Basi di Dati e Conoscenza

Progetto A.A. 2019/2020

DIRECTORY AZIENDALE

0253508

Daniele La Prova

Indice

1. Descrizione del Minimondo

Realizzare un servizio di “directory aziendale”, che consenta di tenere traccia di tutte le informazioni legate ai recapiti ed alle mansioni di un’azienda.
Ogni dipendente dell’azienda, identificato da codice fiscale, nome, cognome, data di nascita, luogo di nascita, indirizzo di residenza svolge una particolare mansione all’interno dell’azienda.
Tali mansioni vengono svolte in differenti uffici dell’azienda. L’azienda ha a disposizione un numero arbitrario di edifici, ciascuno di un numero arbitrario di piani. In ciascun piano può esserci un numero arbitrario di uffici. Ciascun ufficio è assegnato ad una mansione specifica dell’azienda. Un ufficio ha a disposizione un certo numero di postazioni, ciascuno associato ad un numero telefonico interno ed esterno. La gestione degli uffici, piani, edifici è demandata al Settore Spazi dell’azienda, anch’esso composto da dipendenti assegnati a particolari uffici.
L’assegnazione di un dipendente ad un ufficio avviene su base di turnazioni periodiche. Ogni dipendente, pertanto, è associato alla data dell’ultimo trasferimento.
Periodicamente, il sistema indica qual è l’insieme dei dipendenti che deve essere soggetto a trasferimento. Su base periodica, l’ufficio spazi genera un report indicante tutti i dipendenti, raggruppati per mansione, che devono essere spostati.
Un dipendente dell’ufficio spazi deve poter effettuare un trasferimento di dipendente. 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.

2. Analisi dei Requisiti

Identificazione dei termini ambigui e correzioni possibili

Linea	Termine	Nuovo termine	Motivo correzione
13	Assegnazione a Ufficio	Assegnazione a Postazione	Un dipendente è in verità associato a una postazione all'interno di un ufficio, più che a un ufficio
16	Ufficio spazi	Settore spazi	Ambiguità: Settore Spazi composto da dipendenti assegnati a particolari uffici, ma riferito anche come "ufficio" spazi? Interpretazione: Un Settore rappresenta una sovrastruttura che accomuna dipendenti di diversi uffici con mansioni simili. In questo caso, il ruolo è quello del Settore Spazi. Dunque, dipendenti con mansioni diverse associati a uffici diversi possono rientrare nello stesso settore. In questo caso, chi stila materialmente il report sono uno o più uffici del Settore Spazi. Il report risulta però generato non da uno specifico ufficio, ma dall'intero Settore.
30	Ubicazione	Postazione	Esiste una associazione (1,1) tra Postazione e Dipendente, perciò l'ubicazione di ogni dipendente è individuata dalla sua postazione.
31	Numero di telefono	Numero di telefono esterno	Dal numero di telefono bisogna potersi ricavare l'ufficio, il piano e l'edificio corrispondenti. Se il numero di telefono fosse interno all'ufficio, allora non sarebbe possibile rintracciare l'ufficio corrispondente. Pertanto, è necessario fornire il numero di telefono esterno.
37	E-mail d'ufficio a cui fa capo		Il termine ufficio qui è stato interpretato come sinonimo di Mansione, mentre nel resto del testo può essere interpretato come ufficio fisico. Sarebbe più corretto parlare dunque di Mansione, ma dato che suonerebbe "strano" parlare di un e-mail di Mansione è stato scelto di presevare la scelta del termine, nonostante l'ambiguità.

Specifica disambiguata

Realizzare un servizio di “directory aziendale”, che consenta di tenere traccia di tutte le informazioni legate ai recapiti ed alle mansioni di un’azienda.
Ogni dipendente dell’azienda, identificato da codice fiscale, nome, cognome, data di nascita, luogo di nascita, indirizzo di residenza svolge una particolare mansione all’interno dell’azienda.
Tali mansioni vengono svolte in differenti uffici dell’azienda. L’azienda ha a disposizione un numero arbitrario di edifici, ciascuno di un numero arbitrario di piani. In ciascun piano può esserci un numero arbitrario di uffici. Ciascun ufficio è assegnato ad una mansione specifica dell’azienda. Un ufficio ha a disposizione un certo numero di postazioni, ciascuno associato ad un numero telefonico interno ed esterno. La gestione degli uffici, piani, edifici è demandata al Settore Spazi dell’azienda, anch’esso composto da dipendenti assegnati a particolari uffici.
L’assegnazione di un dipendente ad **una postazione** 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, **il Settore Spazi** genera un report indicante tutti i dipendenti, raggruppati per mansione, che devono essere spostati.
Un dipendente **del Settore 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 **postazione** 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 email personale ed uno associato all'ufficio a cui fa capo.

Glossario dei Termini

Termine	Descrizione	Sinonimi	Collegamenti
Dipendente	Un membro lavoratore dell'azienda, per cui svolge una particolare Mansione	(Null)	Postazione, Ufficio, Mansione
Mansione	Attività svolta dai Dipendenti di un Ufficio	(Null)	Ufficio, Dipendente
Settore	Aggregazione di diversi uffici associati a Mansioni simili, per esempio finalizzate a uno scopo comune.	(Null)	Ufficio
Postazione	Assegnazione di un Dipendente a un Ufficio	Ubicazione	Dipendente, Ufficio
Ufficio	Partizione di Postazioni occupate da Dipendenti dedicati a una stessa Mansione, localizzati in un particolare Piano di un Edificio	(Null)	Settore, Mansione, Postazione, Piano.
Piano	Partizione degli Uffici situati in uno stesso edifico	(Null)	Ufficio, Edificio
Edificio	Partizione degli Uffici dell'azienda	(Null)	Piano
Trasferimento	Un dipendente è soggetto a trasferimento se viene assegnato a una nuova Postazione.	(Null)	Dipendente, Postazione

Raggruppamento dei requisiti in insiemi omogenei

NB: Frasi relative a più di una parola chiave sono riportate all'interno della tabella riguardo a una sola delle parole chiave coinvolte.

Frasi relative a Dipendente
Ogni dipendente dell'azienda, identificato da codice fiscale, nome, cognome, data di nascita, luogo di nascita, indirizzo di residenza svolge una particolare mansione all'interno dell'azienda.
Ogni dipendente 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 postazione 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.
Un dipendente ha un indirizzo email personale ed uno associato all'ufficio a cui fa capo.
Un dipendente dell settore spazi deve poter effettuare un trasferimento di dipendente.
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 Mansione
Tali mansioni vengono svolte in differenti uffici dell'azienda.

5

Frasi relative a Settore
La gestione degli uffici, piani, edifici è demandata al Settore Spazi dell'azienda, anch'esso composto da dipendenti assegnati a particolari uffici.
Su base periodica, il settore spazi genera un report indicante tutti i dipendenti, raggruppati per mansione, che devono essere spostati.
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.

Frasi relative a Postazione
Ogni posttazione è associata ad un numero telefonico interno ed esterno.

Frasi relative a Ufficio
Ciascun ufficio è assegnato ad una mansione specifica dell'azienda
Un ufficio ha a disposizione un certo numero di postazioni

Frasi relative a Piano
In ciascun piano può esserci un numero arbitrario di uffici

Frasi relative a Edificio
L’azienda ha a disposizione un numero arbitrario di edifici, ciascuno di un numero arbitrario di piani.

Frasi relative a Trasferimento
L’assegnazione di un dipendente ad un ufficio avviene su base di turnazioni periodiche.
Periodicamente, il sistema indica qual è l’insieme dei dipendenti che deve essere soggetto a trasferimento.
Tale trasferimento deve essere effettuato rispettando l’associazione tra le mansioni e gli uffici

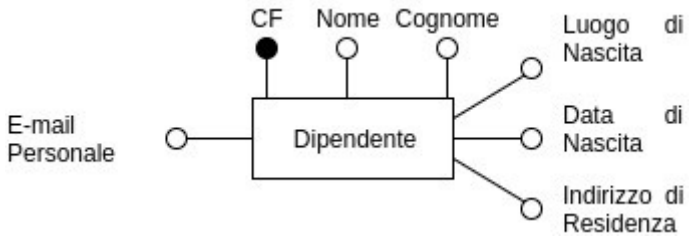
3. Progettazione concettuale

Costruzione dello schema E-R

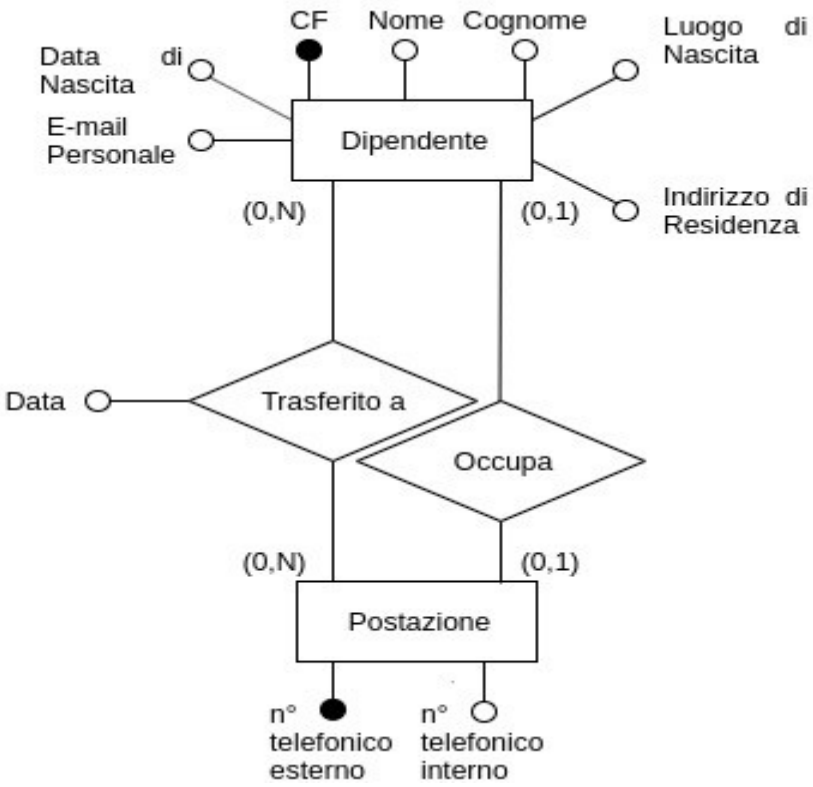
Per la stesura del diagramma ER è stata seguita una tecnica mista, procedendo nel seguente modo:

- Si scelgono una alla volta le entità da riportare nel diagramma ER secondo l'ordine in cui appaiono nella specifica;
- Per ogni entità scelta:
  - Si riporta nel diagramma l'entità dettagliata secondo uno schema bottom-up;
  - Si aggiungono le relazioni e le entità relative navigando a macchia d'olio tra i concetti secondo uno schema inside-out.

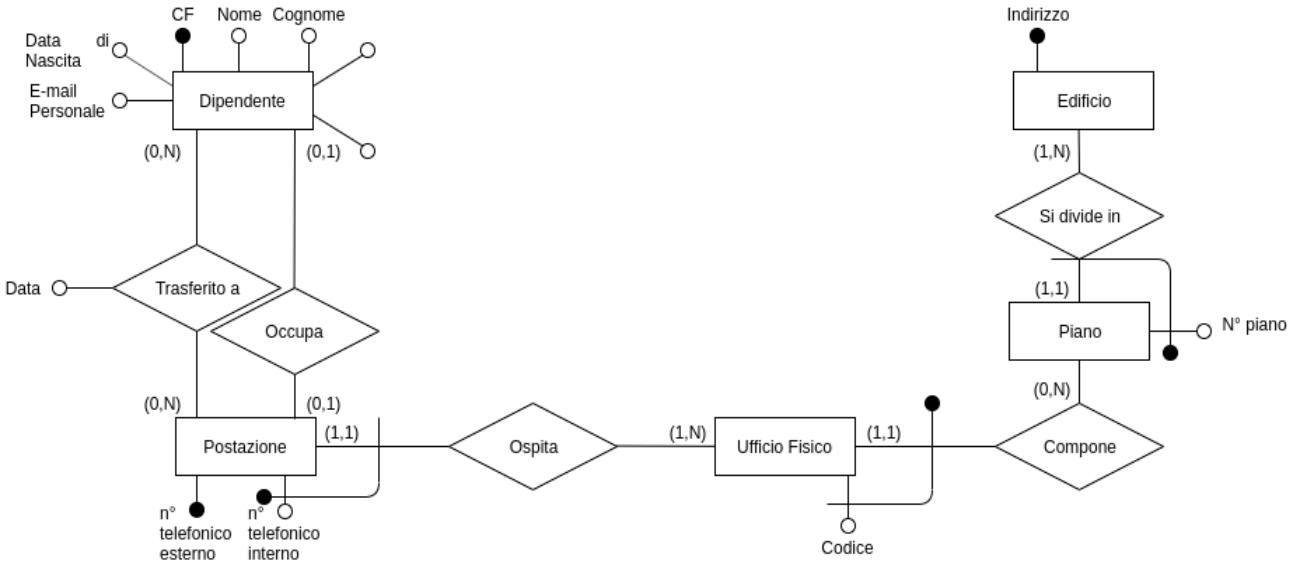
Di seguito sono illustrati i vari passaggi con relativa descrizione.



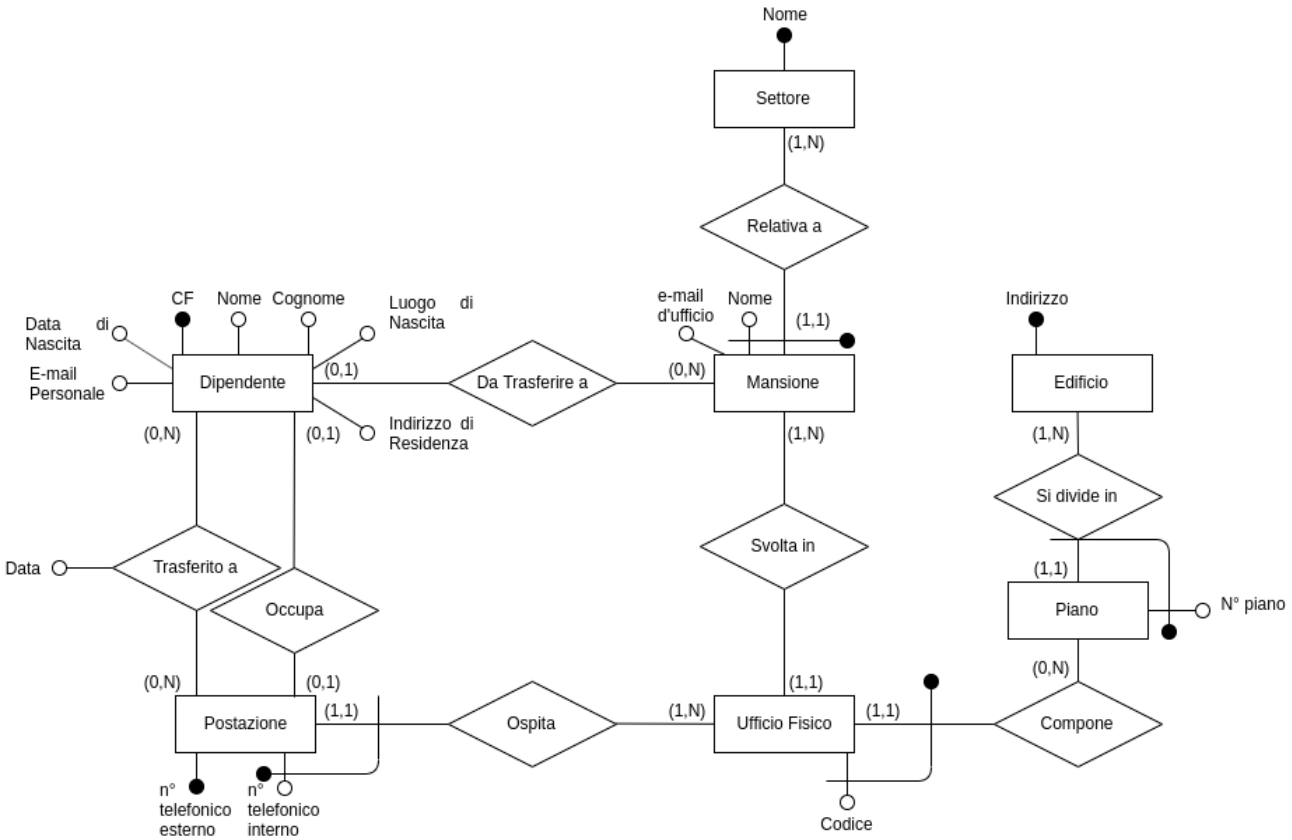
Step 1: La prima entità ad essere raffigurata è Dipendente, con i suoi vari attributi. Risulta facile scegliere l'attributo CF come identificatore primario.



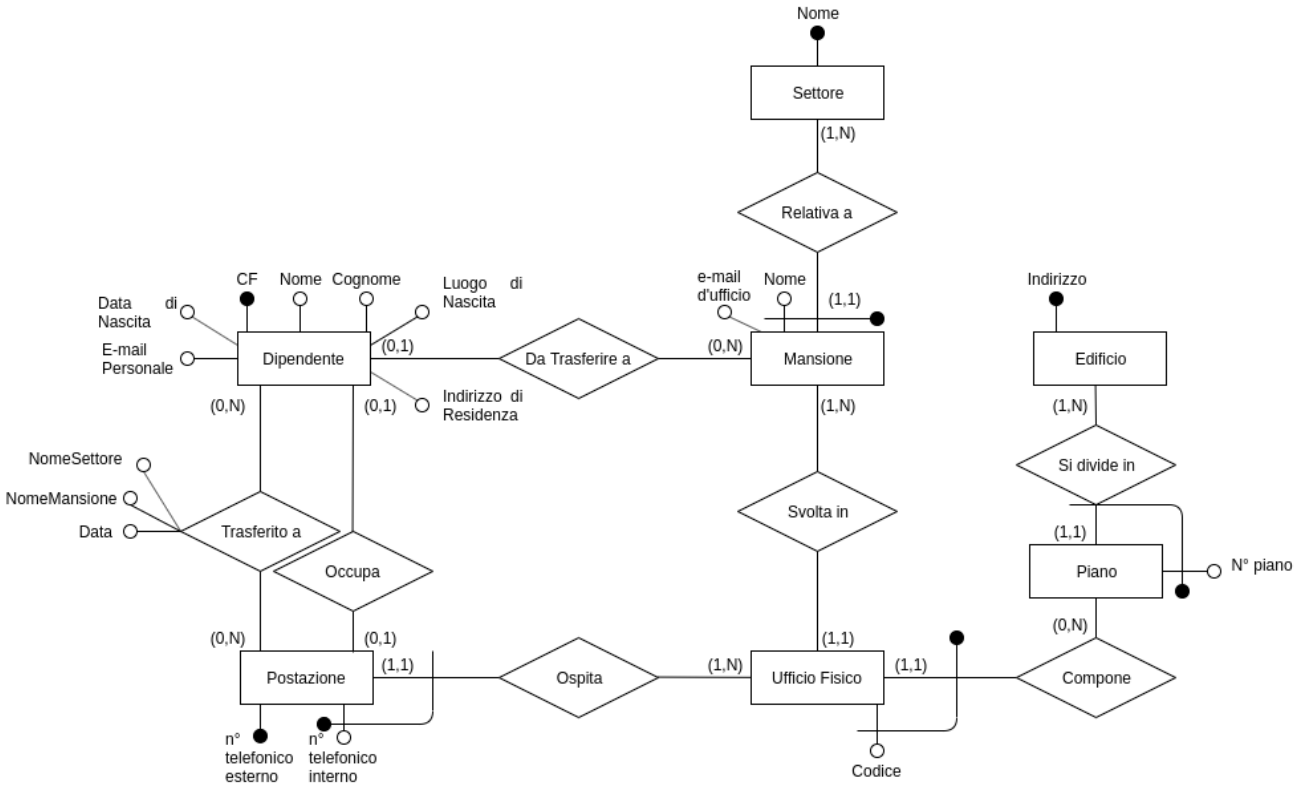
Step 2: Una postazione è identificata dal n° telefonico esterno ad essa associata. Un Dipendente può legarsi ad una Postazione mediante l'associazione Occupa o l'associazione Trasferito A. Occupa tiene traccia del Dipendente che attualmente è assegnato a una certa postazione. Può succedere che a un Dipendente non sia assegnata nessuna postazione (ad esempio perchè è stato appena assunto e deve essere ancora trasferito a una postazione), oppure che a una postazione non sia ancora stato assegnato nessun Dipendente. Trasferito A tiene traccia della storia dei trasferimenti di un Dipedente tra le varie Postazioni presenti nell'azienda, distinguendo le varie coppie (Dipendente, Postazione) mediante l'attributo di relazione Data.



Step 3: Le Postazioni sono tutte raccolte all'interno dei diversi Uffici Fisici, i quali sono identificati da un codice in funzione del piano che lo ospitano. Da notare che l'associazione Compone prevede che in un Edificio possano esistere Piani che non contengono Uffici (senza escludere che in futuro lo possano fare, ad esempio a seguito di lavori di ristrutturazione o di contratti d'affitto).



Step 4: L'associazione Svolta In lega ogni Ufficio Fisico dell'azienda a una Mansione, identificata da un Nome relativo al Settore con cui è legata attraverso l'associazione Relativa A. Ogni Mansione presenta una e-mail d'ufficio, ovvero un recapito a cui è possibile inviare le comunicazioni destinate a tutti gli Uffici Fisici che svolgono una particolare Mansione. L'associazione Da Trasferire A tiene traccia dei Dipendenti che sono stati indicati come da trasferire in futuro a una specifica Mansione.



Step 5: Per permettere la storicizzazione delle mansioni assegnate a un Dipendente, sono stati aggiunti degli attributi alla relazione Trasferito A. Registrare unicamente la Postazione non sarebbe bastato in quanto una postazione è associata univocamente a un Ufficio Fisico, il quale però nel tempo potrebbe aver cambiato Mansione.

Integrazione finale

Non sono stati riscontrati conflitti sui nomi o conflitti strutturali durante la stesura del diagramma ER.

Regole aziendali

- Un Dipendente che occupa una Postazione deve figurare come Trasferito A tale Postazione;
- Un dipendente non può essere riassegnato ad una postazione in cui era già stato assegnato nei passati tre anni

Dizionario dei dati

Entità	Descrizione	Attributi	Identificatori

Dipendente	Un membro lavoratore dell'azienda, per cui svolge una particolare Mansione	CF, Nome, Cognome, Data di Nascita, E-Mail, Luogo di nascita, Indirizzo di Residenza.	CF
Mansione	Attività svolta dai Dipendenti di un Ufficio	Nome, E-mail d'Ufficio	Coppia Nome, Nome del Settore (id. esterno)
Settore	Aggregazione di diversi uffici associati a Mansioni simili, per esempio finalizzate a uno scopo comune.	Nome	Nome
Ufficio	Partizione di Postazioni occupate da Dipendenti dedicati a una stessa Mansione, localizzati in un particolare Piano di un Edificio	Codice	Codice, Numero Piano, Indirizzo Edificio (id esterno)
Postazione	Assegnazione di un Dipendente a un Ufficio	N° telefonico interno, N° telefonico esterno	N° telefonico esterno OPPURE coppia N° telefonico interno, E-Mail Ufficio (id esterno)
Piano	Un qualsiasi piano di un Edificio dell'Azienda, che può contenere Uffici.	N° piano	Coppia N° piano, Indirizzo edificio (id esterno)
Edificio	Un edificio appartenente all'Azienda.	Indirizzo	Indirizzo

4. Progettazione logica

Volume dei dati

Alcune note precedenti la Tavola dei Volumi dei dati:

- Il n° dei Dipendenti è una stima frutto di una media aritmetica del n° dei dipendenti delle seguenti aziende (Fonte: Wikipedia):
  - Facebook: 23 165;
  - Google: 114 096;
  - Microsoft: 125 000;
  - Sony: 128 400;
  - Nintendo: 5 944;
  - PayPal: 18 100.
- Si assume che il 5% dei Dipendenti sia in media segnalato come da trasferire, e che il 5% di tali dipendenti sia neo-assunto;
- Si assume che in media un Dipendente sia stato soggetto a Trasferimento per 3 volte;
- Si assume che in media ogni Ufficio ospiti 50 postazioni, di cui 40 sono occupate;
- Il n° di Uffici è calcolato come $\frac{\text{n° Dipendenti}}{(n° \, Postazioni\, occupate \,\forall\, Ufficio)}$;
- Si assume che ogni Settore comprenda 50 Mansioni;
- Si assume che in media ogni Edificio ospiti 5 piani, e che ogni piano ospiti 10 uffici.

Concetto nello schema	Tipo[1]	Volume atteso
Dipendente	E	70 000
Mansione	E	360
Ufficio	E	1 750
Settore	E	36
Postazione	E	90 000
Edificio	E	36
Piano	E	180
Da Trasferire A	R	3 500
Svolta In	R	1 750
Occupa	R	69 825
Trasferito A	R	209 475
Relativa a	R	360
Ospita	R	90 000
Compone	R	1 750
Si Divide In	R	180

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

Tavola delle operazioni

Alcune note:

- Per poter gestire il caso in cui un nuovo dipendente venga assunto, è stata aggiunta l'operazione 9, sebbena essa non sia espicitamente citata dalla specifica. In questo modo, è possibile registrare il neo-dipendente con postazione e data ultimo trasferimento pari a NULL, e come da trasferire alla mansione per cui è stato assunto.

Cod.	Descrizione	Frequenza attesa
1	Genera report indicante tutti i dipendenti, raggruppati per mansione, che devono essere trasferiti. (sola lettura)	Una volta alla settimana
2-1	Dato un dipendente, elenca tutti gli altri dipendenti con cui è possibile scambiare la postazione (sola lettura)	937 volte alla settimana
2-2	Scambia due dipendenti assegnati a una Postazione contenuta nello stesso o diversi Uffici che svolgono la stessa Mansione.	Guardando la Tavola dei Volumi, notiamo che il num dipendenti da trasferire ogni settimana = $70000 \text{ dipendenti} \cdot \left(\frac{5}{100}\right)$, dunque sono 1875 coppie. Assumendo che la metà delle coppie sia composta da Dipendenti che svolgono la stessa Mansione, otteniamo un tasso di utilizzo di tale operazione pari a 937 volte alla settimana.
3-1	Trova tutti gli uffici assegnati a una Mansione che presentano almeno una postazione vuota (sola lettura)	1875 volte alla settimana
3-2	Assegna un Dipendente a una Postazione libera in base alla Mansione a cui è correntemente assegnato. La postazione libera può essere scelta tra quelle presenti nel result set fornito dalla 3-1.	1875 volte alla settimana
4	Modifica la Mansione di un Dipendente	1875 volte a settimana
5	Genera report indicante a quali uffici un Dipendente è stato assegnato nel tempo (sola lettura)	3500 volte a settimana
6	Ricerca parametrizzata di un Dipendente per conoscerne l'ubicazione e i recapiti. (sola lettura)	70000 volte al giorno
7	Ricerca informazioni tramite numero di telefono (sola lettura)	70000 volte al giorno
8	Controlla per ogni Dipendente se tra le tuple in cui appare nella relazione Trasferito A la data della tupla più recente indica che è decorso il periodo di turnazione. Se questo è il caso, aggiunge un'associazione tra il Dipendente e la Mansione che sta attualmente svolgendo alla relazione Da Trasferire A. (tale operazione può essere invocata	1 volta al giorno

	periodicamente dal sistema senza necessitare un intervento umano)	
9	Registra un nuovo dipendente all'interno del DB, indicandolo come da trasferire in base alla Mansione per cui è stato assunto.	200 volte al mese

Costo delle operazioni

Codice	Costo
2-2	- Scelta una coppia di Dipendenti assegnati alla stessa Mansione, bisogna aggiornare le tuple di entrambi i dipendenti nella relazione Occupa scambiandone i valori delle Postazioni. 2 letture e 2 scritture = costo 6; - Per ogni membro della coppia, occorre aggiungere una tupla nella relazione Trasferito A. 2 scritture = costo 4; Il costo totale a invocazione è di 10 a invocazione, dunque 9370 a settimana.
3-2	- Lettura della tupla del dipendente da trasferire nella relazione Da Trasferire A per conoscere la Mansione. 1 lettura = costo 1; - Invocazione della 3-1 per trovare una postazione libera. $\frac{1750 \text{ uffici}}{360 \text{ mansioni}} \cdot 50 \, postazioni = 250 \, letture$ = costo 250; - Aggiornare la tupla relativa al Dipendente trasferito nella relazione Occupa. 1 scrittura = costo 2; - Aggiugere un'associazione tra il Dipendente e la Postazione nella relazione Trasferito A. 1 scrittura = costo 2;

	- Rimuovere il Dipendente nella relazione Da Trasferire A. 1 scrittura = costo 2 Costo totale a invocazione 257, a settimana 481‘875
4	- Aggiungere una nuova tupla con un Dipendente e la sua Mansione futura nella relazione Da Trasferire A. 1 scrittura = Costo 2 Costo totale a invocazione 2, a settimana 3750
8	- Leggere tutte le tuple all’interno della relazione Trasferito A per trovare quelle con la data più recente per ogni dipendente. 209 475 letture = costo 209 475; - Per i Dipendenti che posseggono una tupla con la data più recente indicante che è decorso il periodo di turnazione, aggiungere un’associazione tra il Dipendente e la sua attuale Mansione nella relazione Da trasferire A. 1875 scritture = costo 3750; Costo totale a invocazione 213 225, a settimana 1’492’575
9	- Aggiunta di una nuova entità Dipendente. 1 scrittura = costo 2; - Aggiunta di un’associazione tra il Dipendente e la Mansione per cui è stato assunto nella relazione Da Trasferire A. 1 scrittura = Costo 2; Costo totale a invocazione 4, a settimana 200

Ristrutturazione dello schema E-R

- Analisi delle rindondanze:
  - Si può dedurre l’occupazione di un Dipendente cercando la tupla con quel Dipendente
5 come partecipante nella relazione Trasferito A con la data più recente, rendendo quindi rindondante la presenza della relazione Occupa. La rimozione della relazione Occupa risparmierebbe alcuni aggiornamenti alle operazioni 2 e 3-2, riducendo il loro rispettivo

costo a 3748/settimana e 478’125/settimana. Tuttavia, la deduzione dell’occupazione di un Dipendente dalla relazione Trasferito A comporta una ricerca di tutte le tuple di tale relazione in cui il Dipendente partecipa, estrarre quella con la data più recente e leggere la postazione partecipante, ovvero un numero di operazioni più elevato rispetto alla singola lettura della relazione Occupa. Da ciò si deduce che tutte le operazioni che prevedono una lettura dell’occupazione di un Dipendente vedrebbero il loro costo aumentato a seguito della rimozione della ridondanza. Ad esempio, la 3-1, in presenza della ridondanza, dovrebbe leggere per ogni Ufficio le Postazioni associate nella relazione Ospita, e per ogni Postazione leggere se presenta una tupla nella relazione Ospita, con un costo totale di $1 \cdot 50 \cdot 1750 = 87500$. In assenza della ridondanza, la 3-1 dovrebbe comunque leggere per ogni Ufficio le Postazioni associate nella relazione Ospita, e per ogni Postazione trovare tutte le tuple in cui la Postazione partecipa nella relazione Trasferito A, estrarre quella con la data più recente e leggerne il Dipendente associato. Assumendo che ogni Postazione compaia in 2 tuple della relazione Trasferito A, la 3-1 presenta un costo di $(2+1+1) \cdot 50 \cdot 1750 = 350000$. Si noti come la rimozione della ridondanza comporterebbe una riduzione dei costi da parte delle operazioni con scritture nell’ordine delle migliaia, e un aumento dei costi da parte di una singola operazione di sola lettura nell’ordine delle centinaia di migliaia. La preservazione della ridondanza comporterebbe la presenza di 69 825 tuple superflue. Assumendo una dimensione di 2 B per tupla, la memoria sprecata ammonterebbe a circa 140 KB, una cifra considerata ragionevole rispetto al guadagno di prestazioni. Pertanto, è stato deciso di preservare la ridondanza.

- Il peso maggiore che influenza il costo della 8 è dato dall’estrazione della tupla più recente nella relazione Trasferito A. Aggiungere un attributo ridondante “Data ultimo trasferimento” con cardinalità (0,1) all’entità Dipendente ridurebbe il numero di letture da 209 475 a 70 000, portando il costo totale da 1’492’575/settimana a 516 250/settimana. Per mantenere tale ridondanza coerente con le altre informazioni presenti nella BD, la 2 dovrebbe eseguire 2 scritture supplementari (costo 4, dunque 13 118/settimana) e la 3-2 dovrebbe eseguire 1 scrittura supplementare (costo 2. dunque 485’625/settimana). Il vantaggio dato dalle letture risparmiate supera grandemente il numero di modifiche aggiuntive dato dagli aggiornamenti supplementari. Assumendo che l’attributo occupi uno spazio di 8 B, la memoria sprecata dalla ridondanza ammonterebbe a 560 KB. Con questi dati alla mano, è stato deciso di introdurre una ridondanza a favore delle prestazioni della BD.

- Eliminazione delle generalizzazioni:
  - Non sono presenti generalizzazioni da eliminare;
- Scelta identificatori primari:
  - Dipendente: CF;
  - Postazione: n° telefonico esterno;
  - Ufficio Fisico: Codice, Numero Piano, Indirizzo Edificio;
  - Mansione: Nome, NomeSettore. Usando tale coppia si permette ai Settori di nominare le Mansioni indipendentemente dagli altri settori, al costo di una leggera complicazione della traduzione nello schema relazionale;
  - Settore: Nome;
  - Piano: n° piano, IndirizzoEdificio;
  - Edificio: Indirizzo

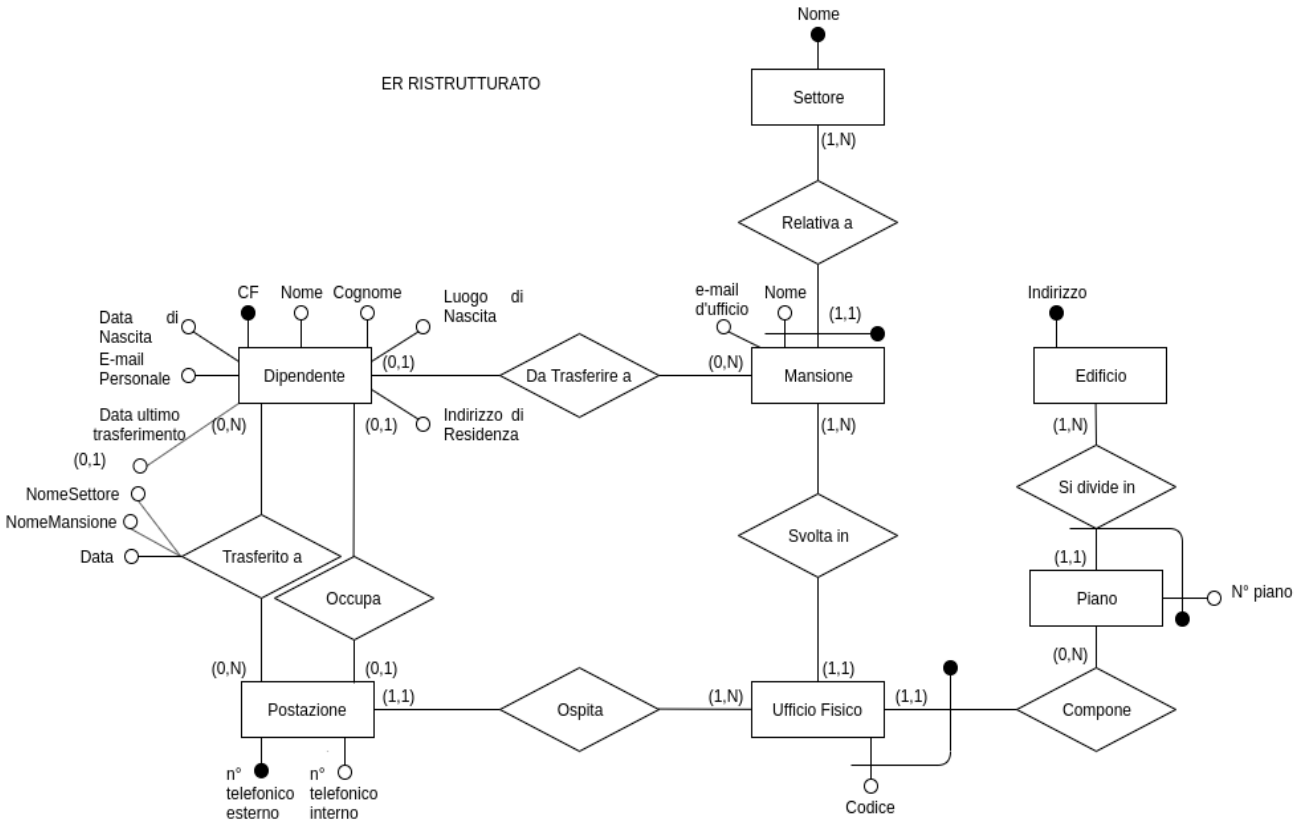


Figure 1: Diagramma ER dopo il processo di ristrutturazione.

Trasformazione di attributi e identificatori

Per identificare le Postazioni era possibile anche utilizzare la coppia (n° telefonico interno, e-mail d’ufficio), tuttavia la scelta di un identificatore esterno come chiave primaria avrebbe complicato la

traduzione nello schema relazionale.

La scelta dell'identificatore esterno come chiave primaria è stata già trattata nella sezione Ristrutturazione dello Schema ER (scelta degli identificatori primari);

È ragionevole pensare che un Piano debba essere identificato, oltre che dal suo numero, anche dall'Edificio ove è sito, pertanto è stato scelto di mantenere l'identificatore esterno.

Traduzione di entità e associazioni

Alcune note:

- Sono state applicate traduzioni ottime laddove tale processo non avrebbe implicato la presenza di molte tuple con valori nulli, per ridurre il n° di tabelle necessarie;
- Dato che è altamente improbabile che un piano possa cambiare Edificio, e che non esistano Edifici senza Piani, è stato deciso che la sola presenza della relazione Piano basti a includere anche le informazioni relative agli Edifici. Lo stesso ragionamento è stato applicato per usare la relazione Mansione per mantenere anche le informazioni relative ai Settori.
- È ragionevole pensare che in media il numero di dipendenti neo-assunti (e dunque ancora non assegnati a una postazione) sia di molto inferiore rispetto a quello dei dipendenti veterani. Pertanto è stato deciso di non creare una relazione OCCUPA e di includere la Postazione nella relazione DIPENDENTE. Ciò comporterà la presenza di qualche valore nullo, un costo da pagare ragionevole in confronto alle numerose tuple ridondanti che sarebbero state presenti nella relazione OCCUPA.

DIPENDENTE(<u>CF</u>, Nome, Cognome, LuogoNascita, DataNascita, EmailPersonale, IndirizzoResidenza, NumTelefonicoEsternoPostazione, DataUltimoTrasferimento);

POSTAZIONE(<u>NumTelefonicoEsterno</u>, NumTelefonicoInterno, CodiceUfficio, NumeroPiano, IndirizzoEdificio);

UFFICIO_FISICO(<u>Codice, NumPiano, IndirizzoEdificio</u>, NomeMansione, NomeSettore);

MANSIONE(<u>Nome, NomeSettore</u>, EmailUfficio);

PIANO(<u>Numero, IndirizzoEdificio</u>);

TRASFERITO_A(CFDipendente, <u>NumTelefonicoEsternoPostazione, Data,</u> NomeMansione, NomeSettore);

DA_TRASFERIRE_A(<u>CFDipendente</u>, NomeMansione, NomeSettore);

POSTAZIONE(CodiceUfficio, NumeroPIano, IndirizzoEdificio) $\subseteq$ UFFICIO(Codice, NumeroPiano, IndirizzoEdificio);

UFFICIO(NomeMansione, NomeSettore) ⊆ MANSIONE(Nome, NomeSettore);
UFFICIO(NumPiano, IndirizzoEdificio) ⊆ PIANO(Numero, IndirizzoEdificio);
DA_TRASFERIRE_A(CFDipendente) ⊆ DIPENDENTE(CF);
DA_TRASFERIRE_A(NomeMansione, NomeSettore) ⊆ MANSIONE(Nome, NomeSettore);
DIPENDENTE(NumTelefonicoEsternoPostazione) ⊆ POSTAZIONE(NumTelefonicoEsterno)
TRASFERITO_A(CFDipendente) ⊆ DIPENDENTE;
TRASFERITO_A(NomeMansione, NomeSettore) ⊆ MANSIONE (Nome, NomeSettore);

Normalizzazione del modello relazionale

Le dipendenze funzionali sono:

1. CF → Nome, Cognome, LuogoNascita, DataNascita, EmailPersonale, IndirizzoResidenza, NumTelefonicoEsternoPostazione, DataUltimoTrasferimento;
2. NumTelefonicoEsterno → EmailUfficio, NumTelefonicoInterno,
3. NumeroUfficio, NumeroPiano, IndirizzoEdificio → NomeMansione, NomeSettore;
4. CFDipendente, Data → NumTelefonicoEsternoPostazione;
5. NumTelefonicoEsternoPostazione, Data → CFDipendente;
6. CFDipendente → NomeMansione, NomeSettore;
7. NomeMansione, NomeSettore → EmailUfficio.

Tutti i membri a sinistra sono chiavi delle loro rispettive relazioni, dunque tutte le relazioni sono in BCNF.

Non sono presenti dipendenze parziali né transitive, dunque lo schema è in 3NF.

5. Progettazione fisica

Utenti e privilegi

- "**dipendente**" (ruoli: Dipendente), progettato per l'uso da parte del dipendente comune per semplici operazioni di lettura ristrette a determinate tuple:
  - ricercaDipendente: EXECUTE;
  - ricercaPerNumeroTelefono: EXECUTE;
- "**dipendenteSettoreSpazi**" (ruoli: Dipendente, DipendenteSettoreSpazi), progettato per l'utilizzo da parte dei dipendenti del settore spazi, mediante il quale possono manipolare le informazioni relative a piani, edifici, uffici e postazioni, e gestire i trasferimenti dei dipendenti:
  - PIANO: DELETE, INSERT, SELECT, UPDATE;
  - UFFICIO_FISICO: DELETE, INSERT, SELECT, UPDATE;
  - POSTAZIONE: DELETE, INSERT, SELECT, UPDATE;
  - TRASFERITO_A: DELETE, INSERT, SELECT, UPDATE;
  - DA_TRASFERIRE_A: SELECT
  - generaReportDaTrasferire: EXECUTE;
  - trovaDipendentiScambiabili: EXECUTE;
  - scambiaDipendenti: EXECUTE;
  - trovaUfficioConPostazioneVuota: EXECUTE;
  - assegnaDipendenteAPostazioneVuota: EXECUTE.
- "**dipendenteSettoreAmministrativo**" (ruoli: Dipendente, DipendenteSettoreAmministrativo), progettato per l'utilizzo da parte dei dipendenti del settore amministrativo, mediante il quale possono manipolare informazioni relative a dipendenti, mansioni e settori:
  - DIPENDENTE: DELETE, INSERT, SELECT, UPDATE;
  - MANSIONE: DELETE, INSERT, SELECT, UPDATE;
  - DA_TRASFERIRE_A: DELETE, INSERT, SELECT, UPDATE;
  - cambiaMansioneDipendente: EXECUTE;
  - assumiDipendente: EXECUTE;
  - elencaTrasferimentiDipendente: EXECUTE;
- "**maintainer**"(ruoli: Maintainer), utente destinato all'utilizzo da parte dello sviluppatore che si occuperà di mantenere lo schema, apportando modifiche ove necessario:
  - directory_aziendale.*: CREATE, DROP, GRANT OPTION, REFERENCES, EVENT, LOCK TABLES.

La divisione dei privilegi tra i vari utenti è stata progettata in modo che nessun utente possa avere singolarmente il totale controllo dello schema, il che può rivelarsi vantaggioso in caso di compromissione di un utente.
Da notare che l'utente "maintaner", sebbene presenti tutti i privilegi di manipolazione dello schema, non ha accesso ai valori delle tuple contenute nelle tabelle nè può eseguire stored procedures.

Strutture di memorizzazione

Tabella <DIPENDENTE>		
Attributo	**Tipo di dato**	**Attributi**[2]
CF	CHAR(16)	PK

Nome	VARCHAR(45)	NN
Cognome	VARCHAR(45)	NN
LuogoNascita	VARCHAR(45)	
DataNascita	DATE	
EmailPersonale	VARCHAR(45)	UQ
IndirizzoResidenza	VARCHAR(45)	
NumTelefonicoEsternoPostazione	VARCHAR(45)	UQ
DataUltimoTrasferimento	DATE	

Tabella <MANSIONE>		
Attributo	**Tipo di dato**	**Attributi**
Nome	VARCHAR(45)	PK
NomeSettore	VARCHAR(45)	PK
EmailUfficio	VARCHAR(45)	

Tabella <PIANO>		
Attributo	**Tipo di dato**	**Attributi**
Numero	INT	PK, UN
IndirizzoEdificio	VARCHAR(45)	PK

Tabella <POSTAZIONE>		
Attributo	**Tipo di dato**	**Attributi**
NumTelefonicoEsterno	VARCHAR(45)	PK
NumTelefonicoInterno	VARCHAR(45)	NN
CodiceUfficio	VARCHAR(45)	NN
NumPiano	INT	NN, UN
IndirizzoEdificio	VARCHAR(45)	NN

5

Tabella <UFFICIO_FISICO>		
Attributo	**Tipo di dato**	**Attributi**
Codice	VARCHAR(45)	PK
NomeMansione	VARCHAR(45)	NN
NomeSettore	VARCHAR(45)	NN
NumPiano	INT	PK
IndirizzoEdificio	VARCHAR(45)	PK

[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.

Tabella <TRASFERITO_A>		
Attributo	**Tipo di dato**	**Attributi**
CFDipendente	CHAR(16)	NN
NumTelefonicoEsternoPostazione	VARCHAR(45)	PK
Data	DATE	PK
NomeMansione	VARCHAR(45)	NN
NomeSettore	VARCHAR(45)	NN

Tabella <DA_TRASFERIRE_A>		
Attributo	**Tipo di dato**	**Attributi**
CFDipendente	CHAR(16)	PK
NomeMansione	VARCHAR(45)	NN
NomeSettore	VARCHAR(45)	NN

Indici

Tabella <DA_TRASFERIRE_A>	
Indice <PRIMARY>	**Tipo[3]:**
CFDipendente	PR

5

Tabella <DA_TRASFERIRE_A>	
Indice <MansioneDaTrasferire>	**Tipo:**
NomeMansione	UQ
NomeSettore	UQ

Indice che implementa vincolo di chiave necessario per instanziare una Foreign Key sulla chiave primaria di MANSIONE.

Tabella <DIPENDENTE>	
Indice <PRIMARY>	**Tipo:**
CFDipendente	PR

Tabella <DIPENDENTE>	
Indice <NumTelefonicoEsternoPostazione_UNIQUE>	**Tipo:**
NumTelefonicoEsternoPostazione	UQ

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

Necessario affinchè non risultino due dipendenti assegnati contemporaneamente alla stessa postazione.

Tabella <DIPENDENTE>	
Indice <EmailPersonale_UNIQUE>	**Tipo:**
EmailPersonale	UQ

Necessario affinchè non risultino due dipendenti con la stessa email personale, evitando in questo modo ambiguità.

Tabella <MANSIONE>	
Indice <PRIMARY>	**Tipo:**
Nome	PR
NomeSettore	PR

Tabella <MANSIONE>	
Indice <EmailUfficio>	**Tipo:**
EmailUfficio	UQ

Necessario affinchè non risultino due mansioni con la stessa email d'ufficio, evitando in questo modo ambiguità.

Tabella <PIANO>	
Indice <PRIMARY>	**Tipo:**
Numero	PR
IndirizzoEdificio	PR

Tabella <POSTAZIONE>	
Indice <PRIMARY>	**Tipo:**
NumTelefonicoEsterno	PR

Tabella <POSTAZIONE>	
Indice <NumTelInt_Uff_UNIQUE>	**Tipo:**
NumTelefonicoInterno	UQ
CodiceUfficio	UQ
NumPiano	UQ

IndirizzoEdificio	UQ

Con tale indice è possibile scorrere le postazioni in base alle coordinate relative agli uffici fisici che le contengono all'interno dell'azienda.

Tabella <POSTAZIONE>	
Indice <UFFICIO_FK>	**Tipo:**
CodiceUfficio	UQ
NumPiano	UQ
IndirizzoEdificio	UQ

Indice che implementa vincolo di chiave necessario per instanziare una Foreign Key sulla chiave primaria di UFFICIO_FISICO.

Tabella <TRASFERITO_A>	
Indice <PRIMARY>	**Tipo:**
NumTelefonicoEsternoPostazione	PR
Data	PR

Si assume che a postazione non possano essere trasferiti due dipendenti diversi nella stessa data.

Tabella <UFFICIO_FISICO>	
Indice <PRIMARY>	**Tipo:**
Codice	PR
NumPiano	PR
IndirizzoEdificio	PR

Tabella <UFFICIO_FISICO>	
Indice <MansioneUfficio_FK>	**Tipo:**
NomeMansione	UQ
NomeSettore	UQ

Indice che implementa vincolo di chiave necessario per instanziare una Foreign Key sulla chiave primaria di MANSIONE.

Tabella <UFFICIO_FISICO>	
Indice <Edificio_FK>	**Tipo:**

NumPiano	UQ
IndirizzoEdificio	UQ

Indice che implementa vincolo di chiave necessario per instanziare una Foreign Key sulla chiave primaria di PIANO.

Trigger

```
-- Esegue controlli definiti nella stored procedure
DIPENDENTE_before_insert_update prima di permettere un insert

CREATE
DEFINER=`root`@`localhost`
TRIGGER `directory_aziendale`.`DIPENDENTE_BEFORE_INSERT`
BEFORE INSERT ON `directory_aziendale`.`DIPENDENTE`
FOR EACH ROW
BEGIN
    call checkEmail(new.EmailPersonale);
    call checkTrasferitoA(new.NumTelefonicoEsternoPostazione, new.CF,
new.DataUltimoTrasferimento);
END

-- Esegue controlli definiti nella stored procedure
DIPENDENTE_before_insert_update prima di permettere un update

CREATE
DEFINER=`root`@`localhost`
TRIGGER `directory_aziendale`.`DIPENDENTE_BEFORE_UPDATE`
BEFORE UPDATE ON `directory_aziendale`.`DIPENDENTE`
FOR EACH ROW
BEGIN
    call checkEmail(new.EmailPersonale);
    call checkTrasferitoA(new.NumTelefonicoEsternoPostazione, new.CF,
new.DataUltimoTrasferimento);
END

-- Controlla che esista una tupla in DA_TRASFERIRE_A che corrisponda a quella che
si vuole inserire in TRASFERITO_A. Se c'è, la cancella.
-- Controlla se un dipendente è stato trasferito alla stessa postazione nei
passati 3 anni. Se è questo il caso, segnala un errore
```

```
CREATE
DEFINER=`root`@`localhost`
TRIGGER `directory_aziendale`.`TRASFERITO_A_BEFORE_INSERT`
BEFORE INSERT ON `directory_aziendale`.`TRASFERITO_A`
FOR EACH ROW
BEGIN
    if (select CFDipendente from DA_TRASFERIRE_A where new.CFDipendente =
DA_TRASFERIRE_A.CFDipendente) is not null
        then delete from DA_TRASFERIRE_A where new.CFDipendente =
DA_TRASFERIRE_A.CFDipendente;
         end if;
    if (select NumTelefonicoEsternoPostazione
        from TRASFERITO_A
        where TRASFERITO_A.NumTelefonicoEsternoPostazione =
new.NumTelefonicoEsternoPostazione
            and TRASFERITO_A.CFDipendente = new.CFDipendente
            and timestampdiff(year, TRASFERITO_A.`Data`, curdate()) <= 3
    ) is not null
        then signal sqlstate '45004' set message_text = "ERROR: Un dipendente non
può essere trasferito a una postazione dove è stato già trasferito meno di tre
anni fa";
    end if;
END

-- Mantiene coerente l'attributo ridondante DIPENDENTE.DataUltimoTrasferimento
-- Aggiorna la postazione corrente del DIPENDENTE

CREATE
DEFINER=`root`@`localhost`
TRIGGER `directory_aziendale`.`TRASFERITO_A_AFTER_INSERT`
AFTER INSERT ON `directory_aziendale`.`TRASFERITO_A`
FOR EACH ROW
BEGIN
    update DIPENDENTE
    set
        DataUltimoTrasferimento = new.`data`,
         NumTelefonicoEsternoPostazione = new.NumTelefonicoEsternoPostazione
    where CF = new.CFDipendente;
END
```

```
-- controlla la validità del formato di EmailUfficio prima di un insert

CREATE
DEFINER=`root`@`localhost`
TRIGGER `directory_aziendale`.`MANSIONE_BEFORE_INSERT`
BEFORE INSERT ON `directory_aziendale`.`MANSIONE`
FOR EACH ROW
BEGIN
    call checkEmail(new.EmailUfficio);
END

-- controlla la validità del formato di EmailUfficio prima di un update

CREATE
DEFINER=`root`@`localhost`
TRIGGER `directory_aziendale`.`MANSIONE_BEFORE_UPDATE`
BEFORE UPDATE ON `directory_aziendale`.`MANSIONE`
FOR EACH ROW
BEGIN
    call checkEmail(new.EmailUfficio);
END
```

Eventi

```
-- Implementa l'operazione 8. Instanziato in fase di configurazione del sistema

create if not exists event directory_aziendale.trovaUtentiDaTrasferire on
schedule every 1 day on completion preserve do
    insert ignore into DA_TRASFERIRE_A (CFDipendente, NomeMansione, NomeSettore)
     select CFDipendente, NomeMansione, NomeSettore
     from view_DipendentiDaTrasferirePeriodoScadutoConMansione;
```

Viste

```
-- Elenca tutti i dipendenti di cui il periodo di turnazione è decorso con la
loro attuale mansione.
-- Si assume che il periodo di turnazione dei dipendenti sia di 30 giorni.

CREATE VIEW `view_DipendentiDaTrasferirePeriodoScadutoConMansione` AS
    select DIPENDENTE.CF as CFDipendente, UFFICIO_FISICO.NomeMansione,
```

```
UFFICIO_FISICO.NomeSettore
    from DIPENDENTE join POSTAZIONE join UFFICIO_FISICO
        on DIPENDENTE.NumTelefonicoEsternoPostazione =
POSTAZIONE.NumTelefonicoEsterno
        and POSTAZIONE.CodiceUfficio = UFFICIO_FISICO.Codice
         and POSTAZIONE.NumPiano = UFFICIO_FISICO.NumPiano
         and POSTAZIONE.IndirizzoEdificio = UFFICIO_FISICO.IndirizzoEdificio
    where datediff(DIPENDENTE.DataUltimoTrasferimento, current_date) > 30
```

Stored Procedures e transazioni

```
-- op 1: Genera report indicante tutti i dipendenti, raggruppati per mansione,
che devono essere trasferiti.  (sola lettura)

CREATE PROCEDURE `generaReportDaTrasferire` ()
BEGIN
    select CFDipendente, DA_TRASFERIRE_A.NomeMansione as NewNomeMansione,
DA_TRASFERIRE_A.NomeSettore as NewNomeSettore
    from DA_TRASFERIRE_A
    order by NewNomeMansione, NewNomeSettore;
END

-- op 2-1

CREATE PROCEDURE `trovaDipendentiScambiabili` (in cfDipendente char(16))
BEGIN
    select d1.NumTelefonicoEsternoPostazione as possibilePostazioneDaScambiare,
        d1.CF as cfDipendenteOccupante
    from DIPENDENTE as d1
       join POSTAZIONE as p1 on d1.NumTelefonicoEsternoPostazione =
p1.NumTelefonicoesterno
        join UFFICIO_FISICO as u1 on p1.CodiceUfficio = u1.Codice
           and p1.NumPiano = u1.NumPiano
            and p1.IndirizzoEdificio = u1.IndirizzoEdificio
   where d1.CF != cfDipendente
       and (u1.NomeMansione, u1.NomeSettore) in (
           select NomeMansione, NomeSettore
           from DIPENDENTE as d2 join POSTAZIONE as p2 on
d2.NumTelefonicoEsternoPostazione = p2.NumTelefonicoesterno
               join UFFICIO_FISICO as u2 on p2.CodiceUfficio = u2.Codice
                   and p2.NumPiano = u2.NumPiano
```

```
                and p2.IndirizzoEdificio = u2.IndirizzoEdificio
            where d2.CF = cfDipendente
    );
END

-- op 2-2

CREATE PROCEDURE `scambiaDipendenti` (
    in cfDipendente1 char(16),
    in cfDipendente2 char(16)
)
BEGIN
    declare tempNumeroTelefonico1 varchar(45);
    declare tempNumeroTelefonico2 varchar(45);
    declare tempNomeMansione varchar(45);
    declare tempNomeSettore varchar(45);
    declare exit handler for sqlexception
    begin
       rollback;
        resignal;
    end;
    start transaction;
     -- controlla se le due postazioni appartengono ad uffici fisici assegnati alla stessa mansione e settore, che verranno salvati dentro delle variabili
            call checkDipendentiStessaMansione(cfDipendente1, cfDipendente2, tempNomeMansione, tempNomeSettore);
    if (tempNomeMansione is null and tempNomeSettore is null)
       then signal sqlstate "45005" set message_text = "ERROR: Le postazioni occupate dai dipendenti forniti appartengono ad uffici fisici che sono attualmente assegnati a due mansioni diverse.";
    end if;
    -- scambia i dipendenti e aggiorna la tabella TRASFERITO_A atomicamente
    set tempNumeroTelefonico1 = (select NumTelefonicoEsternoPostazione from DIPENDENTE where CF = cfDipendente1);
    set tempNumeroTelefonico2 = (select NumTelefonicoEsternoPostazione from DIPENDENTE where CF = cfDipendente2);
      update DIPENDENTE set NumTelefonicoEsternoPostazione = null where CF = cfDipendente1;
    update DIPENDENTE set NumTelefonicoEsternoPostazione = null where CF = cfDipendente2;
```

```
        insert into TRASFERITO_A values (cfDipendente1, tempNumeroTelefonico2,
curdate(), tempNomeMansione, tempNomeSettore);
        insert into TRASFERITO_A values (cfDipendente2, tempNumeroTelefonico1,
curdate(), tempNomeMansione, tempNomeSettore);
    commit;
END

-- op 3-1. Passando dei valori null ai parametri è possibile trovare tutti gli
uffici con postazione vuota a prescindere dalla mansione che svolgono

CREATE DEFINER=`root`@`localhost` PROCEDURE `trovaUfficiConPostazioneVuota`(in
inNomeMansione varchar(45), in inNomeSettore varchar(45))
BEGIN
    select POSTAZIONE.NumTelefonicoEsterno, UFFICIO_FISICO.Codice,
UFFICIO_FISICO.NumPiano, UFFICIO_FISICO.IndirizzoEdificio
    from UFFICIO_FISICO
        join POSTAZIONE on UFFICIO_FISICO.Codice = POSTAZIONE.CodiceUfficio
            and UFFICIO_FISICO.NumPiano = POSTAZIONE.NumPiano
              and UFFICIO_FISICO.IndirizzoEdificio = POSTAZIONE.IndirizzoEdificio
        left join DIPENDENTE on POSTAZIONE.NumTelefonicoEsterno =
DIPENDENTE.NumTelefonicoEsternoPostazione
    where DIPENDENTE.CF is null
        and (UFFICIO_FISICO.NomeMansione = inNomeMansione or inNomeMansione is
null)
         and (UFFICIO_FISICO.NomeSettore = inNomeSettore or inNomeSettore is
null);
END

-- op 3-2

CREATE DEFINER=`root`@`localhost` PROCEDURE
`assegnaDipendenteAPostazioneVuota`(in inCFDipendente char(16), in
inNumTelefonicoEsternoPostazione varchar(45))
BEGIN
    declare varNomeMansione varchar(45);
    declare varNomeSettore varchar(45);
    -- if an error occurs a rollback is performed and the error is resignaled to
the caller
    declare exit handler for sqlexception
    begin
```

```
        rollback;
         resignal;
    end;
         start transaction;
    -- checks if the provided seat is empty, raises a signal otherwise
    if (select DIPENDENTE.CF
        from DIPENDENTE
         where DIPENDENTE.NumTelefonicoEsternoPostazione =
inNumTelefonicoEsternoPostazione
    ) is not null
        then signal sqlstate '45002' set message_text = "ERROR: provided seat is
not empty";
    end if;
    -- checks if the job which the provided employer is to be transferred and the
job assigned to the physical office containing the provided seat are the same,
raises an error otherwise
    call getMansioneDaPostazioneSeCorrispondeADaTrasferireA(inCFDipendente,
inNumTelefonicoEsternoPostazione, varNomeMansione, varNomeSettore);
    if varNomeMansione is null and varNomeSettore is null
        then signal sqlstate '45006' set message_text = "ERROR: provided employer
is not registered to be transferred to the job assigned to the physical office
which contains the seat";
    end if;
    -- assigns the provided employer to the provided seat
    insert into TRASFERITO_A values (inCFDipendente,
inNumTelefonicoEsternoPostazione, curdate(),  varNomeMansione, varNomeSettore);
    commit;

END

-- op 4: Modifica la Mansione di un Dipendente.

CREATE PROCEDURE `cambiaMansioneDipendente` (in CFDipendente char(16), in
NomeNuovaMansione varchar(45), in NomeNuovoSettore varchar(45))
BEGIN
    insert into DA_TRASFERIRE_A values (CFDipendente, NomeNuovaMansione,
NomeNuovoSettore);
END

-- op 5
```

```
CREATE  DEFINER=`root`@`localhost`  PROCEDURE  `elencaTrasferimentiDipendente`(in
inCFDipendente char(16))
BEGIN
    select *
    from TRASFERITO_A
    where TRASFERITO_A.CFDipendente = inCFDipendente or inCFDipendente is null
    order by TRASFERITO_A.CFDipendente, TRASFERITO_A.`Data`;
END

-- op 6

CREATE DEFINER=`root`@`localhost` PROCEDURE `ricercaDipendente`(in inNome
varchar(45), in inCognome varchar(45))
BEGIN
    select DIPENDENTE.Nome, DIPENDENTE.Cognome, DIPENDENTE.IndirizzoResidenza,
DIPENDENTE.EmailPersonale, DIPENDENTE.NumTelefonicoEsternoPostazione,
MANSIONE.EmailUfficio
    from DIPENDENTE left join POSTAZIONE on
DIPENDENTE.NumTelefonicoEsternoPostazione = POSTAZIONE.NumTelefonicoEsterno
       left join UFFICIO_FISICO on POSTAZIONE.CodiceUfficio =
UFFICIO_FISICO.Codice
          and POSTAZIONE.NumPiano = UFFICIO_FISICO.NumPiano
            and POSTAZIONE.IndirizzoEdificio = UFFICIO_FISICO.IndirizzoEdificio
       left join MANSIONE on UFFICIO_FISICO.NomeMansione = MANSIONE.Nome
          and UFFICIO_FISICO.NomeSettore = MANSIONE.NomeSettore
    where (DIPENDENTE.Nome = inNome or inNome is null)
       and (DIPENDENTE.Cognome = inCognome or inCognome is null);
END

-- op 7

CREATE PROCEDURE `ricercaPerNumeroTelefono` (in numTelefono varchar(45))
BEGIN
    select UFFICIO_FISICO.Codice as CodiceUfficio, UFFICIO_FISICO.NumPiano,
UFFICIO_FISICO.IndirizzoEdificio, DIPENDENTE.CF as CFDipendente, DIPENDENTE.Nome
as NomeDipendente, DIPENDENTE.Cognome as CognomeDipendente,
DA_TRASFERIRE_A.NomeMansione as NomeMansioneInTrasferimentoA,
DA_TRASFERIRE_A.NomeSettore as NomeSettoreInTrasferimentoA
    from POSTAZIONE join UFFICIO_FISICO on POSTAZIONE.CodiceUfficio =
```

```
UFFICIO_FISICO.Codice
          and POSTAZIONE.NumPiano = UFFICIO_FISICO.NumPiano
          and POSTAZIONE.IndirizzoEdificio = UFFICIO_FISICO.IndirizzoEdificio
        left join DIPENDENTE on DIPENDENTE.NumTelefonicoEsternoPostazione =
numTelefono
        left join DA_TRASFERIRE_A on DIPENDENTE.CF = DA_TRASFERIRE_A.CFDipendente
    where POSTAZIONE.NumTelefonicoEsterno = numTelefono;
END

-- op 9

CREATE PROCEDURE `assumiDipendente` (
    in cf char(16),
    in nome varchar(45),
    in cognome varchar(45),
    in luogoNascita varchar(45),
    in dataNascita date,
    in emailPersonale varchar(45),
    in indirizzoResidenza varchar(45),
    in nomeMansione varchar(45),
    in nomeSettore varchar(45)
)
BEGIN
    -- on any error performs a rollback, then resignals to caller
    declare exit handler for sqlexception
    begin
       rollback;
        resignal;
    end;
    -- performs op 9 atomically
    start transaction;
    insert into DIPENDENTE
       values (
           cf,
             nome,
             cognome,
             luogoNascita,
             dataNascita,
             emailPersonale,
             indirizzoResidenza,
```

```
            null,
            null
        );
    insert into DA_TRASFERIRE_A
        values (
            cf,
            nomeMansione,
            nomeSettore
        );
    commit;
END

-- controlla che il formato e-mail sia valido, altrimenti invia un segnale di errore

CREATE DEFINER=`root`@`localhost` PROCEDURE `checkEmail`(in email varchar(45))
BEGIN
    if email not like '_%@_%.__%'
        then signal sqlstate '45001'
            set message_text = "ERROR: invalid Email format, a compliant one is username@hostname.domain";
    end if;
END

-- controlla se un dipendente registrato come assegnato a una postazione risulti prima come trasferito a tale postazione. Nel caso non lo sia invia un segnale di errore.

CREATE DEFINER=`root`@`localhost` PROCEDURE `checkTrasferitoA`(in numTelefonicoEsternoPostazione varchar(45), in cfDipendente char(16), in dataUltimoTrasferimento date)
BEGIN
    if numTelefonicoEsternoPostazione is not null
        and dataUltimoTrasferimento is not null
        and (select NumTelefonicoEsternoPostazione from TRASFERITO_A where TRASFERITO_A.NumTelefonicoEsternoPostazione = numTelefonicoEsternoPostazione and TRASFERITO_A.`Data` = dataUltimoTrasferimento) is null
        then signal sqlstate '45003'
        set message_text =  "ERROR: Can't find a transfer of this employer in TRASFERITO_A to designed postation with provided date";
```

```
    end if;
END

CREATE DEFINER=`root`@`localhost` PROCEDURE
`getMansioneDaPostazioneSeCorrispondeADaTrasferireA`(
    in inCFDipendente char(16),
     in inNumTelefonicoEsternoPostazione varchar(45),
    out outNomeMansione varchar(45),
    out outNomeSettore varchar(45)
)
BEGIN
    select DA_TRASFERIRE_A.NomeMansione, DA_TRASFERIRE_A.NomeSettore
    from DIPENDENTE
        left join DA_TRASFERIRE_A on DIPENDENTE.CF = DA_TRASFERIRE_A.CFDipendente
        left join UFFICIO_FISICO on DA_TRASFERIRE_A.NomeMansione =
UFFICIO_FISICO.NomeMansione
            and DA_TRASFERIRE_A.NomeSettore = UFFICIO_FISICO.NomeSettore
        left join POSTAZIONE on UFFICIO_FISICO.Codice = POSTAZIONE.CodiceUfficio
            and UFFICIO_FISICO.NumPiano = POSTAZIONE.NumPiano
            and UFFICIO_FISICO.IndirizzoEdificio = POSTAZIONE.IndirizzoEdificio
    where POSTAZIONE.NumTelefonicoEsterno = inNumTelefonicoEsternoPostazione
        and DIPENDENTE.CF = inCFDipendente
        and DA_TRASFERIRE_A.NomeMansione = UFFICIO_FISICO.NomeMansione
        and DA_TRASFERIRE_A.NomeSettore = UFFICIO_FISICO.NomeSettore
     into outNomeMansione, outNomeSettore
    ;
END

-- controlla se le due postazioni appartengono ad uffici fisici assegnati alla
stessa mansione e settore, che verranno salvati dentro delle variabili

CREATE PROCEDURE `checkDipendentiStessaMansione` (
    in cfDipendente1 char (16),
    in cfDipendente2 char(16),
     out nomeMansione varchar(45),
     out nomeSettore varchar(45)
)
BEGIN
    select u1.NomeMansione, u1.NomeSettore
        from DIPENDENTE as d1 join POSTAZIONE as p1 on
```

```
d1.NumTelefonicoEsternoPostazione = p1.NumTelefonicoEsterno
            join UFFICIO_FISICO as u1 on p1.CodiceUfficio = u1.Codice
              and p1.NumPiano = u1.NumPiano
              and p1.IndirizzoEdificio = u1.IndirizzoEdificio
        where d1.CF = cfDipendente1
         and (u1.NomeMansione, u1.NomeSettore) = (
           select u2.NomeMansione, u2.NomeSettore
            from DIPENDENTE as d2 join POSTAZIONE as p2 on
d2.NumTelefonicoEsternoPostazione = p2.NumTelefonicoEsterno
              join UFFICIO_FISICO as u2 on p2.CodiceUfficio = u2.Codice
                  and p2.NumPiano = u2.NumPiano
                  and p2.IndirizzoEdificio = u2.IndirizzoEdificio
           where d2.CF = cfDipendente2
        ) into nomeMansione, nomeSettore;
END
```

Appendice: Implementazione

Codice SQL per instanziare il database

```
-- creating schema

CREATE DATABASE `directory_aziendale`;

-- creating tables

CREATE TABLE IF NOT EXISTS `directory_aziendale`.`DIPENDENTE` (
  `CF` CHAR(16) primary key,
  `Nome` VARCHAR(45) NOT NULL,
  `Cognome` VARCHAR(45) NOT NULL,
  `LuogoNascita` VARCHAR(45) COMMENT 'Un valore NULL indica che il campo è sconosciuto. Qualora lo fosse, andrebbe aggiornato quando possibile.',
  `DataNascita` DATE COMMENT 'Un valore NULL indica che il campo è sconosciuto. Qualora lo fosse, andrebbe aggiornato quando possibile.',
  `EmailPersonale` VARCHAR(45) unique COMMENT 'Le stringhe contenute in questa colonna devono essere nella forma dettata dalla seguente espressione regolare: .*@.*\\..* (ad esempio: username@hostname.domain) ',
  `IndirizzoResidenza` VARCHAR(45) COMMENT 'Un valore NULL indica che il campo è sconosciuto. Qualora lo fosse, andrebbe aggiornato quando possibile.',
  `NumTelefonicoEsternoPostazione` VARCHAR(45) unique COMMENT 'I caratteri presenti devono essere solo cifre numeriche (0-9) ed eventuali simboli (+, -, ...).',
  `DataUltimoTrasferimento` DATE COMMENT 'Un valore NULL indica che il dipendente è stato assunto da poco nell\'azienda e non è stato ancora soggetto a trasferimento dal Settore Spazi.',
)
COMMENT = 'Registra i dipendenti nell\'azienda con i loro recapiti, la loro attuale postazione e la data dell\'ultimo trasferimento, distinti dal CF.'
;

CREATE TABLE IF NOT EXISTS `directory_aziendale`.`UFFICIO_FISICO` (
  `NomeMansione` VARCHAR(45),
  `NomeSettore` VARCHAR(45),
  Codice varchar(45),
  `NumPiano` INT(11) NOT NULL,
```

```
  `IndirizzoEdificio` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`Numero`, NumPiano, IndirizzoEdificio))
COMMENT = 'Registra recapiti, ubicazione all\'interno di un edificio e mansione
svolta degli uffici dell\'azienda.';

CREATE TABLE IF NOT EXISTS `directory_aziendale`.`POSTAZIONE` (
  `NumTelefonicoEsterno` VARCHAR(45) NOT NULL,
  `NumTelefonicoInterno` VARCHAR(45) NOT NULL,
  Codice varchar(45) not null ,
  NumPiano int unsigned not null ,
  IndirizzoEdificio varchar(45),
  PRIMARY KEY (`NumTelefonicoEsterno`),
alter table directory_aziendale.POSTAZIONE
    add constraint NumTelInt_Uff_UNIQUE unique (NumTelefonicoInterno, Codice,
NumPiano, IndirizzoEdificio);

create table if not exists `directory_aziendale`.`MANSIONE`(
    Nome varchar(45),
    NomeSettore varchar(45) unique,
    EmailUfficio varchar(45) unique,
    primary key (Nome, NomeSettore)
)
comment = 'Registra le mansioni svolte all\'interno dell\'azienda. Esse sono
identificate dal loro nome e dal nome del Settore cui fanno parte, in maniera
tale che ogni settore possa nominare le sue mansione l\'uno in maniera
indipendente dall\'altro';

create table if not exists directory_aziendale.PIANO(
    Numero integer unsigned,
    IndirizzoEdificio varchar(45),
    primary key (Numero, IndirizzoEdificio)
)
comment = 'Registra i piani e gli edifici occupati dall\'azienda.';

create table if not exists directory_aziendale.TRASFERITO_A(
    CFDipendente char(16) not null,
    NumTelefonicoEsternoPostazione varchar(45),
    `Data` date,
    NomeMansione varchar(45) not null,
    NomeSettore varchar(45) not null,
```

```
    primary key (NumTelefonicoEsternoPostazione, `Data`)
)
comment = 'Registra tutti i trasferimenti a cui i dipendenti dell\'azienda sono stati sottoposti durante il corso del tempo.';

create table if not exists directory_aziendale.DA_TRASFERIRE_A(
    CFDipendente char(16) primary key,
    NomeMansione varchar(45) not null,
    NomeSettore varchar(45) not null
)
comment = 'Registra tutti i dipendenti che devono essere sottoposti a trasferimento, specificando la mansione che dovranno svolgere a trasferimento compiuto. Se un dipendente è trasferito a una postazione diversa dello stesso ufficio, oppure a un altro ufficio che svolge la stessa mansione, la mansione registrata sarà quella che il dipendente stava svolgendo prima di essere trasferito.'

-- adding foreign keys

alter table directory_aziendale.POSTAZIONE
    add constraint Ufficio_FK
        foreign key (CodiceUfficio, NumPiano, IndirizzoEdificio) references UFFICIO_FISICO (Codice, NumPiano, IndirizzoEdificio)
    on update cascade
    on delete cascade;

alter table directory_aziendale.UFFICIO_FISICO
    add constraint MansioneUfficio_FK
        foreign key (NomeMansione, NomeSettore) references MANSIONE (Nome, NomeSettore)
    on delete cascade,
    add constraint Edificio_FK
        foreign key (NumPiano, IndirizzoEdificio) references PIANO (Numero, IndirizzoEdificio)
    on delete cascade;

alter table directory_aziendale.DA_TRASFERIRE_A
    add constraint CFDipendente_FK
    foreign key (CFDipendente) references DIPENDENTE(CF)
    on update cascade
```

```
    on delete cascade,
    add constraint MansioneDaTrasferireA_FK
        foreign key (NomeMansione, NomeSettore) references MANSIONE (Nome,
NomeSettore)
    on delete cascade;

alter table directory_aziendale.DIPENDENTE
   add constraint NumTelefonicoEsternoPostazioneDipendente_FK
         foreign key (NumTelefonicoEsternoPostazione) references POSTAZIONE
(NumTelefonicoEsterno)
    on update cascade
    on delete restrict
    -- se occupata, per rimuovere la postazione è necessario prima trasferire il
dipendente che la occupa.

-- utenti e privilegi

CREATE USER 'dipendente' IDENTIFIED BY 'Dipendente01!';
GRANT EXECUTE ON procedure `directory_aziendale`.`ricercaDipendente` TO
'dipendente';
GRANT EXECUTE ON procedure `directory_aziendale`.`ricercaPerNumeroTelefono` TO
'dipendente';
CREATE USER 'dipendenteSettoreSpazi' IDENTIFIED BY 'DipendenteSettoreSpazi01!';

GRANT EXECUTE ON procedure `directory_aziendale`.`ricercaDipendente` TO
'dipendenteSettoreSpazi';
GRANT EXECUTE ON procedure `directory_aziendale`.`ricercaPerNumeroTelefono` TO
'dipendenteSettoreSpazi';
GRANT INSERT, SELECT, UPDATE, DELETE ON TABLE `directory_aziendale`.`PIANO` TO
'dipendenteSettoreSpazi';
GRANT INSERT, SELECT, DELETE ON TABLE `directory_aziendale`.`UFFICIO_FISICO` TO
'dipendenteSettoreSpazi';
GRANT EXECUTE ON procedure `directory_aziendale`.`generaReportDaTrasferire` TO
'dipendenteSettoreSpazi';
GRANT EXECUTE ON procedure `directory_aziendale`.`trovaDipendentiScambiabili` TO
'dipendenteSettoreSpazi';
GRANT EXECUTE ON procedure `directory_aziendale`.`scambiaDipendenti` TO
'dipendenteSettoreSpazi';
GRANT EXECUTE ON procedure `directory_aziendale`.`trovaUfficiConPostazioneVuota`
TO 'dipendenteSettoreSpazi';
```

```
GRANT EXECUTE ON procedure
`directory_aziendale`.`assegnaDipendenteAPostazioneVuota` TO
'dipendenteSettoreSpazi';
GRANT DELETE, INSERT, SELECT, UPDATE ON TABLE `directory_aziendale`.`POSTAZIONE`
TO 'dipendenteSettoreSpazi';
CREATE USER 'dipendenteSettoreAmministrativo' IDENTIFIED BY
'DipendenteSettoreAmministrativo01!';

GRANT EXECUTE ON procedure `directory_aziendale`.`ricercaDipendente` TO
'dipendenteSettoreAmministrativo';
GRANT EXECUTE ON procedure `directory_aziendale`.`ricercaPerNumeroTelefono` TO
'dipendenteSettoreAmministrativo';
GRANT DELETE, INSERT, SELECT, UPDATE ON TABLE `directory_aziendale`.`DIPENDENTE`
TO 'dipendenteSettoreAmministrativo';
GRANT DELETE, INSERT, SELECT ON TABLE `directory_aziendale`.`MANSIONE` TO
'dipendenteSettoreAmministrativo';
GRANT EXECUTE ON procedure `directory_aziendale`.`cambiaMansioneDipendente` TO
'dipendenteSettoreAmministrativo';
GRANT EXECUTE ON procedure `directory_aziendale`.`assumiDipendente` TO
'dipendenteSettoreAmministrativo';
GRANT EXECUTE ON procedure `directory_aziendale`.`elencaTrasferimentiDipendente`
TO 'dipendenteSettoreAmministrativo';
GRANT DELETE, INSERT, SELECT, UPDATE ON TABLE
`directory_aziendale`.`DA_TRASFERIRE_A` TO 'dipendenteSettoreAmministrativo';
CREATE USER 'maintainer' IDENTIFIED BY 'Maintainer01!';

GRANT CREATE, DROP, GRANT OPTION, REFERENCES, EVENT, LOCK TABLES ON
directory_aziendale.* TO 'maintainer';
```

Codice del Front-End

```
// main.c

// Default headers
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <getopt.h>
#include <stdbool.h>
#include <errno.h>
```

```c
// Custom headers
#include "logger.h"
#include "controller.h"
#include "passwordAsker.h"
// Macros
#define SHORT_OPTS ""
void printOps(){
    logMsg(I, "Here is a list of supported operations. Please note that You must have the correct privileges for an operation in order to execute it. For executing an operation, type it in the format: opCode[:{arg0 | NULL}:...]\n");
    for (enum opCode c = op1; c < NUM_OPS; c ++){
        logMsg(I, "%s: %s(%s)\n", getOpString(c), getOpName(c), getOpParams(c));
    }
}

int main(int argc, char* argv[]){
    // variables
    int opt, connResult;
    int isPasswordRequired = 1;
    char *passwd = NULL;
    MYSQL *conn;
    // Constants
    const struct option longOptions[] = {
        {"nopasswd", no_argument, &isPasswordRequired, 0},
        {0, 0, 0, 0}
    };

    // greets user
    logMsg(I, "DirAz Thin Client - connector for directory_aziendale DB\n");
    // parses command line args
    while ((opt = getopt_long(argc, argv, SHORT_OPTS, longOptions, NULL)) != -1);
    // At least username must be provided
    if (argc - optind < 1){
        logMsg(I, "Usage: %s [%s] username\n", argv[0], longOptions[0].name);
        exit(EXIT_SUCCESS);
    }
    // Tries to connect to db with provided credentials until login succeeds or a signal is caught
    do {
        // asks for password
```

```
        if (isPasswordRequired && askPassword(&passwd)){
            logMsg(E, "failed to collect password\n");
        }
        // connects to db
        initController();
        connResult = connectToDB(argv[optind], passwd, &conn);
        if (connResult){
           logMsg(E, "connection to db failed. Check username and password\n");
            if (!isPasswordRequired){
                exit(EXIT_FAILURE);
            }
        }
        disposePassword(passwd);
    } while (connResult);
    logMsg(I, "Succesfully logged in as %s\n", argv[optind]);
    // lists available options
    printOps();
    char *input, *buf;
    char *inOpString = "";
    char *inOpArgs = "";
    size_t inputLen;
    int c;
    enum opCode selectedOpCode;
    while (true){
        // collects user's input and partially parses it
        logMsg(I, "Type here:\n");
        input = NULL;
        inputLen = 0;
        while (getline(&input, &inputLen, stdin) < 0){
            int err = errno;
            logMsg(E, "scanf: %s\n", strerror(err));
            while ((c = getchar()) != '\n' && c != EOF);
        }
        input[strlen(input) - 1] = '\0';
        buf = strtok(input, ARG_DEL);
        if (buf != NULL){
            inOpString = buf;
            inOpArgs = input + strlen(inOpString) + 1;
        }

        // calls matched op. No op could never have NUM_OPS has opCode, so it can
```

```
be used as a invalid op code too
        selectedOpCode = NUM_OPS;
        for (enum opCode op = op1; op < NUM_OPS; op ++){
            if (strcmp(inOpString, getOpString(op)) == 0){
                selectedOpCode = op;
                break;
            }
        }
        if (callOp(conn, selectedOpCode, inOpArgs)){
            logMsg(E, "Failed to execute %s\n", inOpString);
        }
        else{
            logMsg(I, "Done!\n");
        }
        // disposes of user input
        free(input);
    }
}

// controller.h

#ifndef CONTROLLER_H_INCLUDED
#define CONTROLLER_H_INCLUDED
#define ARG_DEL ":"
#define ARG_DATE_DEL "/"
#define ARG_NULL "NULL"
#include <mysql.h>

enum opCode {
    op1,
    op2_1,
    op2_2,
    op3_1,
    op3_2,
    op4,
    op5,
    op6,
    op7,
    op9,
    NUM_OPS     // must be last
```

```
};

void initController();   // must be called first
int connectToDB(char *username, char* passwd, MYSQL **conn);
int callOp(MYSQL* conn, const enum opCode, char *opArgs);
const char* getOpName(enum opCode code);
const char* getOpParams(enum opCode code);
const char* getOpString(enum opCode code);
#endif // CONTROLLER_H_INCLUDED

// controller.c

#include <errno.h>
#include <stdio.h>
#include <string.h>
#include <stdarg.h>
#include <stdbool.h>
#include <mysql_time.h>

#include "controller.h"
#include "logger.h"

#define MAX_LENGTH 1024

struct op {
    const char* name;
    const char* params;
    const char* stmt;
};

static const char* db_name = "directory_aziendale";
static const char* host_name = "localhost";
static struct op operations[NUM_OPS];
static char *opStrings[NUM_OPS];

void initController(){
    // populates opStrings with literal form of opCode
    opStrings[op1] = "op1";
    opStrings[op2_1] = "op2_1";
```

```
    opStrings[op2_2] = "op2_2";
    opStrings[op3_1] = "op3_1";
    opStrings[op3_2] = "op3_2";
    opStrings[op4] = "op4";
    opStrings[op5] = "op5";
    opStrings[op6] = "op6";
    opStrings[op7] = "op7";
    opStrings[op9] = "op9";

    // Populates structures in operations with data for each op. (TODO: data should be read from a file)
    operations[op1].name = "generaReportDaTrasferire";
    operations[op1].params = "";
    operations[op1].stmt = "call generaReportDaTrasferire()";

    operations[op2_1].name = "trovaDipendentiScambiabili";
    operations[op2_1].params = "cfDipendente";
    operations[op2_1].stmt = "call trovaDipendentiScambiabili(?)";

    operations[op2_2].name = "scambiaDipendenti";
    operations[op2_2].params = "cfDipendente1, cfDipendente2";
    operations[op2_2].stmt = "call scambiaDipendenti(?, ?)";

    operations[op3_1].name = "trovaUfficiConPostazioneVuota";
    operations[op3_1].params = "nomeMansione, nomeSettore";
    operations[op3_1].stmt = "call trovaUfficiConPostazioneVuota(?, ?)";

    operations[op3_2].name = "assegnaDipendenteAPostazioneVuota";
    operations[op3_2].params = "cfDipendente, numTelefonicoEsternoPostazioneVuota";
    operations[op3_2].stmt = "call assegnaDipendenteAPostazioneVuota(?, ?)";

    operations[op4].name = "cambiaMansioneDipendente";
    operations[op4].params = "cfDipendente, nomeNuovaMansione, nomeNuovoSettore";
    operations[op4].stmt = "call cambiaMansioneDipendente(?, ?, ?)";

    operations[op5].name = "elencaTrasferimentiDipendente";
    operations[op5].params = "cfDipendente";
    operations[op5].stmt = "call elencaTrasferimentiDipendente(?)";
```

```
    operations[op6].name = "ricercaDipendente";
    operations[op6].params = "nome, cognome";
    operations[op6].stmt = "call ricercaDipendente(?, ?)";

    operations[op7].name = "ricercaPerNumeroTelefono";
    operations[op7].params = "numTelefonoEsterno";
    operations[op7].stmt = "call ricercaPerNumeroTelefono(?)";

    operations[op9].name = "assumiDipendente";
    operations[op9].params = "cf, nome, cognome, luogoNascita, dataNascita,
emailPersonale, indirizzoResidenza, nomeMansione, nomeSettore";
    operations[op9].stmt = "call assumiDipendente(?, ?, ?, ?, ?, ?, ?, ?, ?)";

}

int prepareOp(MYSQL *conn, enum opCode op, MYSQL_STMT **stmtAddr){
    // init stmt
    if ((*stmtAddr = mysql_stmt_init(conn)) == NULL){
        logMsg(E, "mysql_stmt_init: %s\n", mysql_error(conn));
        return 1;
    }
    // prepares stmt
    MYSQL_STMT *stmt = *stmtAddr;
    if (mysql_stmt_prepare(stmt, operations[op].stmt,
strlen(operations[op].stmt)) != 0){
        logMsg(E, "mysql_stmt_prepare: %s\n", mysql_stmt_error(stmt));
        return 1;
    }
    return 0;
}

int launchOp(MYSQL *conn, MYSQL_STMT *stmt, MYSQL_BIND *inParams){
    // bind params
    if (inParams != NULL && mysql_stmt_bind_param(stmt, inParams)){
        logMsg(E, "mysql_stmt_bind_param: %s\n", mysql_stmt_error(stmt));
        return 1;
    }
```

```
    // execute statement
    if (mysql_stmt_execute(stmt)){
        logMsg(D, "%s\n", mysql_sqlstate(conn));
        logMsg(E, "mysql_stmt_execute: %s\n", mysql_stmt_error(stmt));
        return 1;
    }
    // buffers result set
    if (mysql_stmt_store_result(stmt) != 0){
        logMsg(E, "mysql_stmt_store_result: %s\n", mysql_stmt_error(stmt));
        return 1;
    }
    return 0;
}

int prepareAndLaunchOp(MYSQL *conn, enum opCode op, MYSQL_BIND *inParams,
MYSQL_STMT **stmtAddr){
    // prepares stmt
    if (prepareOp(conn, op, stmtAddr)){
        logMsg(E, "failed to prepare statement\n");
        return 1;
    }
    MYSQL_STMT *stmt = *stmtAddr;
    // launches op
    if (launchOp(conn, stmt, inParams)){
        logMsg(E, "failed to launch statement\n");
        return 1;
    }
    return 0;
}

int printRes(MYSQL_STMT* stmt, MYSQL_RES *metaRes, MYSQL_BIND *resultSetCols){
    mysql_field_seek(metaRes, 0);
    int resNumCol = mysql_num_fields(metaRes);
    int width[resNumCol], res;
    logMsg(I, "r)   ");
    for (int c = 0; c < resNumCol; c ++){
        printf("%s%n | ", mysql_fetch_field(metaRes) -> name, width + c);
    }
    printf("\n");
    fflush(stdout);
```

```
    for (int r = 0; ; r ++){
        if ((res = mysql_stmt_fetch(stmt)) == MYSQL_NO_DATA){
            break;
        }
        switch(res) {
            case 1:
                logMsg(E, "mysql_stmt_fetch: %d\n", mysql_stmt_error(stmt));
                return 1;
            case MYSQL_DATA_TRUNCATED:
                logMsg(W, "data truncation occurred\n", r);
            case 0:
                break;
        }
        logMsg(I, "%d) ", r);
        for (int c = 0; c < resNumCol; c ++){
            switch((resultSetCols + c) -> buffer_type){
            case MYSQL_TYPE_STRING:
            case MYSQL_TYPE_VAR_STRING:
            case MYSQL_TYPE_NEWDECIMAL:
                printf("%*s | ", width[c], (*((bool *) ((resultSetCols + c) ->
is_null)))? "NULL" : (char *) resultSetCols[c].buffer);
                break;
            case MYSQL_TYPE_TINY:
            case MYSQL_TYPE_SHORT:
            case MYSQL_TYPE_INT24:
            case MYSQL_TYPE_LONG:
            case MYSQL_TYPE_LONGLONG:
                if (*((bool *) ((resultSetCols + c) -> is_null))){
                    printf("%*s | ", width[c], "NULL");
                }
                else {
                    printf("%*d | ", width[c], *((int *)
(resultSetCols[c].buffer)));
                }
                break;
            case MYSQL_TYPE_FLOAT:
            case MYSQL_TYPE_DOUBLE:
                if (*((bool *) ((resultSetCols + c) -> is_null))){
                    printf("%*s | ", width[c], "NULL");
                }
                else {
```

```
                    printf("%*f | ", width[c], *((double *)
(resultSetCols[c].buffer)));
                }
                break;
            case MYSQL_TYPE_DATE:
                if (*((bool *) ((resultSetCols + c) -> is_null))){
                    printf("%s/ ", "NULL");
                }
                else {
                    printf("%d/%d/%d | ",
                        ((MYSQL_TIME *)(resultSetCols[c].buffer)) -> day,
                        ((MYSQL_TIME *)(resultSetCols[c].buffer)) -> month,
                        ((MYSQL_TIME *)(resultSetCols[c].buffer)) -> year
                    );
                }
                break;
            default:
                printf("(not supported) | ");
            }
        }
        printf("\n");
    }
    fflush(stdout);

    return 0;
}

void freeResultSet(MYSQL_BIND *resultSetCols, int resNumCol){
   for (int i = 0; i < resNumCol; i ++){
        free(resultSetCols[i].buffer);
        free(resultSetCols[i].length);
        free(resultSetCols[i].is_null);
        free(resultSetCols[i].error);
    }
    free(resultSetCols);
}

MYSQL_BIND *callocResultSetCols(MYSQL_RES *metaRes){
    int resNumCol = mysql_num_fields(metaRes);
    MYSQL_BIND *resultSetCols = calloc(resNumCol, sizeof(MYSQL_BIND));
```

```
    MYSQL_FIELD *currentField;

    mysql_field_seek(metaRes, 0);
    for (int i = 0; i < resNumCol; i ++){
    currentField = mysql_fetch_field(metaRes);
    resultSetCols[i].buffer_type = currentField -> type;
    resultSetCols[i].buffer = calloc(MAX_LENGTH, sizeof(char));
    resultSetCols[i].buffer_length  = MAX_LENGTH;
    resultSetCols[i].length = (unsigned long*) calloc(1, sizeof(unsigned long));
    resultSetCols[i].is_null = (bool *) calloc(1, sizeof(bool));
    resultSetCols[i].error = (bool *) calloc(1, sizeof(bool));
    }
    return resultSetCols;
}

int bindRes(MYSQL_STMT *stmt, MYSQL_BIND **resultSetColsAddr, MYSQL_RES
**metaResAddr){
    MYSQL_RES *metaRes;
    MYSQL_BIND *resultSetCols;
    int resNumCol;
    int numRes = 0;
    if ((*metaResAddr = mysql_stmt_result_metadata(stmt)) == NULL){
            logMsg(E, "mysql_stmt_result_metadata: %s\n",
mysql_stmt_error(stmt));
            return -1;
    }
    metaRes = *metaResAddr;
    resNumCol = mysql_num_fields(metaRes);
    if (resNumCol > 0){
        numRes ++;
        // binds result set dinamically
        *resultSetColsAddr = callocResultSetCols(metaRes);
        resultSetCols = *resultSetColsAddr;
        if (mysql_stmt_bind_result(stmt, resultSetCols)){
            logMsg(E, "mysql_stmt_bind_result: %s\n", mysql_stmt_error(stmt));
            return -1;
        }
    }
    return numRes;
}
```

```
int callOp1(MYSQL *conn){
    MYSQL_STMT *stmt;
    MYSQL_BIND *resSet;
    MYSQL_RES *metaRes;
    int hasNext;

    // prepare and launches stmt
    if (prepareAndLaunchOp(conn, op1, NULL, &stmt)){
        logMsg(E, "failed to prepare and launch statement\n");
        return 1;
    }

    // binds res set
    if (bindRes(stmt, &resSet, &metaRes) <= 0){
        logMsg(W, "Either failed to bind a result set or no result set was
available to bind\n");
    }

    // prints res set;
    if (printRes(stmt, metaRes, resSet)){
        logMsg(E, "Error while printing results\n");
        return 1;
    }

    // discards remaining result sets
    do {mysql_stmt_free_result(stmt);} while ((hasNext =
mysql_stmt_next_result(stmt) == 0));
    if (hasNext > 0){
        logMsg(E, "mysql_stmt_next_result: %s\n", mysql_stmt_error(stmt));
        return 1;
    }

    // frees memory allocated dinamically
    freeResultSet(resSet, mysql_num_fields(metaRes));
    mysql_free_result(metaRes);
    if (mysql_stmt_close(stmt) != 0){
        logMsg(E, "mysql_stmt_close: %s\n", mysql_error(conn));
        return 1;
    }
```

```
    return 0;
}

int callOp2_1(MYSQL *conn, int numOfArgs, char *cfDipendente){
    MYSQL_STMT *stmt;
    MYSQL_BIND *resSet;
    MYSQL_BIND *inParams = calloc(numOfArgs, sizeof(MYSQL_BIND));
    MYSQL_RES *metaRes;
    int hasNext;
    int curParam = 0;
    unsigned long len[numOfArgs];
    bool isNull[numOfArgs];

    // prepares params

    memset(isNull, false, sizeof(bool) * numOfArgs);

    if (cfDipendente == NULL || !strcmp(cfDipendente, ARG_NULL)){
        cfDipendente = "";
        isNull[curParam] = true;
        inParams -> is_null = isNull + curParam;
    }
    len[curParam] = sizeof(char) * strlen(cfDipendente);
    (inParams + curParam) -> buffer_type = MYSQL_TYPE_STRING;
    (inParams + curParam) -> buffer = cfDipendente;
    (inParams + curParam) -> buffer_length = len[curParam];
    (inParams + curParam) -> length = len + curParam;
    curParam ++;

    // prepare and launches stmt
    if (prepareAndLaunchOp(conn, op2_1, inParams, &stmt)){
        logMsg(E, "failed to prepare and launch statement\n");
        return 1;
    }

    // binds res set
    if (bindRes(stmt, &resSet, &metaRes) <= 0){
        logMsg(W, "Either failed to bind a result set or no result set was
```

```
available to bind\n");
    }

    // prints res set;
    if (printRes(stmt, metaRes, resSet)){
        logMsg(E, "Error while printing results\n");
        return 1;
    }
    // discards remaining result sets
    do {mysql_stmt_free_result(stmt);} while ((hasNext =
mysql_stmt_next_result(stmt) == 0));
    if (hasNext > 0){
        logMsg(E, "mysql_stmt_next_result: %s\n", mysql_stmt_error(stmt));
        return 1;
    }

    // frees memory allocated dinamically
    free(inParams);
    freeResultSet(resSet, mysql_num_fields(metaRes));
    mysql_free_result(metaRes);
    if (mysql_stmt_close(stmt) != 0){
        logMsg(E, "mysql_stmt_close: %s\n", mysql_error(conn));
        return 1;
    }

    return 0;
}

int callOp2_2(MYSQL *conn, int numOfArgs, char *cfDipendente1, char
*cfDipendente2){
    MYSQL_STMT *stmt;
    MYSQL_BIND *inParams = calloc(numOfArgs, sizeof(MYSQL_BIND));
    int hasNext;
    int curParam = 0;
    unsigned long len[numOfArgs];
    bool isNull[numOfArgs];

    // prepares params

    memset(isNull, false, sizeof(bool) * numOfArgs);
```

```
    if (cfDipendente1 == NULL || !strcmp(cfDipendente1, ARG_NULL)){
        cfDipendente1 = "";
        isNull[curParam] = true;
        inParams -> is_null = isNull + curParam;
    }
    len[curParam] = sizeof(char) * strlen(cfDipendente1);
    (inParams + curParam) -> buffer_type = MYSQL_TYPE_STRING;
    (inParams + curParam) -> buffer = cfDipendente1;
    (inParams + curParam)-> buffer_length = len[curParam];
    (inParams + curParam) -> length = len + curParam;
    curParam ++;

    if (cfDipendente2 == NULL || !strcmp(cfDipendente2, ARG_NULL)){
        cfDipendente2 = "";
        isNull[curParam] = true;
        (inParams + curParam) -> is_null = isNull + curParam;
    }
    len[curParam] = sizeof(char) * strlen(cfDipendente2);
    (inParams + curParam) -> buffer_type = MYSQL_TYPE_STRING;
    (inParams + curParam) -> buffer = cfDipendente2;
    (inParams + curParam) -> buffer_length = len[curParam];
    (inParams + curParam) -> length = len + curParam;
    curParam ++;

    // prepare and launches stmt
    if (prepareAndLaunchOp(conn, op2_2, inParams, &stmt)){
        logMsg(E, "failed to prepare and launch statement\n");
        return 1;
    }

    // no res set has to be printed

    // discards remaining result sets
    do {mysql_stmt_free_result(stmt);} while ((hasNext =
mysql_stmt_next_result(stmt) == 0));
    if (hasNext > 0){
        logMsg(E, "mysql_stmt_next_result: %s\n", mysql_stmt_error(stmt));
        return 1;
    }
```

```
    // frees memory allocated dinamically
    free(inParams);
    if (mysql_stmt_close(stmt) != 0){
        logMsg(E, "mysql_stmt_close: %s\n", mysql_error(conn));
        return 1;
    }

    return 0;
}

int callOp3_1(MYSQL *conn, int numOfArgs, char *nomeMansione, char *nomeSettore){
    MYSQL_STMT *stmt;
    MYSQL_BIND *resSet;
    MYSQL_BIND *inParams = calloc(numOfArgs, sizeof(MYSQL_BIND));
    MYSQL_RES *metaRes;
    int hasNext;
    int curParam = 0;
    unsigned long len[numOfArgs];
    bool isNull[numOfArgs];

    // prepares params

    memset(isNull, false, sizeof(bool) * numOfArgs);

    if (nomeMansione == NULL || !strcmp(nomeMansione, ARG_NULL)){
        nomeMansione = "";
        isNull[curParam] = true;
        inParams -> is_null = isNull + curParam;
    }
    len[curParam] = sizeof(char) * strlen(nomeMansione);
    (inParams + curParam) -> buffer_type = MYSQL_TYPE_STRING;
    (inParams + curParam) -> buffer = nomeMansione;
    (inParams + curParam) -> buffer_length = len[curParam];
    (inParams + curParam) -> length = len + curParam;
    curParam ++;

    if (nomeSettore == NULL || !strcmp(nomeSettore, ARG_NULL)){
        nomeSettore = "";
```

```
        isNull[curParam] = true;
        (inParams + curParam) -> is_null = isNull + curParam;
    }
    len[curParam] = sizeof(char) * strlen(nomeSettore);
    (inParams + curParam) -> buffer_type = MYSQL_TYPE_STRING;
    (inParams + curParam) -> buffer = nomeSettore;
    (inParams + curParam) -> buffer_length = len[curParam];
    (inParams + curParam) -> length = len + curParam;
    curParam ++;

    // prepare and launches stmt
    if (prepareAndLaunchOp(conn, op3_1, inParams, &stmt)){
        logMsg(E, "failed to prepare and launch statement\n");
        return 1;
    }

    // binds res set
    if (bindRes(stmt, &resSet, &metaRes) <= 0){
        logMsg(W, "Either failed to bind a result set or no result set was
available to bind\n");
    }

    // prints res set;
    if (printRes(stmt, metaRes, resSet)){
        logMsg(E, "Error while printing results\n");
        return 1;
    }

    // discards remaining result sets
    do {mysql_stmt_free_result(stmt);} while ((hasNext =
mysql_stmt_next_result(stmt) == 0));
    if (hasNext > 0){
        logMsg(E, "mysql_stmt_next_result: %s\n", mysql_stmt_error(stmt));
        return 1;
    }

    // frees memory allocated dinamically
    free(inParams);
    freeResultSet(resSet, mysql_num_fields(metaRes));
    mysql_free_result(metaRes);
```

```
    if (mysql_stmt_close(stmt) != 0){
        logMsg(E, "mysql_stmt_close: %s\n", mysql_error(conn));
        return 1;
    }

    return 0;
}

int callOp3_2(MYSQL *conn, int numOfArgs, char *cfDipendente, char
*numTelefonicoEsternoPostazione){
    MYSQL_STMT *stmt;
    MYSQL_BIND *inParams = calloc(numOfArgs, sizeof(MYSQL_BIND));
    int hasNext;
    int curParam = 0;
    unsigned long len[numOfArgs];
    bool isNull[numOfArgs];

    // prepares params

    memset(isNull, false, sizeof(bool) * numOfArgs);

    if (cfDipendente == NULL || !strcmp(cfDipendente, ARG_NULL)){
        cfDipendente = "";
        isNull[curParam] = true;
        inParams -> is_null = isNull + curParam;
    }
    len[curParam] = sizeof(char) * strlen(cfDipendente);
    (inParams + curParam) -> buffer_type = MYSQL_TYPE_STRING;
    (inParams + curParam) -> buffer = cfDipendente;
    (inParams + curParam) -> buffer_length = len[curParam];
    (inParams + curParam) -> length = len + curParam;
    curParam ++;

    if (numTelefonicoEsternoPostazione == NULL || !
strcmp(numTelefonicoEsternoPostazione, ARG_NULL)){
        numTelefonicoEsternoPostazione = "";
        isNull[curParam] = true;
        (inParams + curParam) -> is_null = isNull + curParam;
    }
    len[curParam] = sizeof(char) * strlen(numTelefonicoEsternoPostazione);
```

```
    (inParams + curParam) -> buffer_type = MYSQL_TYPE_STRING;
    (inParams + curParam) -> buffer = numTelefonicoEsternoPostazione;
    (inParams + curParam) -> buffer_length = len[curParam];
    (inParams + curParam) -> length = len + curParam;
    curParam ++;

    // prepare and launches stmt
    if (prepareAndLaunchOp(conn, op3_2, inParams, &stmt)){
        logMsg(E, "failed to prepare and launch statement\n");
        return 1;
    }

    // discards remaining result sets
    do {mysql_stmt_free_result(stmt);} while ((hasNext =
mysql_stmt_next_result(stmt) == 0));
    if (hasNext > 0){
        logMsg(E, "mysql_stmt_next_result: %s\n", mysql_stmt_error(stmt));
        return 1;
    }

    // frees memory allocated dinamically
    free(inParams);
    if (mysql_stmt_close(stmt) != 0){
        logMsg(E, "mysql_stmt_close: %s\n", mysql_error(conn));
        return 1;
    }

    return 0;
}

int callOp4(MYSQL *conn, int numOfArgs, char *cfDipendente, char
*nomeNuovaMansione, char* nomeNuovoSettore){
    MYSQL_STMT *stmt;
    MYSQL_BIND *inParams = calloc(numOfArgs, sizeof(MYSQL_BIND));
    int hasNext;
    int curParam = 0;
    unsigned long len[numOfArgs];
    bool isNull[numOfArgs];

    // prepares params
```

```
    memset(isNull, false, sizeof(bool) * numOfArgs);

    if (cfDipendente == NULL || !strcmp(cfDipendente, ARG_NULL)){
        cfDipendente = "";
        isNull[curParam] = true;
        inParams -> is_null = isNull + curParam;
    }
    len[curParam] = sizeof(char) * strlen(cfDipendente);
    (inParams + curParam) -> buffer_type = MYSQL_TYPE_STRING;
    (inParams + curParam) -> buffer = cfDipendente;
    (inParams + curParam) -> buffer_length = len[curParam];
    (inParams + curParam) -> length = len + curParam;
    curParam ++;

    if (nomeNuovaMansione == NULL || !strcmp(nomeNuovaMansione, ARG_NULL)){
        nomeNuovaMansione = "";
        isNull[curParam] = true;
        (inParams + curParam) -> is_null = isNull + curParam;
    }
    len[curParam] = sizeof(char) * strlen(nomeNuovaMansione);
    (inParams + curParam) -> buffer_type = MYSQL_TYPE_STRING;
    (inParams + curParam) -> buffer = nomeNuovaMansione;
    (inParams + curParam) -> buffer_length = len[curParam];
    (inParams + curParam) -> length = len + curParam;
    curParam ++;

    if (nomeNuovoSettore == NULL || !strcmp(nomeNuovoSettore, ARG_NULL)){
        nomeNuovoSettore = "";
        isNull[curParam] = true;
        (inParams + curParam) -> is_null = isNull + curParam;
    }
    len[curParam] = sizeof(char) * strlen(nomeNuovoSettore);
    (inParams + curParam) -> buffer_type = MYSQL_TYPE_STRING;
    (inParams + curParam) -> buffer = nomeNuovoSettore;
    (inParams + curParam) -> buffer_length = len[curParam];
    (inParams + curParam) -> length = len + curParam;
    curParam ++;

    // prepare and launches stmt
```

```
    if (prepareAndLaunchOp(conn, op4, inParams, &stmt)){
        logMsg(E, "failed to prepare and launch statement\n");
        return 1;
    }

    // discards remaining result sets
    do {mysql_stmt_free_result(stmt);} while ((hasNext =
mysql_stmt_next_result(stmt) == 0));
    if (hasNext > 0){
        logMsg(E, "mysql_stmt_next_result: %s\n", mysql_stmt_error(stmt));
        return 1;
    }

    // frees memory allocated dinamically
    free(inParams);
    if (mysql_stmt_close(stmt) != 0){
        logMsg(E, "mysql_stmt_close: %s\n", mysql_error(conn));
        return 1;
    }

    return 0;
}

int callOp5(MYSQL *conn, int numOfArgs, char *cfDipendente){
    MYSQL_STMT *stmt;
    MYSQL_BIND *resSet;
    MYSQL_BIND *inParams = calloc(numOfArgs, sizeof(MYSQL_BIND));
    MYSQL_RES *metaRes;
    int hasNext;
    int curParam = 0;
    unsigned long len[numOfArgs];
    bool isNull[numOfArgs];

    // prepares params

    memset(isNull, false, sizeof(bool) * numOfArgs);

    if (cfDipendente == NULL || !strcmp(cfDipendente, ARG_NULL)){
        cfDipendente = "";
```

```
            isNull[curParam] = true;
            inParams -> is_null = isNull + curParam;
        }
        len[curParam] = sizeof(char) * strlen(cfDipendente);
        (inParams + curParam) -> buffer_type = MYSQL_TYPE_STRING;
        (inParams + curParam) -> buffer = cfDipendente;
        (inParams + curParam) -> buffer_length = len[curParam];
        (inParams + curParam) -> length = len + curParam;
        curParam ++;

        // prepare and launches stmt
        if (prepareAndLaunchOp(conn, op5, inParams, &stmt)){
            logMsg(E, "failed to prepare and launch statement\n");
            return 1;
        }



        // binds res set
        if (bindRes(stmt, &resSet, &metaRes) <= 0){
            logMsg(W, "Either failed to bind a result set or no result set was
available to bind\n");
        }

        // prints res set;
        if (printRes(stmt, metaRes, resSet)){
            logMsg(E, "Error while printing results\n");
            return 1;
        }
        // discards remaining result sets
        do {mysql_stmt_free_result(stmt);} while ((hasNext =
mysql_stmt_next_result(stmt) == 0));
        if (hasNext > 0){
            logMsg(E, "mysql_stmt_next_result: %s\n", mysql_stmt_error(stmt));
            return 1;
        }

        // frees memory allocated dinamically
        free(inParams);
        freeResultSet(resSet, mysql_num_fields(metaRes));
        mysql_free_result(metaRes);
```

```
    if (mysql_stmt_close(stmt) != 0){
        logMsg(E, "mysql_stmt_close: %s\n", mysql_error(conn));
        return 1;
    }

    return 0;
}

int callOp6(MYSQL *conn, int numOfArgs, char *nome, char* cognome){
    MYSQL_STMT *stmt;
    MYSQL_BIND *resSet;
    MYSQL_BIND *inParams = calloc(numOfArgs, sizeof(MYSQL_BIND));
    MYSQL_RES *metaRes;
    int hasNext;
    int curParam = 0;
    unsigned long len[numOfArgs];
    bool isNull[numOfArgs];

    logMsg(D, "args are: %s, %s\n", nome, cognome);

    // prepares params

    memset(isNull, false, sizeof(bool) * numOfArgs);

    if (nome == NULL || !strcmp(nome, ARG_NULL)){
        nome = "";
        isNull[curParam] = true;
        inParams -> is_null = isNull + curParam;
    }
    len[curParam] = sizeof(char) * strlen(nome);
    (inParams + curParam) -> buffer_type = MYSQL_TYPE_STRING;
    (inParams + curParam) -> buffer = nome;
    (inParams + curParam) -> buffer_length = len[curParam];
    (inParams + curParam) -> length = len + curParam;
    curParam ++;

    if (cognome == NULL || !strcmp(cognome, ARG_NULL)){
        cognome = "";
        isNull[curParam] = true;
```

```
        (inParams + curParam) -> is_null = isNull + curParam;
    }
    len[curParam] = sizeof(char) * strlen(cognome);
    (inParams + curParam) -> buffer_type = MYSQL_TYPE_STRING;
    (inParams + curParam) -> buffer = cognome;
    (inParams+ curParam) -> buffer_length = len[curParam];
    (inParams + curParam) -> length = len + curParam;
    curParam ++;

    // prepare and launches stmt
    if (prepareAndLaunchOp(conn, op6, inParams, &stmt)){
        logMsg(E, "failed to prepare and launch statement\n");
        return 1;
    }

    // binds res set
    if (bindRes(stmt, &resSet, &metaRes) <= 0){
        logMsg(W, "Either failed to bind a result set or no result set was
available to bind\n");
    }

    // prints res set;
    if (printRes(stmt, metaRes, resSet)){
        logMsg(E, "Error while printing results\n");
        return 1;
    }
    // discards remaining result sets
    do {mysql_stmt_free_result(stmt);} while ((hasNext =
mysql_stmt_next_result(stmt) == 0));
    if (hasNext > 0){
        logMsg(E, "mysql_stmt_next_result: %s\n", mysql_stmt_error(stmt));
        return 1;
    }

    // frees memory allocated dinamically
    free(inParams);
    freeResultSet(resSet, mysql_num_fields(metaRes));
    mysql_free_result(metaRes);
    if (mysql_stmt_close(stmt) != 0){
        logMsg(E, "mysql_stmt_close: %s\n", mysql_error(conn));
```

```
        return 1;
    }

    return 0;
}

int callOp7(MYSQL *conn, int numOfArgs, char *numTelefonoEsterno){
    MYSQL_STMT *stmt;
    MYSQL_BIND *resSet;
    MYSQL_BIND *inParams = calloc(numOfArgs, sizeof(MYSQL_BIND));
    MYSQL_RES *metaRes;
    int hasNext;
    int curParam = 0;
    unsigned long len[numOfArgs];
    bool isNull[numOfArgs];

    // prepares params

    memset(isNull, false, sizeof(bool) * numOfArgs);

    if (numTelefonoEsterno == NULL || !strcmp(numTelefonoEsterno, ARG_NULL)){
        numTelefonoEsterno = "";
        isNull[curParam] = true;
        inParams -> is_null = isNull + curParam;
    }
    len[curParam] = sizeof(char) * strlen(numTelefonoEsterno);
    (inParams + curParam) -> buffer_type = MYSQL_TYPE_STRING;
    (inParams + curParam)-> buffer = numTelefonoEsterno;
    (inParams + curParam)-> buffer_length = len[curParam];
    (inParams + curParam)-> length = len + curParam;
    curParam ++;

    // prepare and launches stmt
    if (prepareAndLaunchOp(conn, op7, inParams, &stmt)){
        logMsg(E, "failed to prepare and launch statement\n");
        return 1;
    }

    // binds res set
```

```
    if (bindRes(stmt, &resSet, &metaRes) <= 0){
        logMsg(W, "Either failed to bind a result set or no result set was
available to bind\n");
    }

    // prints res set;
    if (printRes(stmt, metaRes, resSet)){
        logMsg(E, "Error while printing results\n");
        return 1;
    }
    // discards remaining result sets
    do {mysql_stmt_free_result(stmt);} while ((hasNext =
mysql_stmt_next_result(stmt) == 0));
    if (hasNext > 0){
        logMsg(E, "mysql_stmt_next_result: %s\n", mysql_stmt_error(stmt));
        return 1;
    }

    // frees memory allocated dinamically
    free(inParams);
    freeResultSet(resSet, mysql_num_fields(metaRes));
    mysql_free_result(metaRes);
    if (mysql_stmt_close(stmt) != 0){
        logMsg(E, "mysql_stmt_close: %s\n", mysql_error(conn));
        return 1;
    }

    return 0;
}

int callOp9(MYSQL *conn, int numOfArgs, char *cf, char *nome, char *cognome, char
*luogoNascita, MYSQL_TIME *dataNascita, char *emailPersonale, char
*indirizzoResidenza, char *nomeMansione, char *nomeSettore){
    MYSQL_STMT *stmt;
    MYSQL_BIND *inParams = calloc(numOfArgs, sizeof(MYSQL_BIND));
    int hasNext;
    int curParam = 0;
    unsigned long len[numOfArgs];
    bool isNull[numOfArgs];

```

```
    // prepares params

    memset(isNull, false, sizeof(bool) * numOfArgs);

    if (cf == NULL || !strcmp(cf, ARG_NULL)){
        cf = "";
        isNull[curParam] = true;
        inParams -> is_null = isNull + curParam;
    }
    len[curParam] = sizeof(char) * strlen(cf);
    (inParams + curParam) -> buffer_type = MYSQL_TYPE_STRING;
    (inParams + curParam) -> buffer = cf;
    (inParams + curParam) -> buffer_length = len[curParam];
    (inParams + curParam) -> length = len + curParam;
    curParam ++;

    if (nome == NULL || !strcmp(nome, ARG_NULL)){
        nome = "";
        isNull[curParam] = true;
        inParams -> is_null = isNull + curParam;
    }
    len[curParam] = sizeof(char) * strlen(nome);
    (inParams + curParam) -> buffer_type = MYSQL_TYPE_STRING;
    (inParams + curParam) -> buffer = nome;
    (inParams + curParam) -> buffer_length = len[curParam];
    (inParams + curParam) -> length = len + curParam;
    curParam ++;

    if (cognome == NULL || !strcmp(cognome, ARG_NULL)){
        cognome = "";
        isNull[curParam] = true;
        inParams -> is_null = isNull + curParam;
    }
    len[curParam] = sizeof(char) * strlen(cognome);
    (inParams + curParam) -> buffer_type = MYSQL_TYPE_STRING;
    (inParams + curParam) -> buffer = cognome;
    (inParams + curParam) -> buffer_length = len[curParam];
    (inParams + curParam) -> length = len + curParam;
    curParam ++;

```

```
    if (luogoNascita == NULL || !strcmp(luogoNascita, ARG_NULL)){
        luogoNascita = "";
        isNull[curParam] = true;
        inParams -> is_null = isNull + curParam;
    }
    len[curParam] = sizeof(char) * strlen(luogoNascita);
    (inParams + curParam) -> buffer_type = MYSQL_TYPE_STRING;
    (inParams + curParam) -> buffer = luogoNascita;
    (inParams + curParam) -> buffer_length = len[curParam];
    (inParams + curParam) -> length = len + curParam;
    curParam ++;

    if ((dataNascita -> year) == 0 || (dataNascita -> month) == 0 || (dataNascita
-> day) == 0){
        // FIXME: Expected: inserting a null date; Actual: [E]
mysql_stmt_execute: Incorrect date value: '' for column 'dataNascita' at row 1
        (inParams + curParam) -> buffer_type = MYSQL_TYPE_NULL;
    }
    else{
    len[curParam] = sizeof(MYSQL_TIME);
    (inParams + curParam) -> buffer_type = MYSQL_TYPE_DATE;
    (inParams + curParam) -> buffer = dataNascita;
    (inParams + curParam) -> buffer_length = len[curParam];
    (inParams + curParam) -> length = len + curParam;
    curParam ++;
    }

    if (emailPersonale == NULL || !strcmp(emailPersonale, ARG_NULL)){
        emailPersonale = "";
        isNull[curParam] = true;
        inParams -> is_null = isNull + curParam;
    }
    len[curParam] = sizeof(char) * strlen(emailPersonale);
    (inParams + curParam) -> buffer_type = MYSQL_TYPE_STRING;
    (inParams + curParam) -> buffer = emailPersonale;
    (inParams + curParam) -> buffer_length = len[curParam];
    (inParams + curParam) -> length = len + curParam;
    curParam ++;

    if (indirizzoResidenza == NULL || !strcmp(indirizzoResidenza, ARG_NULL)){
```

```
        indirizzoResidenza = "";
        isNull[curParam] = true;
        inParams -> is_null = isNull + curParam;
    }
    len[curParam] = sizeof(char) * strlen(indirizzoResidenza);
    (inParams + curParam) -> buffer_type = MYSQL_TYPE_STRING;
    (inParams + curParam) -> buffer = indirizzoResidenza;
    (inParams + curParam) -> buffer_length = len[curParam];
    (inParams + curParam) -> length = len + curParam;
    curParam ++;

    if (nomeMansione == NULL || !strcmp(nomeMansione, ARG_NULL)){
        nomeMansione = "";
        isNull[curParam] = true;
        inParams -> is_null = isNull + curParam;
    }
    len[curParam] = sizeof(char) * strlen(nomeMansione);
    (inParams + curParam) -> buffer_type = MYSQL_TYPE_STRING;
    (inParams + curParam) -> buffer = nomeMansione;
    (inParams + curParam) -> buffer_length = len[curParam];
    (inParams + curParam) -> length = len + curParam;
    curParam ++;

    if (nomeSettore == NULL || !strcmp(nomeSettore, ARG_NULL)){
        nomeSettore = "";
        isNull[curParam] = true;
        inParams -> is_null = isNull + curParam;
    }
    len[curParam] = sizeof(char) * strlen(nomeSettore);
    (inParams + curParam) -> buffer_type = MYSQL_TYPE_STRING;
    (inParams + curParam) -> buffer = nomeSettore;
    (inParams + curParam) -> buffer_length = len[curParam];
    (inParams + curParam) -> length = len + curParam;
    curParam ++;

    // prepare and launches stmt
    if (prepareAndLaunchOp(conn, op9, inParams, &stmt)){
        logMsg(E, "failed to prepare and launch statement\n");
        return 1;
    }
```

```
    // discards remaining result sets
    do {mysql_stmt_free_result(stmt);} while ((hasNext =
mysql_stmt_next_result(stmt) == 0));
    if (hasNext > 0){
        logMsg(E, "mysql_stmt_next_result: %s\n", mysql_stmt_error(stmt));
        return 1;
    }

    // frees memory allocated dinamically
    free(inParams);
    if (mysql_stmt_close(stmt) != 0){
        logMsg(E, "mysql_stmt_close: %s\n", mysql_error(conn));
        return 1;
    }

    return 0;
}


int callOp(MYSQL *conn, const enum opCode op, char *opArgs){
    int res;
    int numOfArgs = 1;
    char **strArgs;
    MYSQL_TIME *dateArgs = NULL;
    switch(op){
        case op1:
            res = callOp1(conn);
            break;
        case op2_1:
            ;
            res = callOp2_1(conn, numOfArgs, strtok(opArgs, ARG_DEL));
            break;
        case op2_2:
            ;
            numOfArgs = 2;
            strArgs = calloc(numOfArgs, sizeof(char*));
            strArgs[0] = strtok(opArgs, ARG_DEL);
            for(int i = 1; i < numOfArgs; i ++){
                strArgs[i] = strtok(NULL, ARG_DEL);
```

```
            }
            res = callOp2_2(conn, numOfArgs, strArgs[0], strArgs[1]);
            free(strArgs);
            break;
        case op3_1:
            ;
            numOfArgs = 2;
            strArgs = calloc(numOfArgs, sizeof(char*));
            strArgs[0] = strtok(opArgs, ARG_DEL);
            for(int i = 1; i < numOfArgs; i ++){
                strArgs[i] = strtok(NULL, ARG_DEL);
            }
            res = callOp3_1(conn, numOfArgs, strArgs[0], strArgs[1]);
            free(strArgs);
            break;
        case op3_2:
            ;
            numOfArgs = 2;
            strArgs = calloc(numOfArgs, sizeof(char*));
            strArgs[0] = strtok(opArgs, ARG_DEL);
            for(int i = 1; i < numOfArgs; i ++){
                strArgs[i] = strtok(NULL, ARG_DEL);
            }
            res = callOp3_2(conn, numOfArgs, strArgs[0], strArgs[1]);
            free(strArgs);
            break;
        case op4:
            ;
            numOfArgs = 3;
            strArgs = calloc(numOfArgs, sizeof(char*));
            strArgs[0] = strtok(opArgs, ARG_DEL);
            for(int i = 1; i < numOfArgs; i ++){
                strArgs[i] = strtok(NULL, ARG_DEL);
            }
            res = callOp4(conn, numOfArgs, strArgs[0], strArgs[1], strArgs[2]);
            free(strArgs);
            break;
        case op5:
            ;
            res = callOp5(conn, numOfArgs, strtok(opArgs, ARG_DEL));
            break;
```

```
        case op6:
            ;
            numOfArgs = 2;
            strArgs = calloc(numOfArgs, sizeof(char*));
            strArgs[0] = strtok(opArgs, ARG_DEL);
            for(int i = 1; i < numOfArgs; i ++){
                strArgs[i] = strtok(NULL, ARG_DEL);
            }
            res = callOp6(conn, numOfArgs, strArgs[0], strArgs[1]);
            free(strArgs);
            break;
        case op7:
            ;
            res = callOp7(conn, numOfArgs, strtok(opArgs, ARG_DEL));
            break;
        case op9:
            ;
            char *bufDate = NULL;
            int bufNum;
            numOfArgs = 9;
            // parses string arguments
            strArgs = calloc(numOfArgs, sizeof(char*));
            strArgs[0] = strtok(opArgs, ARG_DEL);
            for(int i = 1; i < numOfArgs; i ++){
                if (i != 4){
                    strArgs[i] = strtok(NULL, ARG_DEL);
                } else{
                    strArgs[i] = strtok(NULL, ARG_DEL);
                }
            }
            // parses date arguments
            dateArgs = calloc(1, sizeof(MYSQL_TIME));
            if ((bufDate = strtok(bufDate, ARG_DATE_DEL)) == NULL || !
strcmp(bufDate, ARG_NULL)){}
            else {
                sscanf(bufDate, "%d", &bufNum);
                dateArgs -> day = bufNum;
                }
            if ((bufDate = strtok(NULL, ARG_DATE_DEL)) == NULL || !
strcmp(bufDate, ARG_NULL)){}
            else{
```

```
                sscanf(bufDate, "%d", &bufNum);
                dateArgs -> month = bufNum;
            }
            if ((bufDate = strtok(NULL, ARG_DATE_DEL)) == NULL || !
5   strcmp(bufDate, ARG_NULL)){}
            else{
                sscanf(bufDate, "%d", &bufNum);
                dateArgs -> year = bufNum;
            }
10          res = callOp9(conn, numOfArgs, strArgs[0], strArgs[1], strArgs[2],
    strArgs[3], dateArgs, strArgs[5], strArgs[6], strArgs[7], strArgs[8]);
            free(strArgs);
            if (dateArgs != NULL){
                free(dateArgs);
15          }
            break;
        default:
            logMsg(E, "There is no such operation with provided opCode\n");
            return 1;
20  }
    return res;
}

int connectToDB(char *username, char* passwd, MYSQL** connAddr){
25  // Initialize connection
    if ((*connAddr = mysql_init(NULL)) == NULL){
        int err = errno;
        logMsg(E, "mysql_init: %s\n", strerror(err));
        return 1;
30  }
    MYSQL *conn = *connAddr;
    // Tries to connect with db. NULL values are read from settings file
    if ((mysql_real_connect(conn,
                        host_name,
35                      username,
                        passwd,
                        db_name,
                        0, // port number
                        NULL, // socket name
40                      CLIENT_MULTI_STATEMENTS )) == NULL){
        logMsg(E, "mysql_real_connect: %s\n", mysql_error(conn));
```

```
        return 1;
    }
    return 0;
}

const char* getOpName(enum opCode code){
    return operations[code].name;
}

const char* getOpParams(enum opCode code){
    return operations[code].params;
}

const char* getOpString(enum opCode code){
    return opStrings[code];
}

// passwordAsker.h

#ifndef PASSWORDASKER_H_INCLUDED
#define PASSWORDASKER_H_INCLUDED

int askPassword(char** passwd);
void disposePassword(char *password);

#endif // PASSWORDASKER_H_INCLUDED

// passwordAsker.c

#include <termios.h>
#include <string.h>
#include <signal.h>
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <errno.h>

#include "passwordAsker.h"
#include "logger.h"
```

```c
static struct termios oldTermConf;

int restoreEchoing(){
    if (tcsetattr(fileno(stdin), TCSANOW, &oldTermConf)){
        int err = errno;
        logMsg(E, "tctsetattr: %s\n", strerror(err));
        return 1;
    }
    return 0;
}

void handler(int sig){
    if (sig == SIGINT){
        if (restoreEchoing()){
            logMsg(E, "can't restore term\n");
            exit(EXIT_FAILURE);
        }
    exit(EXIT_SUCCESS);
    }
}

int disableEchoing(){
    struct termios newTermConf;
    memset(&newTermConf, 0, sizeof(struct termios));
    // save current terminal conf
    if (tcgetattr(fileno(stdin), &oldTermConf)){
        int err = errno;
        logMsg(E, "tcgetattr: %s\n", strerror(err));
        return 1;
    }
    // Sets an handler for SIGINT
    struct sigaction sa_sigint;
    memset(&sa_sigint, 0, sizeof(struct sigaction));
    sa_sigint.sa_handler = handler;
    if (sigaction(SIGINT, &sa_sigint, NULL) < 0){
        int err = errno;
        logMsg(E, "sigaction: %s\n", strerror(err));
        return 1;
    }
```

```
    // Sets terminal conf to obfuscate password
    memcpy(&newTermConf, &oldTermConf, sizeof(struct termios));
    newTermConf.c_lflag &= ~ECHO;
    if (tcsetattr(fileno(stdin), TCSANOW, &newTermConf)){
        int err = errno;
        logMsg(E, "tcsetattr: %s\n", strerror(err));
        return 1;
    }
    return 0;
}

int askPassword(char** passwd){

    // prompts user to type password
    logMsg(I, "Please enter password:\n");
    // sets term in non-echoing mode
    if (disableEchoing()){
        logMsg(E, "failed to set term in non-echoing mode\n");
        return 1;
    }
    // collects user password
    if(scanf("%ms[^;-#`$|\n]", passwd) != 1){
        int err = errno;
        logMsg(E, "scanf: %s\n", strerror(err));
        return 1;
    }
    int c;
    while((c = getchar()) != '\n' && c != EOF);
    // sets term in echoing mode
    if (restoreEchoing()){
        logMsg(E, "failed to set term in echoing mode\n");
        return 1;
    }
    return 0;
}

void disposePassword(char* password){
    // more actions could be taken to securely dispose of the password
    int passLen = strlen(password);
    memset(password, 0, passLen);
```

```
    // password must be disposed coerently to the collecting method used in askPassword (for istance, if scanf("%ms") is used function free() must be called)
    free(password);
}

// logger.h

#ifndef LOGGER_H_INCLUDED
#define LOGGER_H_INCLUDED

#define MSG_MAX_LEN 1024

enum Tag {I, W, E, D};

void logMsg(enum Tag tag, const char* msg, ...);

#endif // LOGGER_H_INCLUDED

// logger.c

#include <stdio.h>
#include <stdarg.h>
#include <string.h>
#include "logger.h"

static const char* tagStrings[] = {"I", "W", "E", "D"};

const char* getStringFromTag(enum Tag tag){
    return tagStrings[tag];
}

void logMsg(enum Tag tag, const char* format, ...){

    char buf[MSG_MAX_LEN];
    int firstHalfLen;
    va_list ap;
    va_start(ap, format);
```

```
    // Inserts tag on first half of log msg
    snprintf(buf, sizeof(char) * MSG_MAX_LEN, "[%s] ", getStringFromTag(tag));
    // concats format to log msg
    firstHalfLen = strlen(buf);
    vsnprintf(buf + firstHalfLen, MSG_MAX_LEN - firstHalfLen, format, ap);
    // Prints log msg on stdout
    fprintf(stdout, buf, getStringFromTag(tag), format);
    fflush(stdout);
    va_end(ap);
}
```