buffer = postazione;  
 param[1].buffer\_length = strlen((char \*)param[1].buffer);  
  
 if (mysql\_stmt\_bind\_param(prepared\_stmt, param) != 0) {  
 finish\_with\_stmt\_error(conn, prepared\_stmt, "Could not bind parameters for transfer\n", true);  
 }  
  
 // Run procedure  
 if (mysql\_stmt\_execute(prepared\_stmt) != 0) {  
 print\_stmt\_error(prepared\_stmt, "An error occurred while transfering the employee.");  
 goto out;  
 }  
  
 //La call non produce un output in questo caso  
 printf("Trasferimento eseguito con successo");  
  
 out:  
 mysql\_stmt\_close(prepared\_stmt);  
}  
  
static void scambioDipendenti(MYSQL \*conn)  
{  
 MYSQL\_STMT \*prepared\_stmt;  
 MYSQL\_BIND param[2];  
  
 // Prepare stored procedure call  
 if(!setup\_prepared\_stmt(&prepared\_stmt, "call ScambioPostazione(?, ?)", conn)) {  
 finish\_with\_stmt\_error(conn, prepared\_stmt, "Unable to initialize 'ScambioPostazione' statement\n", false);  
 }  
  
 // Prepare parameters  
 memset(param, 0, sizeof(param));  
  
 param[0].buffer\_type = *MYSQL\_TYPE\_VAR\_STRING*;  
 printf("Inserisci il codice fiscale del primo dipendente da scambiare: ");  
 char string[16];  
 getInput(16, string, false);  
 param[0].buffer = string;  
 param[0].buffer\_length = strlen((char \*)param[0].buffer);  
  
 param[1].buffer\_type = *MYSQL\_TYPE\_VAR\_STRING*;  
 printf("\nInserisci il codice fiscale del primo dipendente da scambiare: ");  
 char string2[16];  
 getInput(16, string2, false);  
 param[1].buffer = string2;  
 param[1].buffer\_length = strlen((char \*)param[1].buffer);  
  
 if (mysql\_stmt\_bind\_param(prepared\_stmt, param) != 0) {  
 finish\_with\_stmt\_error(conn, prepared\_stmt, "Could not bind parameters for post switch\n", true);  
 }  
  
 // Run procedure  
 if (mysql\_stmt\_execute(prepared\_stmt) != 0) {  
 print\_stmt\_error(prepared\_stmt, "An error occurred while switching posts.");  
 goto out;  
 }  
  
 //La call non produce un output in questo caso  
 printf("Scambio di postazioni eseguito con successo");  
  
 out:  
 mysql\_stmt\_close(prepared\_stmt);  
}  
  
static void inserisciDipendente(MYSQL \*conn)  
{  
 MYSQL\_STMT \*prepared\_stmt;  
 MYSQL\_BIND param[12];  
 MYSQL\_TIME ts, ts2;  
  
 // Prepare stored procedure call  
 if(!setup\_prepared\_stmt(&prepared\_stmt, "call InserisciDipendente(?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ? )", conn)) {  
 finish\_with\_stmt\_error(conn, prepared\_stmt, "Unable to initialize 'InserisciDipendente' statement\n", false);  
 }  
  
 // Prepare parameters  
 memset(param, 0, sizeof(param));  
  
 //codice fiscale  
 param[0].buffer\_type = *MYSQL\_TYPE\_VAR\_STRING*;  
 printf("Inserisci il codice fiscale del dipendente da trasferire: ");  
 char string[16];  
 getInput(16, string, false);  
 param[0].buffer = string;  
 param[0].buffer\_length = strlen((char \*)param[0].buffer);  
  
 //numero postazione  
 param[1].buffer\_type = *MYSQL\_TYPE\_VAR\_STRING*;  
 printf("\nInserisci il numero ESTERNO della postazione in cui inserire il dipendente\n");  
 printf("[N.B: La postazione deve essere libera e compatibile con la mansione del dipendente scelto]\n");  
 char postazione[45];  
 getInput(45, postazione, false);  
 param[1].buffer = postazione;  
 param[1].buffer\_length = strlen((char \*)param[1].buffer);  
  
 //data trasferimento/assunzione  
 param[2].buffer\_type = *MYSQL\_TYPE\_DATE*;  
 char \*today;  
 size\_t s = 0;  
 printf("\nInserisci la data di assunzione del dipendente nel formato YYYY-MM-DD: ");  
 getline(&today, &s, stdin);  
 date\_to\_mysql\_time(today, &ts);  
 param[2].buffer = &ts;  
 param[2].buffer\_length = sizeof(ts);  
  
  
 //mansione  
 param[3].buffer\_type = *MYSQL\_TYPE\_VAR\_STRING*;  
 printf("\nInserisci la mansione del nuovo dipendente: ");  
 char mansione[45];  
 getInput(45, mansione, false);  
 param[3].buffer = mansione;  
 param[3].buffer\_length = strlen((char \*)param[3].buffer);  
  
 //nome  
 param[4].buffer\_type = *MYSQL\_TYPE\_VAR\_STRING*;  
 printf("\nInserisci il nome del nuovo dipendente: ");  
 char nome[45];  
 getInput(45, nome, false);  
 param[4].buffer = nome;  
 param[4].buffer\_length = strlen((char \*)param[4].buffer);  
  
 //cognome  
 param[5].buffer\_type = *MYSQL\_TYPE\_VAR\_STRING*;  
 printf("\nInserisci il cognome del nuovo dipendente: ");  
 char cognome[45];  
 getInput(45, cognome, false);  
 param[5].buffer = cognome;  
 param[5].buffer\_length = strlen((char \*)param[5].buffer);  
  
 //data di nascita  
 param[6].buffer\_type = *MYSQL\_TYPE\_DATE*;  
 char \*buffer;  
 size\_t sz = 0;  
 printf("\nInserisci la data di nascita del dipendente nel formato YYYY-MM-DD: ");  
 getline(&buffer, &sz, stdin);  
 date\_to\_mysql\_time(buffer, &ts2);  
 param[6].buffer = &ts2;  
 param[6].buffer\_length = sizeof(ts2);  
  
 //Luogo di nascita  
 param[7].buffer\_type = *MYSQL\_TYPE\_VAR\_STRING*;  
 printf("\nInserisci il paese di nascita del nuovo dipendente: ");  
 char luogoNascita[45];  
 getInput(45, luogoNascita, false);  
 param[7].buffer = luogoNascita;  
 param[7].buffer\_length = strlen((char \*)param[7].buffer);  
  
 //Città residenza  
 param[8].buffer\_type = *MYSQL\_TYPE\_VAR\_STRING*;  
 printf("\nInserisci la città di residenza del nuovo dipendente: ");  
 char citta[45];  
 getInput(45, citta, false);  
 param[8].buffer = citta;  
 param[8].buffer\_length = strlen((char \*)param[8].buffer);  
  
 //Via residenza  
 param[9].buffer\_type = *MYSQL\_TYPE\_VAR\_STRING*;  
 char via[45];  
 printf("\nInserisci la via di residenza del nuovo dipendente: ");  
 getInput(45, via, false);  
 param[9].buffer = via;  
 param[9].buffer\_length = strlen((char \*)param[9].buffer);  
  
 //civico indirizzo  
 param[10].buffer\_type = *MYSQL\_TYPE\_LONG*; //int in sql  
 printf("\nInserisci il numero civico di residenza del nuovo dipendente: ");  
 getline(&buffer, &sz, stdin);  
 int civico;  
 sscanf(buffer, "%d", &civico);  
 param[10].buffer = (void \*)&civico;  
 param[10].buffer\_length = strlen((char \*)param[10].buffer);  
  
 //email personale (quella aziendale è generata dal DBMS  
 param[11].buffer\_type = *MYSQL\_TYPE\_VAR\_STRING*;  
 printf("\nInserisci l'email personale del nuovo dipendente: ");  
 char emailPers[45];  
 getInput(45, emailPers, false);  
 param[11].buffer = emailPers;  
 param[11].buffer\_length = strlen((char \*)param[11].buffer);  
  
 if (mysql\_stmt\_bind\_param(prepared\_stmt, param) != 0) {  
 finish\_with\_stmt\_error(conn, prepared\_stmt, "Could not bind parameters for 'InserisciDipendente'\n", true);  
 }  
  
 // Run procedure  
 if (mysql\_stmt\_execute(prepared\_stmt) != 0) {  
 print\_stmt\_error(prepared\_stmt, "An error occurred while inserting the new employee.");  
 goto out;  
 }  
  
 //La call non produce un output in questo caso  
 printf("Inserimento dipendente eseguito con successo");  
  
 out:  
 mysql\_stmt\_close(prepared\_stmt);  
}  
  
static void eliminaDipendente(MYSQL \*conn)  
{  
 MYSQL\_STMT \*prepared\_stmt;  
 MYSQL\_BIND param[1];  
  
 // Prepare stored procedure call  
 if(!setup\_prepared\_stmt(&prepared\_stmt, "call EliminaDipendente(?)", conn)) {  
 finish\_with\_stmt\_error(conn, prepared\_stmt, "Unable to initialize 'EliminaDipendente' statement\n", false);  
 }  
  
 // Prepare parameters  
 memset(param, 0, sizeof(param));  
  
 param[0].buffer\_type = *MYSQL\_TYPE\_VAR\_STRING*;  
 printf("Inserisci il codice fiscale del dipendente da eliminare: ");  
 char string[16];  
 getInput(16, string, false);  
 param[0].buffer = string;  
 param[0].buffer\_length = strlen((char \*)param[0].buffer);  
  
 if (mysql\_stmt\_bind\_param(prepared\_stmt, param) != 0) {  
 finish\_with\_stmt\_error(conn, prepared\_stmt, "Could not bind parameters for 'EliminaDipendente'\n", true);  
 }  
  
 // Run procedure  
 if (mysql\_stmt\_execute(prepared\_stmt) != 0) {  
 print\_stmt\_error(prepared\_stmt, "An error occurred while removing the selected employee, no changes were made to Directory Aziendale.");  
 goto out;  
 }  
  
  
 //non produce result set  
 printf("Dipendente rimosso con successo dalla directory aziendale");  
  
 out:  
 mysql\_stmt\_close(prepared\_stmt);  
}  
  
  
void runAsSetSpazi(MYSQL \*conn)  
{  
 char options[11] = { '0','1','2', '3', '4', '5', '6', '7', '8', '9', 'X'};  
 char op;  
   
 printf("Passaggio a account Settore Spazi\n");  
  
 if(!parse\_config("users/setSpazi.json", &conf)) {  
 fprintf(stderr, "Unable to load professor configuration\n");  
 exit(EXIT\_FAILURE);  
 }  
  
 if(mysql\_change\_user(conn, conf.db\_username, conf.db\_password, conf.database)) {  
 fprintf(stderr, "mysql\_change\_user() failed\n");  
 exit(EXIT\_FAILURE);  
 }  
  
 while(true) {  
 printf("\033[2J\033[H");  
 printf("\*\*\* Quale Operazione vuoi eseguire? \*\*\*\n\n");  
 printf("0) Ricerca dipendente per nome\n");  
 printf("1) Ricerca dipendente per cognome\n");  
 printf("2) Ricerca dipendente per nome e cognome\n");  
 printf("3) Ricerca postazione\n");  
 printf("4) Report Dipendenti da Trasferire\n");  
 printf("5) Report Dipendenti da Trasferire (raggruppati per Mansione)\n");  
 printf("6) Trasferimento Dipendente\n");  
 printf("7) Scambio postazione tra dipendenti\n");  
 printf("8) Inserisci dipendente\n");  
 printf("9) Elimina dipendente\n");  
 printf("X) Esci\n");  
  
  
  
  
  
 op = multiChoice("Seleziona un'opzione", options, 11);  
  
 switch(op) {  
 case '0':  
 ricercaDipNome(conn);  
 break;  
   
 case '1':  
 ricercaDipCognome(conn);  
 break;  
 case '2':  
 ricercaDipNomeCognome(conn);  
 break;  
 case '3':  
 ricercaPostazione(conn);  
 break;  
 case '4':  
 reportDipendenti(conn);  
 break;  
 case '5':  
 reportDipendentiMansione(conn);  
 break;  
 case '6':  
 trasferisciDipendente(conn);  
 break;  
 case '7':  
 scambioDipendenti(conn);  
 break;  
 case '8':  
 inserisciDipendente(conn);  
 break;  
 case '9':  
 eliminaDipendente(conn);  
 break;  
 case 'X':  
 return;  
   
 default:  
 fprintf(stderr, "Invalid condition at %s:%d\n", \_\_FILE\_\_, \_\_LINE\_\_);  
 abort();  
 }  
  
 getchar();  
 }  
}

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

-setAmministrativo.c

#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
  
#include "defines.h"  
  
static void modificaMansione(MYSQL \*conn)  
{  
 MYSQL\_STMT \*prepared\_stmt;  
 MYSQL\_BIND param[2];  
  
 // Prepare stored procedure call  
 if(!setup\_prepared\_stmt(&prepared\_stmt, "call ModificaMansione(?, ?)", conn)) {  
 finish\_with\_stmt\_error(conn, prepared\_stmt, "Unable to initialize 'ModificaMansione' statement\n", false);  
 }  
  
 // Prepare parameters  
 memset(param, 0, sizeof(param));  
  
 param[0].buffer\_type = *MYSQL\_TYPE\_VAR\_STRING*;  
 param[1].buffer\_type = *MYSQL\_TYPE\_VAR\_STRING*;  
 char cf[16];  
 char mansione[45];  
 printf("Inserisci il codice fiscale del dipendente a cui si vuole modificare la mansione assegnata\n");  
 getInput(16, cf, false);  
 printf("Inserisci la nuova mansione: ");  
 getInput(45, mansione, false);  
 param[0].buffer = cf;  
 param[1].buffer = mansione;  
 param[0].buffer\_length = strlen((char \*)param[0].buffer);  
 param[1].buffer\_length = strlen((char \*)param[1].buffer);  
  
  
 if (mysql\_stmt\_bind\_param(prepared\_stmt, param) != 0) {  
 finish\_with\_stmt\_error(conn, prepared\_stmt, "Could not bind parameters for 'ModificaMansione'\n", true);  
 }  
  
 // Run procedure  
 if (mysql\_stmt\_execute(prepared\_stmt) != 0) {  
 print\_stmt\_error(prepared\_stmt, "An error occurred while changing employee's job.");  
 goto out;  
 }  
  
 //non produce un result set  
 printf("\nMansione assegnata aggiornata con successo.\nIl dipendente verrà ora segnalato come da trasferire nei report del Settore Spazi.");  
  
 out:  
 mysql\_stmt\_close(prepared\_stmt);  
}  
  
static void mostraStoricoUffici(MYSQL \*conn)  
{  
 MYSQL\_STMT \*prepared\_stmt;  
 MYSQL\_BIND param[1];  
  
 // Prepare stored procedure call  
 if(!setup\_prepared\_stmt(&prepared\_stmt, "call StoricoUffici(?)", conn)) {  
 finish\_with\_stmt\_error(conn, prepared\_stmt, "Unable to initialize 'StoricoUffici' statement\n", false);  
 }  
  
 // Prepare parameters  
 memset(param, 0, sizeof(param));  
  
 param[0].buffer\_type = *MYSQL\_TYPE\_VAR\_STRING*;  
 printf("Inserisci il codice fiscale del dipendente di cui si vuole ottenere lo storico degli uffici\n");  
 char string[16];  
 getInput(16, string, false);  
 param[0].buffer = string;  
 param[0].buffer\_length = strlen((char \*)param[0].buffer);  
  
 if (mysql\_stmt\_bind\_param(prepared\_stmt, param) != 0) {  
 finish\_with\_stmt\_error(conn, prepared\_stmt, "Could not bind parameters for 'StroicoUffici't\n", true);  
 }  
  
 // Run procedure  
 if (mysql\_stmt\_execute(prepared\_stmt) != 0) {  
 print\_stmt\_error(prepared\_stmt, "An error occurred while retrieving the offices report.");  
 goto out;  
 }  
  
  
  
 do {  
 dump\_result\_set(conn, prepared\_stmt, "\nUffici occupati nel tempo:\n");  
 } while (mysql\_stmt\_next\_result(prepared\_stmt) == 0);//mi assicuro di consumare tutto il result set, altrimenti potrei ottenere "out of sync"  
  
 out:  
 mysql\_stmt\_close(prepared\_stmt);  
}  
  
  
void runAsAmministrativo(MYSQL \*conn)  
{  
 char options[7] = {'1','2', '3', '4', '5', '6', '7'};  
 char op;  
   
 printf("Passaggio ad Account del Settore Amministrativo...\n");  
  
 if(!parse\_config("users/setAmministrativo.json", &conf)) {  
 fprintf(stderr, "Unable to load administrator configuration\n");  
 exit(EXIT\_FAILURE);  
 }  
  
 if(mysql\_change\_user(conn, conf.db\_username, conf.db\_password, conf.database)) {  
 fprintf(stderr, "mysql\_change\_user() failed\n");  
 exit(EXIT\_FAILURE);  
 }  
  
 while(true) {  
 printf("\033[2J\033[H");  
 printf("\*\*\* Quale operazione vuoi eseguire? \*\*\*\n\n");  
 printf("1) Modifica la mansione assegnata a un dipendente\n");  
 printf("2) Mostra lo storico degli uffici di un dipendente\n");  
 printf("3) Ricerca dipendente per nome\n");  
 printf("4) Ricerca dipendente per cognome\n");  
 printf("5) Ricerca dipendente per nome e cognome\n");  
 printf("6) Ricerca postazione\n");  
 printf("7) Esci\n");  
  
 op = multiChoice("Select an option", options, 7);  
  
 switch(op) {  
 case '1':  
 modificaMansione(conn);  
 break;  
 case '2':  
 mostraStoricoUffici(conn);  
 break;  
 case '3':  
 ricercaDipNome(conn);  
 break;  
 case '4':  
 ricercaDipCognome(conn);  
 break;  
 case '5':  
 ricercaDipNomeCognome(conn);  
 break;  
 case '6':  
 ricercaPostazione(conn);  
 break;  
 case '7':  
 return;  
   
 default:  
 fprintf(stderr, "Invalid condition at %s:%d\n", \_\_FILE\_\_, \_\_LINE\_\_);  
 abort();  
 }  
  
 getchar();  
 }  
}

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

-inout.c

#include <unistd.h>  
#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
#include <ctype.h>  
#include <termios.h>  
#include <sys/ioctl.h>  
#include <pthread.h>  
#include <signal.h>  
#include <stdbool.h>  
  
#include "defines.h"  
  
// Per la gestione dei segnali  
static volatile sig\_atomic\_t signo;  
typedef struct sigaction sigaction\_t;  
static void handler(int s);  
  
char \*getInput(unsigned int lung, char \*stringa, bool hide)  
{  
 char c;  
 unsigned int i;  
  
 // Dichiara le variabili necessarie ad un possibile mascheramento dell'input  
 sigaction\_t sa, savealrm, saveint, savehup, savequit, saveterm;  
 sigaction\_t savetstp, savettin, savettou;  
 struct termios term, oterm;  
  
 if(hide) {  
 // Svuota il buffer  
 (void) fflush(stdout);  
  
 // Cattura i segnali che altrimenti potrebbero far terminare il programma, lasciando l'utente senza output sulla shell  
 sigemptyset(&sa.sa\_mask);  
 sa.sa\_flags = SA\_INTERRUPT; // Per non resettare le system call  
 sa.sa\_handler = handler;  
 (void) sigaction(SIGALRM, &sa, &savealrm);  
 (void) sigaction(SIGINT, &sa, &saveint);  
 (void) sigaction(SIGHUP, &sa, &savehup);  
 (void) sigaction(SIGQUIT, &sa, &savequit);  
 (void) sigaction(SIGTERM, &sa, &saveterm);  
 (void) sigaction(SIGTSTP, &sa, &savetstp);  
 (void) sigaction(SIGTTIN, &sa, &savettin);  
 (void) sigaction(SIGTTOU, &sa, &savettou);  
   
 // Disattiva l'output su schermo  
 if (tcgetattr(fileno(stdin), &oterm) == 0) {  
 (void) memcpy(&term, &oterm, sizeof(struct termios));  
 term.c\_lflag &= ~(ECHO|ECHONL);  
 (void) tcsetattr(fileno(stdin), TCSAFLUSH, &term);  
 } else {  
 (void) memset(&term, 0, sizeof(struct termios));  
 (void) memset(&oterm, 0, sizeof(struct termios));  
 }  
 }  
   
 // Acquisisce da tastiera al più lung - 1 caratteri  
 for(i = 0; i < lung; i++) {  
 (void) fread(&c, sizeof(char), 1, stdin);  
 if(c == '\n') {  
 stringa[i] = '\0';  
 break;  
 } else  
 stringa[i] = c;  
  
 // Gestisce gli asterischi  
 if(hide) {  
 if(c == '\b') // Backspace  
 (void) write(fileno(stdout), &c, sizeof(char));  
 else  
 (void) write(fileno(stdout), "\*", sizeof(char));  
 }  
 }  
   
 // Controlla che il terminatore di stringa sia stato inserito  
 if(i == lung - 1)  
 stringa[i] = '\0';  
  
 // Se sono stati digitati più caratteri, svuota il buffer della tastiera  
 if(strlen(stringa) >= lung) {   
 // Svuota il buffer della tastiera  
 do {  
 c = getchar();  
 } while (c != '\n');  
 }  
  
 if(hide) {  
 //L'a capo dopo l'input  
 (void) write(fileno(stdout), "\n", 1);  
  
 // Ripristina le impostazioni precedenti dello schermo  
 (void) tcsetattr(fileno(stdin), TCSAFLUSH, &oterm);  
  
 // Ripristina la gestione dei segnali  
 (void) sigaction(SIGALRM, &savealrm, NULL);  
 (void) sigaction(SIGINT, &saveint, NULL);  
 (void) sigaction(SIGHUP, &savehup, NULL);  
 (void) sigaction(SIGQUIT, &savequit, NULL);  
 (void) sigaction(SIGTERM, &saveterm, NULL);  
 (void) sigaction(SIGTSTP, &savetstp, NULL);  
 (void) sigaction(SIGTTIN, &savettin, NULL);  
 (void) sigaction(SIGTTOU, &savettou, NULL);  
  
 // Se era stato ricevuto un segnale viene rilanciato al processo stesso  
 if(signo)  
 (void) raise(signo);  
 }  
   
 return stringa;  
}  
  
// Per la gestione dei segnali  
static void handler(int s) {  
 signo = s;  
}  
  
  
  
char multiChoice(char \*domanda, char choices[], int num)  
{  
  
 // Genera la stringa delle possibilità  
 char \*possib = malloc(2 \* num \* sizeof(char));  
 int i, j = 0;  
 for(i = 0; i < num; i++) {  
 possib[j++] = choices[i];  
 possib[j++] = '/';  
 }  
 possib[j-1] = '\0'; // Per eliminare l'ultima '/'  
  
 // Chiede la risposta  
 while(true) {  
 // Mostra la domanda  
 printf("%s [%s]: ", domanda, possib);  
  
 char c;  
 getInput(1, &c, false);  
  
 // Controlla se è un carattere valido  
 for(i = 0; i < num; i++) {  
 if(c == choices[i])  
 return c;  
 }  
 }  
}  
  
void date\_to\_mysql\_time(char \*str, MYSQL\_TIME \*time)  
{  
 memset(time, 0, sizeof(\*time));  
 sscanf(str, "%4d-%2d-%2d", &time->year, &time->month, &time->day);  
 time->time\_type = *MYSQL\_TIMESTAMP\_DATE*;  
}

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

-parse.c

#include <stddef.h>  
#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
  
#include "defines.h"  
  
#define BUFF\_SIZE 4096  
  
// The final config struct will point into this  
static char config[BUFF\_SIZE];  
  
/\*\*  
 \* JSON type identifier. Basic types are:  
 \* o Object  
 \* o Array  
 \* o String  
 \* o Other primitive: number, boolean (true/false) or null  
 \*/  
typedef enum {  
 *JSMN\_UNDEFINED* = 0,  
 *JSMN\_OBJECT* = 1,  
 *JSMN\_ARRAY* = 2,  
 *JSMN\_STRING* = 3,  
 *JSMN\_PRIMITIVE* = 4  
} jsmntype\_t;  
  
enum jsmnerr {  
 /\* Not enough tokens were provided \*/  
 *JSMN\_ERROR\_NOMEM* = -1,  
 /\* Invalid character inside JSON string \*/  
 *JSMN\_ERROR\_INVAL* = -2,  
 /\* The string is not a full JSON packet, more bytes expected \*/  
 *JSMN\_ERROR\_PART* = -3  
};  
  
/\*\*  
 \* JSON token description.  
 \* type type (object, array, string etc.)  
 \* start start position in JSON data string  
 \* end end position in JSON data string  
 \*/  
typedef struct {  
 jsmntype\_t type;  
 int start;  
 int end;  
 int size;  
#ifdef JSMN\_PARENT\_LINKS  
 int parent;  
#endif  
} jsmntok\_t;  
  
/\*\*  
 \* JSON parser. Contains an array of token blocks available. Also stores  
 \* the string being parsed now and current position in that string  
 \*/  
typedef struct {  
 unsigned int pos; /\* offset in the JSON string \*/  
 unsigned int toknext; /\* next token to allocate \*/  
 int toksuper; /\* superior token node, e.g parent object or array \*/  
} jsmn\_parser;  
  
/\*\*  
 \* Allocates a fresh unused token from the token pool.  
 \*/  
static jsmntok\_t \*jsmn\_alloc\_token(jsmn\_parser \*parser, jsmntok\_t \*tokens, size\_t num\_tokens) {  
 jsmntok\_t \*tok;  
 if (parser->toknext >= num\_tokens) {  
 return NULL;  
 }  
 tok = &tokens[parser->toknext++];  
 tok->start = tok->end = -1;  
 tok->size = 0;  
#ifdef JSMN\_PARENT\_LINKS  
 tok->parent = -1;  
#endif  
 return tok;  
}  
  
/\*\*  
 \* Fills token type and boundaries.  
 \*/  
static void jsmn\_fill\_token(jsmntok\_t \*token, jsmntype\_t type,  
 int start, int end) {  
 token->type = type;  
 token->start = start;  
 token->end = end;  
 token->size = 0;  
}  
  
/\*\*  
 \* Fills next available token with JSON primitive.  
 \*/  
static int jsmn\_parse\_primitive(jsmn\_parser \*parser, const char \*js,  
 size\_t len, jsmntok\_t \*tokens, size\_t num\_tokens) {  
 jsmntok\_t \*token;  
 int start;  
  
 start = parser->pos;  
  
 for (; parser->pos < len && js[parser->pos] != '\0'; parser->pos++) {  
 switch (js[parser->pos]) {  
#ifndef JSMN\_STRICT  
 /\* In strict mode primitive must be followed by "," or "}" or "]" \*/  
 case ':':  
#endif  
 case '\t' : case '\r' : case '\n' : case ' ' :  
 case ',' : case ']' : case '}' :  
 goto found;  
 }  
 if (js[parser->pos] < 32 || js[parser->pos] >= 127) {  
 parser->pos = start;  
 return *JSMN\_ERROR\_INVAL*;  
 }  
 }  
#ifdef JSMN\_STRICT  
 /\* In strict mode primitive must be followed by a comma/object/array \*/  
 parser->pos = start;  
 return JSMN\_ERROR\_PART;  
#endif  
  
found:  
 if (tokens == NULL) {  
 parser->pos--;  
 return 0;  
 }  
 token = jsmn\_alloc\_token(parser, tokens, num\_tokens);  
 if (token == NULL) {  
 parser->pos = start;  
 return *JSMN\_ERROR\_NOMEM*;  
 }  
 jsmn\_fill\_token(token, *JSMN\_PRIMITIVE*, start, parser->pos);  
#ifdef JSMN\_PARENT\_LINKS  
 token->parent = parser->toksuper;  
#endif  
 parser->pos--;  
 return 0;  
}  
  
/\*\*  
 \* Fills next token with JSON string.  
 \*/  
static int jsmn\_parse\_string(jsmn\_parser \*parser, const char \*js,  
 size\_t len, jsmntok\_t \*tokens, size\_t num\_tokens) {  
 jsmntok\_t \*token;  
  
 int start = parser->pos;  
  
 parser->pos++;  
  
 /\* Skip starting quote \*/  
 for (; parser->pos < len && js[parser->pos] != '\0'; parser->pos++) {  
 char c = js[parser->pos];  
  
 /\* Quote: end of string \*/  
 if (c == '\"') {  
 if (tokens == NULL) {  
 return 0;  
 }  
 token = jsmn\_alloc\_token(parser, tokens, num\_tokens);  
 if (token == NULL) {  
 parser->pos = start;  
 return *JSMN\_ERROR\_NOMEM*;  
 }  
 jsmn\_fill\_token(token, *JSMN\_STRING*, start+1, parser->pos);  
#ifdef JSMN\_PARENT\_LINKS  
 token->parent = parser->toksuper;  
#endif  
 return 0;  
 }  
  
 /\* Backslash: Quoted symbol expected \*/  
 if (c == '\\' && parser->pos + 1 < len) {  
 int i;  
 parser->pos++;  
 switch (js[parser->pos]) {  
 /\* Allowed escaped symbols \*/  
 case '\"': case '/' : case '\\' : case 'b' :  
 case 'f' : case 'r' : case 'n' : case 't' :  
 break;  
 /\* Allows escaped symbol \uXXXX \*/  
 case 'u':  
 parser->pos++;  
 for(i = 0; i < 4 && parser->pos < len && js[parser->pos] != '\0'; i++) {  
 /\* If it isn't a hex character we have an error \*/  
 if(!((js[parser->pos] >= 48 && js[parser->pos] <= 57) || /\* 0-9 \*/  
 (js[parser->pos] >= 65 && js[parser->pos] <= 70) || /\* A-F \*/  
 (js[parser->pos] >= 97 && js[parser->pos] <= 102))) { /\* a-f \*/  
 parser->pos = start;  
 return *JSMN\_ERROR\_INVAL*;  
 }  
 parser->pos++;  
 }  
 parser->pos--;  
 break;  
 /\* Unexpected symbol \*/  
 default:  
 parser->pos = start;  
 return *JSMN\_ERROR\_INVAL*;  
 }  
 }  
 }  
 parser->pos = start;  
 return *JSMN\_ERROR\_PART*;  
}  
  
/\*\*  
 \* Parse JSON string and fill tokens.  
 \*/  
static int jsmn\_parse(jsmn\_parser \*parser, const char \*js, size\_t len, jsmntok\_t \*tokens, unsigned int num\_tokens) {  
 int r;  
 int i;  
 jsmntok\_t \*token;  
 int count = parser->toknext;  
  
 for (; parser->pos < len && js[parser->pos] != '\0'; parser->pos++) {  
 char c;  
 jsmntype\_t type;  
  
 c = js[parser->pos];  
 switch (c) {  
 case '{': case '[':  
 count++;  
 if (tokens == NULL) {  
 break;  
 }  
 token = jsmn\_alloc\_token(parser, tokens, num\_tokens);  
 if (token == NULL)  
 return *JSMN\_ERROR\_NOMEM*;  
 if (parser->toksuper != -1) {  
 tokens[parser->toksuper].size++;  
#ifdef JSMN\_PARENT\_LINKS  
 token->parent = parser->toksuper;  
#endif  
 }  
 token->type = (c == '{' ? *JSMN\_OBJECT* : *JSMN\_ARRAY*);  
 token->start = parser->pos;  
 parser->toksuper = parser->toknext - 1;  
 break;  
 case '}': case ']':  
 if (tokens == NULL)  
 break;  
 type = (c == '}' ? *JSMN\_OBJECT* : *JSMN\_ARRAY*);  
#ifdef JSMN\_PARENT\_LINKS  
 if (parser->toknext < 1) {  
 return JSMN\_ERROR\_INVAL;  
 }  
 token = &tokens[parser->toknext - 1];  
 for (;;) {  
 if (token->start != -1 && token->end == -1) {  
 if (token->type != type) {  
 return JSMN\_ERROR\_INVAL;  
 }  
 token->end = parser->pos + 1;  
 parser->toksuper = token->parent;  
 break;  
 }  
 if (token->parent == -1) {  
 if(token->type != type || parser->toksuper == -1) {  
 return JSMN\_ERROR\_INVAL;  
 }  
 break;  
 }  
 token = &tokens[token->parent];  
 }  
#else  
 for (i = parser->toknext - 1; i >= 0; i--) {  
 token = &tokens[i];  
 if (token->start != -1 && token->end == -1) {  
 if (token->type != type) {  
 return *JSMN\_ERROR\_INVAL*;  
 }  
 parser->toksuper = -1;  
 token->end = parser->pos + 1;  
 break;  
 }  
 }  
 /\* Error if unmatched closing bracket \*/  
 if (i == -1) return *JSMN\_ERROR\_INVAL*;  
 for (; i >= 0; i--) {  
 token = &tokens[i];  
 if (token->start != -1 && token->end == -1) {  
 parser->toksuper = i;  
 break;  
 }  
 }  
#endif  
 break;  
 case '\"':  
 r = jsmn\_parse\_string(parser, js, len, tokens, num\_tokens);  
 if (r < 0) return r;  
 count++;  
 if (parser->toksuper != -1 && tokens != NULL)  
 tokens[parser->toksuper].size++;  
 break;  
 case '\t' : case '\r' : case '\n' : case ' ':  
 break;  
 case ':':  
 parser->toksuper = parser->toknext - 1;  
 break;  
 case ',':  
 if (tokens != NULL && parser->toksuper != -1 &&  
 tokens[parser->toksuper].type != *JSMN\_ARRAY* &&  
 tokens[parser->toksuper].type != *JSMN\_OBJECT*) {  
#ifdef JSMN\_PARENT\_LINKS  
 parser->toksuper = tokens[parser->toksuper].parent;  
#else  
 for (i = parser->toknext - 1; i >= 0; i--) {  
 if (tokens[i].type == *JSMN\_ARRAY* || tokens[i].type == *JSMN\_OBJECT*) {  
 if (tokens[i].start != -1 && tokens[i].end == -1) {  
 parser->toksuper = i;  
 break;  
 }  
 }  
 }  
#endif  
 }  
 break;  
#ifdef JSMN\_STRICT  
 /\* In strict mode primitives are: numbers and booleans \*/  
 case '-': case '0': case '1' : case '2': case '3' : case '4':  
 case '5': case '6': case '7' : case '8': case '9':  
 case 't': case 'f': case 'n' :  
 /\* And they must not be keys of the object \*/  
 if (tokens != NULL && parser->toksuper != -1) {  
 jsmntok\_t \*t = &tokens[parser->toksuper];  
 if (t->type == JSMN\_OBJECT ||  
 (t->type == JSMN\_STRING && t->size != 0)) {  
 return JSMN\_ERROR\_INVAL;  
 }  
 }  
#else  
 /\* In non-strict mode every unquoted value is a primitive \*/  
 default:  
#endif  
 r = jsmn\_parse\_primitive(parser, js, len, tokens, num\_tokens);  
 if (r < 0) return r;  
 count++;  
 if (parser->toksuper != -1 && tokens != NULL)  
 tokens[parser->toksuper].size++;  
 break;  
  
#ifdef JSMN\_STRICT  
 /\* Unexpected char in strict mode \*/  
 default:  
 return JSMN\_ERROR\_INVAL;  
#endif  
 }  
 }  
  
 if (tokens != NULL) {  
 for (i = parser->toknext - 1; i >= 0; i--) {  
 /\* Unmatched opened object or array \*/  
 if (tokens[i].start != -1 && tokens[i].end == -1) {  
 return *JSMN\_ERROR\_PART*;  
 }  
 }  
 }  
  
 return count;  
}  
  
/\*\*  
 \* Creates a new parser based over a given buffer with an array of tokens  
 \* available.  
 \*/  
static void jsmn\_init(jsmn\_parser \*parser) {  
 parser->pos = 0;  
 parser->toknext = 0;  
 parser->toksuper = -1;  
}  
  
static int jsoneq(const char \*json, jsmntok\_t \*tok, const char \*s)  
{  
 if (tok->type == *JSMN\_STRING*  
&& (int) strlen(s) == tok->end - tok->start  
 && strncmp(json + tok->start, s, tok->end - tok->start) == 0) {  
 return 0;  
 }  
 return -1;  
}  
  
static size\_t load\_file(char \*filename)  
{  
 FILE \*f = fopen(filename, "rb");  
 if(f == NULL) {  
 fprintf(stderr, "Unable to open file %s\n", filename);  
 exit(1);  
 }  
   
 fseek(f, 0, SEEK\_END);  
 size\_t fsize = ftell(f);  
 fseek(f, 0, SEEK\_SET); //same as rewind(f);  
  
 if(fsize >= BUFF\_SIZE) {  
 fprintf(stderr, "Configuration file too large\n");  
 abort();  
 }  
   
 fread(config, fsize, 1, f);  
 fclose(f);  
   
 config[fsize] = 0;  
 return fsize;  
}  
  
int parse\_config(char \*path, struct configuration \*conf)  
{  
 int i;  
 int r;  
 jsmn\_parser p;  
 jsmntok\_t t[128]; /\* We expect no more than 128 tokens \*/  
  
 load\_file(path);  
  
 jsmn\_init(&p);  
 r = jsmn\_parse(&p, config, strlen(config), t, sizeof(t)/sizeof(t[0]));  
 if (r < 0) {  
 printf("Failed to parse JSON: %d\n", r);  
 return 0;  
 }  
  
 /\* Assume the top-level element is an object \*/  
 if (r < 1 || t[0].type != *JSMN\_OBJECT*) {  
 printf("Object expected\n");  
 return 0;  
 }  
  
 /\* Loop over all keys of the root object \*/  
 for (i = 1; i < r; i++) {  
 if (jsoneq(config, &t[i], "host") == 0) {  
 /\* We may use strndup() to fetch string value \*/  
 conf->host = strndup(config + t[i+1].start, t[i+1].end-t[i+1].start);  
 i++;  
 } else if (jsoneq(config, &t[i], "username") == 0) {  
 conf->db\_username = strndup(config + t[i+1].start, t[i+1].end-t[i+1].start);  
 i++;  
 } else if (jsoneq(config, &t[i], "password") == 0) {  
 conf->db\_password = strndup(config + t[i+1].start, t[i+1].end-t[i+1].start);  
 i++;  
 } else if (jsoneq(config, &t[i], "port") == 0) {  
 conf->port = strtol(config + t[i+1].start, NULL, 10);  
 i++;  
 } else if (jsoneq(config, &t[i], "database") == 0) {  
 conf->database = strndup(config + t[i+1].start, t[i+1].end-t[i+1].start);  
 i++;  
 } else {  
 printf("Unexpected key: %.\*s\n", t[i].end-t[i].start, config + t[i].start);  
 }  
 }  
 return 1;  
}

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

-utils.c

#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
  
#include "defines.h"  
  
void print\_stmt\_error (MYSQL\_STMT \*stmt, char \*message)  
{  
 fprintf (stderr, "%s\n", message);  
 if (stmt != NULL) {  
 fprintf (stderr, "Error %u (%s): %s\n",  
 mysql\_stmt\_errno (stmt),  
 mysql\_stmt\_sqlstate(stmt),  
 mysql\_stmt\_error (stmt));  
 }  
}  
  
  
void print\_error(MYSQL \*conn, char \*message)  
{  
 fprintf (stderr, "%s\n", message);  
 if (conn != NULL) {  
 #if MYSQL\_VERSION\_ID >= 40101  
 fprintf (stderr, "Error %u (%s): %s\n",  
 mysql\_errno (conn), mysql\_sqlstate(conn), mysql\_error (conn));  
 #else  
 fprintf (stderr, "Error %u: %s\n",  
 mysql\_errno (conn), mysql\_error (conn));  
 #endif  
 }  
}  
  
bool setup\_prepared\_stmt(MYSQL\_STMT \*\*stmt, char \*statement, MYSQL \*conn)  
{  
 bool update\_length = true;  
  
 \*stmt = mysql\_stmt\_init(conn);  
 if (\*stmt == NULL)  
 {  
 print\_error(conn, "Could not initialize statement handler");  
 return false;  
 }  
  
 if (mysql\_stmt\_prepare (\*stmt, statement, strlen(statement)) != 0) {  
 print\_stmt\_error(\*stmt, "Could not prepare statement");  
 return false;  
 }  
  
 mysql\_stmt\_attr\_set(\*stmt, *STMT\_ATTR\_UPDATE\_MAX\_LENGTH*, &update\_length);  
  
 return true;  
}  
  
void finish\_with\_error(MYSQL \*conn, char \*message)  
{  
 print\_error(conn, message);  
 mysql\_close(conn);  
 exit(EXIT\_FAILURE);   
}  
  
void finish\_with\_stmt\_error(MYSQL \*conn, MYSQL\_STMT \*stmt, char \*message, bool close\_stmt)  
{  
 print\_stmt\_error(stmt, message);  
 if(close\_stmt) mysql\_stmt\_close(stmt);  
 mysql\_close(conn);  
 exit(EXIT\_FAILURE);   
}  
  
static void print\_dashes(MYSQL\_RES \*res\_set)  
{  
 MYSQL\_FIELD \*field;  
 unsigned int i, j;  
  
 mysql\_field\_seek(res\_set, 0);  
 putchar('+');  
 for (i = 0; i < mysql\_num\_fields(res\_set); i++) {  
 field = mysql\_fetch\_field(res\_set);  
 for (j = 0; j < field->max\_length + 2; j++)  
 putchar('-');  
 putchar('+');  
 }  
 putchar('\n');  
}  
  
static void dump\_result\_set\_header(MYSQL\_RES \*res\_set)  
{  
 MYSQL\_FIELD \*field;  
 unsigned long col\_len;  
 unsigned int i;  
  
 /\* determine column display widths -- requires result set to be \*/  
 /\* generated with mysql\_store\_result(), not mysql\_use\_result() \*/  
  
 mysql\_field\_seek (res\_set, 0);  
  
 for (i = 0; i < mysql\_num\_fields (res\_set); i++) {  
 field = mysql\_fetch\_field (res\_set);  
 col\_len = strlen(field->name);  
  
 if (col\_len < field->max\_length)  
 col\_len = field->max\_length;  
 if (col\_len < 4 && !IS\_NOT\_NULL(field->flags))  
 col\_len = 4; /\* 4 = length of the word "NULL" \*/  
 field->max\_length = col\_len; /\* reset column info \*/  
 }  
   
 print\_dashes(res\_set);  
 putchar('|');  
 mysql\_field\_seek (res\_set, 0);  
 for (i = 0; i < mysql\_num\_fields(res\_set); i++) {  
 field = mysql\_fetch\_field(res\_set);  
 printf(" %-\*s |", (int)field->max\_length, field->name);  
 }  
 putchar('\n');  
  
 print\_dashes(res\_set);  
}  
  
void dump\_result\_set(MYSQL \*conn, MYSQL\_STMT \*stmt, char \*title)  
{  
 int i;  
 int status;  
 int num\_fields; /\* number of columns in result \*/  
 MYSQL\_FIELD \*fields; /\* for result set metadata \*/  
 MYSQL\_BIND \*rs\_bind; /\* for output buffers \*/  
 MYSQL\_RES \*rs\_metadata;  
 MYSQL\_TIME \*date;  
 size\_t attr\_size;  
  
 /\* Prefetch the whole result set. This in conjunction with  
 \* STMT\_ATTR\_UPDATE\_MAX\_LENGTH set in `setup\_prepared\_stmt`  
 \* updates the result set metadata which are fetched in this  
 \* function, to allow to compute the actual max length of  
 \* the columns.  
 \*/  
 if (mysql\_stmt\_store\_result(stmt)) {  
 fprintf(stderr, " mysql\_stmt\_execute(), 1 failed\n");  
 fprintf(stderr, " %s\n", mysql\_stmt\_error(stmt));  
 exit(0);  
 }   
  
 /\* the column count is > 0 if there is a result set \*/  
 /\* 0 if the result is only the final status packet \*/  
 num\_fields = mysql\_stmt\_field\_count(stmt);  
  
 if (num\_fields > 0) {  
 /\* there is a result set to fetch \*/  
 printf("%s\n", title);  
  
 if((rs\_metadata = mysql\_stmt\_result\_metadata(stmt)) == NULL) {  
 finish\_with\_stmt\_error(conn, stmt, "Unable to retrieve result metadata\n", true);  
 }  
  
 dump\_result\_set\_header(rs\_metadata);  
   
 fields = mysql\_fetch\_fields(rs\_metadata);  
  
 rs\_bind = (MYSQL\_BIND \*)malloc(sizeof (MYSQL\_BIND) \* num\_fields);  
 if (!rs\_bind) {  
 finish\_with\_stmt\_error(conn, stmt, "Cannot allocate output buffers\n", true);  
 }  
 memset(rs\_bind, 0, sizeof (MYSQL\_BIND) \* num\_fields);  
  
 /\* set up and bind result set output buffers \*/  
 for (i = 0; i < num\_fields; ++i) {  
  
 // Properly size the parameter buffer  
 switch(fields[i].type) {  
 case *MYSQL\_TYPE\_DATE*:  
 case *MYSQL\_TYPE\_TIMESTAMP*:  
 case *MYSQL\_TYPE\_DATETIME*:  
 case *MYSQL\_TYPE\_TIME*:  
 attr\_size = sizeof(MYSQL\_TIME);  
 break;  
 case *MYSQL\_TYPE\_FLOAT*:  
 attr\_size = sizeof(float);  
 break;  
 case *MYSQL\_TYPE\_DOUBLE*:  
 attr\_size = sizeof(double);  
 break;  
 case *MYSQL\_TYPE\_TINY*:  
 attr\_size = sizeof(signed char);  
 break;  
 case *MYSQL\_TYPE\_SHORT*:  
 case *MYSQL\_TYPE\_YEAR*:  
 attr\_size = sizeof(short int);  
 break;  
 case *MYSQL\_TYPE\_LONG*:  
 case *MYSQL\_TYPE\_INT24*:  
 attr\_size = sizeof(int);  
 break;  
 case *MYSQL\_TYPE\_LONGLONG*:  
 attr\_size = sizeof(int);  
 break;  
 default:  
 attr\_size = fields[i].max\_length;  
 break;  
 }  
   
 // Setup the binding for the current parameter  
 rs\_bind[i].buffer\_type = fields[i].type;  
 rs\_bind[i].buffer = malloc(attr\_size + 1);  
 rs\_bind[i].buffer\_length = attr\_size + 1;  
  
 if(rs\_bind[i].buffer == NULL) {  
 finish\_with\_stmt\_error(conn, stmt, "Cannot allocate output buffers\n", true);  
 }  
 }  
  
 if(mysql\_stmt\_bind\_result(stmt, rs\_bind)) {  
 finish\_with\_stmt\_error(conn, stmt, "Unable to bind output parameters\n", true);  
 }  
  
 /\* fetch and display result set rows \*/  
 while (true) {  
 status = mysql\_stmt\_fetch(stmt);  
  
 if (status == 1 || status == MYSQL\_NO\_DATA)  
 break;  
  
 putchar('|');  
  
 for (i = 0; i < num\_fields; i++) {  
  
 if (rs\_bind[i].is\_null\_value) {  
 printf (" %-\*s |", (int)fields[i].max\_length, "NULL");  
 continue;  
 }  
  
 switch (rs\_bind[i].buffer\_type) {  
   
 case *MYSQL\_TYPE\_VAR\_STRING*:  
 case *MYSQL\_TYPE\_DATETIME*:  
 printf(" %-\*s |", (int)fields[i].max\_length, (char\*)rs\_bind[i].buffer);  
 break;  
   
 case *MYSQL\_TYPE\_DATE*:  
 case *MYSQL\_TYPE\_TIMESTAMP*:  
 date = (MYSQL\_TIME \*)rs\_bind[i].buffer;  
 printf(" %d-%02d-%02d |", date->year, date->month, date->day);  
 break;  
   
 case *MYSQL\_TYPE\_STRING*:  
 printf(" %-\*s |", (int)fields[i].max\_length, (char \*)rs\_bind[i].buffer);  
 break;  
   
 case *MYSQL\_TYPE\_FLOAT*:  
 case *MYSQL\_TYPE\_DOUBLE*:  
 printf(" %.02f |", \*(float \*)rs\_bind[i].buffer);  
 break;  
   
 case *MYSQL\_TYPE\_LONG*:  
 case *MYSQL\_TYPE\_SHORT*:  
 case *MYSQL\_TYPE\_TINY*:  
 printf(" %-\*d |", (int)fields[i].max\_length, \*(int \*)rs\_bind[i].buffer);  
 break;  
   
 case *MYSQL\_TYPE\_NEWDECIMAL*:  
 printf(" %-\*.02lf |", (int)fields[i].max\_length, \*(float\*) rs\_bind[i].buffer);  
 break;  
   
 default:  
 printf("ERROR: Unhandled type (%d)\n", rs\_bind[i].buffer\_type);  
 abort();  
 }  
 }  
 putchar('\n');  
 print\_dashes(rs\_metadata);  
 }  
  
 mysql\_free\_result(rs\_metadata); /\* free metadata \*/  
  
 /\* free output buffers \*/  
 for (i = 0; i < num\_fields; i++) {  
 free(rs\_bind[i].buffer);  
 }  
 free(rs\_bind);  
 }  
}

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

-dipendente.json

{  
 "host": "127.0.0.1",  
 "username": "Dipendente",  
 "password": "dipendente",  
 "port": 3306,  
 "database": "Directory\_Aziendale"  
}

setAmministrativo.json

{  
 "host": "127.0.0.1",  
 "username": "DipendenteSettoreAmministrativo",  
 "password": "amministrativo",  
 "port": 3306,  
 "database": "Directory\_Aziendale"  
}

setSpazi.json

{  
 "host": "127.0.0.1",  
 "username": "DipendenteSettoreSpazi",  
 "password": "spazi",  
 "port": 3306,  
 "database": "Directory\_Aziendale"  
}

login.json

{  
 "host": "127.0.0.1",  
 "username": "login",  
 "password": "login",  
 "port": 3306,  
 "database": "Directory\_Aziendale"  
}

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)