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

Basi di Dati e Conoscenza

Progetto A.A. 2019/2020

Sistema di gestione di una pizzeria

0244242

Matteo Conti

**Indice**

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

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

[3. Progettazione concettuale 9](#_Toc2081466291)

[4. Progettazione logica 14](#_Toc2147004904)

[5. Progettazione fisica 29](#_Toc518560220)

[Appendice: Implementazione 90](#_Toc403811585)

# Descrizione del Minimondo

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25 | Si vuole progettare il backend di un sistema informativo per la gestione dell’operatività di  una pizzeria. In tale pizzeria è di interesse tenere traccia dei tavoli disponibili ed assegnati,  dei camerieri associati ai tavoli, dei pizzaioli che preparano le pizze, del barista, del  manager. Ciascuno dei lavoratori della pizzeria ha differenti mansioni e può effettuare  operazioni differenti all’interno del sistema.  All’ingresso di un cliente, il manager lo riceve e lo registra, segnando nome, cognome e  numero di commensali, assegnando un tavolo disponibile in grado di ospitarli tutti.  Un cameriere ha sempre la possibilità di visualizzare quali tavoli a lui assegnati sono  occupati e quali sono stati serviti. Al momento di prendere l’ordine, il cameriere registra la  comanda. Parte delle ordinazioni sono espletate dal barista, parte dal pizzaiolo.  Barista e pizzaiolo hanno sempre la possibilità di visualizzare cosa debbono preparare, in  ordine di ricezione della comanda. Quando hanno preparato una bevanda o una pizza, il  cameriere può visualizzare cosa è pronto (in relazione agli ordini) e sapere cosa deve  consegnare a quale tavolo.  La pizzeria opera 24/7, ma per motivi di risparmio, in alcuni giorni sono disponibili un  numero differente di camerieri e vengono utilizzati un numero differente di tavoli. Il  manager può definire quali camerieri lavorano in quali turni e quali tavoli sono utilizzati in  quali turni. Il menu è unico per tutti i turni e definito dal manager, con i rispettivi prezzi.  Nel menu è necessario anche prevedere aggiunte per le pizze (ad esempio, un cliente  potrebbe voler aggiungere del tonno ad una pizza quattro formaggi), con i relativi costi.  Allo stesso modo, il manager ha la possibilità di tenere traccia delle disponibilità dei  singoli prodotti. In questo modo, se viene ordinato ad un cameriere da un cliente un  prodotto che non è disponibile, questo non potrà essere aggiunto all’ordine.    Il manager ha la possibilità di stampare lo scontrino di un ordine. Inoltre, per motivi  statistici, ha la possibilità di visualizzare le entrate giornaliere e/o mensili. |

# Analisi dei Requisiti

## Identificazione dei termini ambigui e correzioni possibili

|  |  |  |  |
| --- | --- | --- | --- |
| **Linea** | **Termine** | **Nuovo termine** | **Motivo correzione** |
| 1 | Assegnati | Occupati | È più chiaro in quanto nella specifica è presente il concetto di assegnazione di un tavolo ad un cameriere, mentre qui si intende l’assegnazione del tavolo al cliente. |
| 2 | Barista | Barista che prepara le bevande | Per completezza in quanto per il pizzaiolo viene specificato che esso prepara le pizze. |
| 7 | Tavolo disponibile | Tavolo libero | E’ più chiaro perché prima si fa riferimento al termine ‘occupato’ per i tavoli. |
| 8 | Assegnati | Associati | Nella riga 3 viene utilizzato il termine associato per indicare che un tavolo è associato ad un cameriere. |
| 15 | Giorni | Turni | Successivamente nella specifica ai camerieri ed ai tavoli è associato il concetto di turno e non di giorno. |
| 17-18 | Quali camerieri lavorano in quali turni e quali tavoli sono utilizzati in  quali turni… | I turni dei camerieri e quali tavoli sono utilizzati nei vari turni… | La nuova frase è più chiara. |
| 20 | Costi | Prezzi | In precedenza nella specifica è utilizzato il termine prezzi. |
| 22-23 | se viene ordinato ad un cameriere da un cliente un prodotto che non è disponibile… | Se un cliente ordina un prodotto che non è disponibile… | La nuova frase è più chiara. |

### Specifica disambiguata

|  |
| --- |
| Si vuole progettare il back end di un sistema informativo per la gestione dell’operatività di  una pizzeria. In tale pizzeria è di interesse tenere traccia dei tavoli disponibili e di quelli occupati,  dei camerieri associati ai tavoli, dei pizzaioli che preparano le pizze, del barista che prepara le bevande e del manager. Ciascuno dei lavoratori della pizzeria ha differenti mansioni  e può effettuare operazioni differenti all’interno del sistema.  All’ingresso di un cliente, il manager lo riceve e lo registra, segnando nome, cognome e  numero di commensali, assegnandogli un tavolo libero in grado di ospitarli tutti.  Un cameriere ha sempre la possibilità di visualizzare quali tavoli a lui associati sono  occupati e quali sono stati serviti. Al momento di prendere l’ordine il cameriere registra la  comanda. Parte delle ordinazioni sono espletate dal barista, parte dal pizzaiolo.  Barista e pizzaiolo hanno sempre la possibilità di visualizzare cosa debbono preparare, in  ordine di ricezione della comanda. Quando viene espletato un ordine, il  cameriere può visualizzare cosa è pronto (in relazione agli ordini) e a quale tavolo va consegnato.  La pizzeria opera 24/7, ma per motivi di risparmio, in alcuni turni sono disponibili  un numero differente di camerieri e vengono utilizzati un numero differente di tavoli. Il  manager può definire i turni dei camerieri e quali tavoli sono utilizzati nei vari turni.  Il menù è unico per tutti i turni e viene definito dal manager, con i rispettivi prezzi.  Nel menù è necessario anche prevedere eventuali aggiunte per le pizze  (ad esempio, un cliente potrebbe voler aggiungere del tonno ad una pizza quattro formaggi),  con i relativi prezzi.  Allo stesso modo, il manager ha la possibilità di tenere traccia delle disponibilità dei  singoli prodotti. In questo modo, se un cliente ordina un prodotto (pizza con eventuali aggiunte o una bevanda) che non è disponibile, questo non potrà essere aggiunto all’ordine.  Il manager ha la possibilità di stampare lo scontrino di un ordine. Inoltre, per motivi  statistici, ha la possibilità di visualizzare le entrate giornaliere e/o mensili. |

## Glossario dei Termini

|  |  |  |  |
| --- | --- | --- | --- |
| **Termine** | **Descrizione** | **Sinonimi** | **Collegamenti** |
| Tavolo | Il tavolo è la postazione che viene assegnata ad un cliente ed i suoi commensali dal manager, ogni tavolo è associato ad un cameriere, può essere occupato o libero. Non tutti i tavoli sono necessariamente utilizzati in tutti i turni di lavoro. |  | Cliente, Cameriere,  Turno, Ordine. |
| Impiegato | È una persona che lavora nella pizzeria, può essere un manager, un cameriere, un pizzaiolo oppure un barista. | Lavoratore. | Tavolo, Turno. |
| Cliente | Una persona che viene a mangiare nella pizzeria solo o con altre persone in particolare occupano un tavolo e fanno delle ordinazioni. |  | Tavolo. |
| Scontrino | Ricevuta fiscale degli ordini fatti da un tavolo. | Ricevuta. | Tavolo, Ordine |
| Ordine | Prodotto richiesto da un tavolo, può essere ordinato in una quantità maggiore di 1. | Comanda, Ordinazione. | Prodotto, Tavolo. |
| Prodotto | Sono gli alimenti che la pizzeria offre, possono essere una pizza, un ingrediente da aggiungere ad una pizza oppure una bevanda.  L’insieme dei prodotti definisce il menù. |  | Ordine. |
| Turno | Sono specifici intervalli di orari di lavoro (giorno/ora) nei quali può variare numero di camerieri, numero di tavoli e variare il singolo impiegato (e,g martedì lavora un barista, mercoledì ne lavora un altro). |  | Tavolo, Impiegato. |

## Raggruppamento dei requisiti in insiemi omogenei

|  |
| --- |
| **Frasi relative al Tavolo:** |
| In tale pizzeria è di interesse tenere traccia dei tavoli disponibili e di quelli occupati,  dei camerieri associati ai tavoli. |
| **Frasi relative all’ Impiegato:** |
| In tale pizzeria è di interesse tenere traccia dei tavoli disponibili e di quelli occupati, dei camerieri associati ai tavoli, dei pizzaioli che preparano le pizze, del barista che prepara le bevande e del manager. |
| Ciascuno dei lavoratori della pizzeria ha differenti mansioni e può effettuare operazioni differenti all’interno del sistema. |
| Un cameriere ha sempre la possibilità di visualizzare quali tavoli a lui associati sono occupati e quali sono stati serviti. |
| Al momento di prendere l’ordine il cameriere registra la comanda. |
| Barista e pizzaiolo hanno sempre la possibilità di visualizzare cosa debbono preparare, in ordine di ricezione della comanda |
| Il manager può definire i turni dei camerieri e quali tavoli sono utilizzati nei vari turni. |
| Allo stesso modo, il manager ha la possibilità di tenere traccia delle disponibilità dei singoli prodotti. |
| Il manager ha la possibilità di stampare lo scontrino di un ordine. Inoltre, per motivi statistici, ha la possibilità di visualizzare le entrate giornaliere e/o mensili. |
| **Frasi relative al Cliente**: |
| All’ingresso di un cliente, il manager lo riceve e lo registra, segnando nome, cognome e  numero di commensali, assegnandogli un tavolo disponibile in grado di ospitarli tutti. |
| **Frasi relative al Turno:** |
| La pizzeria opera 24/7, ma per motivi di risparmio, in alcuni turni sono disponibili  un numero differente di camerieri e vengono utilizzati un numero differente di tavoli. |
| **Frasi relative al Prodotto:** |
| Il menù è unico per tutti i turni e viene definito dal manager, con i rispettivi prezzi. |
| Nel menù è necessario anche prevedere eventuali aggiunte per le pizze (ad esempio, un cliente potrebbe voler aggiungere del tonno ad una pizza quattro formaggi), con i relativi prezzi. |
| **Frasi relative all’ Ordine:** |
| Parte delle ordinazioni sono espletate dal barista, parte dal pizzaiolo |
| Al momento di prendere l’ordine il cameriere registra la comanda. |
| In questo modo, se un cliente ordina un prodotto (pizza con eventuali aggiunte o una bevanda) che non è disponibile, questo non potrà essere aggiunto all’ordine. |
| **Frasi relative allo Scontrino:** |
| Il manager ha la possibilità di stampare lo scontrino di un ordine |

# Progettazione concettuale

## Costruzione dello schema E-R

Per lo sviluppo dello schema ER è stato seguito un approccio bottom-up partendo modellando dapprima i concetti basici che sono stati ricavati dall’analisi della specifica e poi le loro associazioni concettuali, di seguito sono riportati i passi seguiti (alcuni dei quali per evitare ridondanza già rappresentati in modo più specifico):

1. Per il cliente è importante sapere a quale tavolo viene assegnato in una prima fase è stato perciò messo un attributo che verrà in seguito reificato.

Immagine che contiene screenshot

Descrizione generata automaticamente

1. Come per il cliente anche qui il prodotto e il tavolo che sono strettamente legati all’ordine sono riportati come attributi che verranno poi reificati.



1. Per il turno si è scelto di mettere orario di inizio, orario di fine e giorno in quanto è prevedibile che i turni di diverse tipologie di impiegato inizino allo stesso orario ma finiscano in orari diversi (o viceversa) cosa che solo con giorno ed orario di inizio (o ora di fine) non sarebbe rappresentabile.



1. Per il concetto di prodotto si è scelto di specificarlo in 3 tipologie di prodotti che sono bevande, ingredienti aggiuntivi per le pizze per le pizze e pizze. Gli ultimi due formano quella che viene chiamata “Pizza+” che rappresenta una pizza con eventuali ingredienti aggiuntivi.

Immagine che contiene testo

Descrizione generata automaticamente

1. Per l’impiegato si è scelto di mettere una matricola piuttosto che il codice fiscale in quanto in genere in un luogo di lavoro ad un impiegato viene associata una matricola per l’identificazione. Per esplicitare concettualmente le tipologie di utenti che utilizzano l’applicazione si è scelto di mettere una generalizzazione totale che dividere l’impiegato in tutti i suoi possibili ruoli, il cameriere a differenza degli altri ha associati dei tavoli il cui attributo verrà successivamente reificato. Per i turni si è scelto di associare un turno a tutti gli impiegati a differenza di quanto riportato nella specifica in quanto è ragionevole pensare che non sia solo il cameriere ad aver associato un turno.

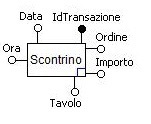


1. Al tavolo come riportato nella specifica è stato associato un codice (e.g. il numero del tavolo), il numero di persone che può ospitare (posti) ed un turno (in quanto nella specifica viene detto che un tavolo può non essere utilizzato sempre).

Immagine che contiene orologio, metro

Descrizione generata automaticamente

1. Nella specifica si parla di voler stampare uno scontrino, esso è utile anche ai fini di contabilità che è un altro elemento chiave della specifica per questo motivo è stato modellato come entità, si è scelto anche se “superfluo” di mettere un identificatore fittizio in quanto tipicamente gli scontrino hanno un codice di transazione, anche se in realtà potrebbe bastare anche solo il timestamp di data ed ora di emissione.



### Integrazione finale

Non è stato necessario risolvere conflitti. Per il tavolo si è scelto di associarlo a cameriere con una relazione con cardinalità (0,N) in modo da lasciare la libertà ai dipendenti di organizzarsi (e.g due camerieri hanno assegnato lo stesso tavolo uno prende gli ordini e l’altro porta le pietanze).

Immagine che contiene testo, mappa

Descrizione generata automaticamente

**Nota:** Il tool di disegno utilizzato non permette la rappresentazione di identificatori formati da un identificatore esterno più altri eventuali attributi di un’entità, perciò per le entità che si identificano in questo modo gli attributi che formano l’identificatore primario insieme a quello esterno sono rappresentati tramite pallino nero. La stessa cosa vale per le entità il cui identificatore è formato da più attributi (come per l’entità turno), in quanto non vi è a possibilità di utilizzare la notazione “barretta con pallino nero” che li unisce.

## Regole aziendali

**Vincoli:**

* Il prezzo di un prodotto non può essere negativo.
* Un prodotto con disponibilità 0 non può essere ordinato.
* Quando viene ordinato un prodotto la sua disponibilità viene ridotta di 1.

**Derivazioni:**

* L’importo dello scontrino deve essere pari alla somma dei prezzi di tutti i prodotti ordinati.
* Il prezzo di una pizza+ deve essere pari alla somma del prezzo della pizza usata come base e dei prezzi di tutti gli ingredienti aggiuntivi.
* Un tavolo è occupato se ha associato un cliente.

## Dizionario dei dati

|  |  |  |  |
| --- | --- | --- | --- |
| **Entità** | **Descrizione** | **Attributi** | **Identificatori** |
| Cliente | Il cliente è la persona che viene nella pizzeria insieme ad eventuali commensali ed occupa un tavolo. | Nome, Cognome, #Commensali, CF | CF |
| Tavolo | Il tavolo è una postazione messa a disposizione dal ristorante per i clienti. | #Tavolo, #Posti | #Tavolo |
| Turno | Il turno è uno slot temporale di una specifica giornata. | Ora Inizio, Ora Fine, Giorno | Ora Inizio, Ora Fine, Giorno |
| Impiegato | E’ il generico lavoratore della pizzeria può ricoprire 4 ruoli cioè cameriere, barman, pizzaiolo e manager | Nome, Cognome, Password, Matricola | Matricola |
| Ordine | E’ un prodotto ordinato dal cliente, può essere una pizza, una pizza con ingredienti aggiuntivi oppure una bevanda, ha inoltre uno stato di lavorazione. | Numero, Tavolo, StatoLavorazione | Numero, Tavolo |
| Prodotto | Il prodotto è un alimento o una bevanda che può essere ordinato nella pizzeria, in particolare gli alimenti sono pizze e ingredienti aggiuntivi. | Nome, Prezzo, #Scorte | Nome |
| Scontrino | E’ la ricevuta fiscale di un tavolo. | Data, Importo, IdTransazione | IdTransazione |
| Pizza+ | E’ una pizza con ingredienti aggiuntivi | Nome, Prezzo | Nome |

# Progettazione logica

## Volume dei dati

**Nota:**

Si è assunto che i tavoli siano da 2 posti (eventualmente possono essere uniti),che un cliente venga insieme ad altre 3 persone e che ognuno ordini una pizza ed una bevanda, che camerieri, pizzaioli e barman abbiano la giornata divisa in 4 turni da 6 ore mentre il manager abbia la giornata divisa in 3 turni da 8 ore e che i tavoli vengano associati contestualmente all’inizio di un turno e che le assegnazioni possano scambiarsi tra camerieri (in caso due tavoli vengano uniti in nuovo tavolo che sostituisce i precedenti due).Si assume inoltre che in 2 giorni della settimana ci siano 4 tavoli in meno e 2 camerieri in meno nei vari turni.

|  |  |  |
| --- | --- | --- |
| **Concetto nello schema** | **Tipo[[1]](#footnote-1)** | **Volume atteso** |
| Cliente | E | 7300 = 20clienti/gg\*365 |
| Occupa | R | 20 in quanto alla stampa dello scontrino il tavolo viene liberato. |
| Tavolo | E | 24 |
| Fatto Da | R | 58400 |
| Ordine | E | 58400=20clienti/gg\*4commensali\*2ordini/persona\*365gg |
| OrdineP | E | 25500 |
| OrdineB |  | 29200 |
| OrdineP+ |  | 3650 |
| OrdinataP | R | 25500=70ordiniPizza/gg\*365 |
| OrdinataB | R | 29200=80ordiniBevanda/gg\*365 |
| Pizza+ | E | 40 |
| OrdinataP+ | R | 3650=10ordiniPizza+/gg\*365 |
| Aggiunta | R | 10 |
| Base | R | 15 |
| Ingrediente | E | 30 |
| Pizza | E | 25 |
| Bevanda | E | 30 |
| Prodotto | E | 85 |
| Usato | R | 640=(24tavoli\*20turni)+(20tavoli\*8turni), i 20 turni sono 5 giorni della settimana in cui si usano tutti i tavoli i restanti 8 turni sono quelli dei 2 giorni in cui si usano solo 20 tavoli. |
| Associato | R | 18 in quanto si è assunto che i 24 tavoli da 2 posti all’inizio di ogni turno vengano divisi in 12 tavoli da 2 e 6 tavoli da 4 e che le associazioni dei camerieri ai tavoli siano fatte contestualmente all’inizio del turno di lavoro (eliminando le precedenti). |
| Turno | E | 49=(4 da 3 ore + 3 da 8 ore)\*7gg |
| Lavora | R | 96=(4camerieri\*20turni)+(2camerieri\*8turni) questo in quanto in ogni giorno regolare lavorano 4 camerieri per turno mentre negli altri 2 camerieri. |
| Impiegato | E | 35 |
| Cameriere | E | 16 |
| Pizzaiolo | E | 8 |
| Barman | E | 8 |
| Manager | E | 3 |
| Scontrino | E | 7300 |
| Appartiene | R | 7300 |
| Contiene | R | 58400 |

## Tavola delle operazioni

**Nota:**

Le frequenze sono pensate per una situazione a regime, ad esempio per i prodotti non si considera la prima volta nel quale vengono inseriti tutti quanti formando il menù. Molte delle operazioni non sono riportate nella specifica tuttavia sono necessarie al fine ad esempio per assegnare un tavolo ad un cliente è necessario prima visualizzare i tavoli.

|  |  |  |
| --- | --- | --- |
| **Cod.** | **Descrizione** | **Frequenza attesa** |
| 1 | Registra cliente (manager) | 20 volte al giorno. |
| 2 | Assegna tavolo a cliente (manager):  Contestualmente viene creato uno scontrino per il tavolo. | 20 volte al giorno. |
| 3 | Visualizza tavoli associati:  Visualizzazione dei tavoli associati ad un cameriere e del loro stato di occupazione e servizio (cioè si guarda se ci sono ordini in lavorazione a carico del tavolo).  (cameriere) | 768 volte al giorno, assumendo che un cameriere esegua questa operazione una volta ogni 7/8 minuti , che una giornata abbia 4 turni da 6 ore e che in ogni turno lavorino 4 camerieri.  8check/h\*6ore\*4turni\*4camerieri |
| 4 | Registra ordine bevanda  (cameriere)  Contestualmente avviene l’aggiunta allo scontrino e l’aumento del prezzo di quest’ultimo. | 80 volte al giorno, assumendo che ogni giorno ci siano 20 clienti in un gruppo di 4 persone le quali ognuna ordina una bevanda, causa la diminuzione della disponibilità della bevanda ordinata. |
| 5 | Registra ordine pizza  (cameriere)  Contestualmente avviene l’aggiunta allo scontrino e l’aumento del prezzo di quest’ultimo. | 70 volte al giorno, assumendo che ogni giorno ci siano 20 clienti in un gruppo di 4 persone le quali ognuna ordina una pizza, causa la riduzione della disponibilità della pizza ordinata. |
| 6 | Registra ordine pizza+  (cameriere)  Si assume che contestualmente venga creata la pizza+, si da la possibilità di aggiungere al massimo 5 ingredienti.  Contestualmente avviene l’aggiunta allo scontrino e l’aumento del prezzo di quest’ultimo. | 10 volte al giorno, assumendo che ogni giorno ci siano 10 persone che ordinano una pizza con ingredienti aggiuntivi, causa la riduzione della disponibilità della pizza usata come base e degli ingredienti aggiunti. |
| 7 | Visualizza ordini pizza  (pizzaiolo) | 576 volte al giorno, 12check/h\*6ore\*4turni\*2pizzaioli |
| 8 | Visualizza ordini bevande (barman) | 576 volte al giorno, con le stesse assunzioni dell’operazione 7. |
| 9 | Espleta ordine bevanda (barman) | 80 volte al giorno, dato che si è assunto vengano ordinate 80 bevande. |
| 10 | Visualizza ordini espletati (cameriere) | 1152 volte al giorno, stesse assunzioni dell’operazione 8, solo che i camerieri sono 4 per turno a differenza dei pizzaioli che sono 2 . |
| 11 | Consegna ordine (cameriere): | 160 volte al giorno. |
| 12 | Inserisci turno (manager) | 49 volte ogni 4 mesi. Si assume che i turni vengano cambiati completamente stagionalmente e che eventuali modifiche ne modifichi l’orario piuttosto che crearne di nuovi, tranne in occasioni di cambio totale dei turni che si suppone avvenire raramente. |
| 13 | Rimuovi turno (manager) | 49 volte ogni 4 mesi, stesse assunzioni operazione 12. |
| 14 | Assegna turno a impiegato (manager) | 277 volte ogni 4 mesi, basandosi sulla tavola dei volumi in quanto ho 4 turni da 6 ore e 3 turni da 8 ore in una giornata, ogni turno da 6 ore deve essere assegnato a 4 camerieri, 2 pizzaioli e 2 barman, ogni turno da 8 ore deve essere assegnato ad un manager. Si sta assumendo che i turni assegnati possano cambiare ogni mese e che una volta a settimana un impiegato chieda al manager di cambiare turno (cosa che impatta sul suo turno e su quello di un altro impiegato che dovrà essere messo al suo posto).  ((4+2+2)\*4+3)\*7+24 |
| 15 | Rimuovi turno a impiegato (manager) | 277 volte al mese, con le stesse assunzioni dell’operazione 14. |
| 16 | Aggiungi tavolo (manager) | 1 volta ogni 6 mesi. |
| 17 | Rimuovi tavolo (manager) | 1 volta ogni 6 mesi |
| 18 | Assegna turno a tavolo (manager) | 672 volte ogni 4 mesi, in quanto i tavoli si assume siano soggetti solo a 4 turni da 6 ore e che si scelgano stagionalmente i turni dei tavoli. |
| 19 | Rimuovi turno a tavolo (manager) | 672 volte ogni 4 mesi, con le stesse assunzioni dell’operazione 18. |
| 20 | Aggiungi impiegato (manager) | Una volta ogni 6 mesi. |
| 21 | Rimuovi impiegato (manager) | Una volta ogni 6 mesi. |
| 22 | Visualizza menù pizze (cameriere, manager) | 26 volte al giorno, assumendo che un cameriere la esegua una volta per ogni volta che va a prendere un ordine ad un tavolo e che un manager lo faccia due volte nel suo turno. |
| 23 | Visualizza menù ingredienti aggiuntivi  (cameriere, manager) | 10 volte al giorno, dato che si assume vengono ordinate 10 pizze+ |
| 24 | Visualizza menù bevande (cameriere, manager) | 20 volte con le stesse assunzioni dell’operazione 22. |
| 25 | Visualizza disponibilità prodotti (manager) | 6 volte al giorno, assumendo che venga fatto due volte per turno dai 3 manager e che i manager lavorino in 3 turni da 8 ore. |
| 26 | Aggiungi prodotto (manager) | 1 volta ogni 4 mesi. |
| 27 | Rimuovi prodotto (manager):  Vengono eliminati anche i relativi ordini in quanto diverrebbero inconsistenti non essendo più riconducibili ai prodotti ordinati. | 1 volta ogni 4 mesi. |
| 28 | Aumenta scorte prodotto (manager) | 90 volte al giorno, 85 avvengono alla mattina quando viene fatta la spesa ricaricando la disponibilità di tutti i prodotti e 5 sono le eventuali volte che un prodotto finisce in giornata e viene ricomprato subito. |
| 29 | Stampa scontrino tavolo (manager):  Viene aggiunta la data di emissione allo scontrino e successivamente viene mostrata insieme alla lista degli ordini ed il totale dovuto. | 20 volte al giorno. |
| 30 | Visualizza entrate giorno (manager) | 1 volta al giorno. |
| 31 | Visualizza entrate mese (manager) | 2 volte al mese, si assume che venga visualizzato il mese corrente e quello passato. |
| 32 | Visualizza turni impiegati (manager) | 3 volte al giorno, una per manager. |
| 33 | Visualizza turni tavoli (manager) | 3 volte al giorno, una per manager. |
| 34 | Assegna tavolo a cameriere (manager) | 80 volte al giorno, assumendo che ad inizio giornata tutti i tavoli vengano assegnati ai camerieri in particolare, 12 tavoli vengono lasciati singoli e i restanti 12 formano 6 tavoli da 4 persone, nel resto del turno si assume che i clienti vengano tutti in gruppi da 4 perciò sarà necessario unire i 12 tavoli singoli i quali sono assegnati ai 4 camerieri nel turno (3 per cameriere) e si assume che in media si possa ottimizzare in modo da evitare riassegnazioni unendo 2 dei 3 tavoli di ogni cameriere. Per i restanti 4 tavoli verrà tolta l’assegnazione a 2 camerieri e data l’assegnazione agli altri 2. Sarà necessario poi riassegnare tutti i tavoli ai quattro camerieri dei tre turni successivo ogni volta assumendo che venga lasciata l’organizzazione in tavoli da 4.  12assTavoliSingoli+6assTavoliDoppi+2riassegnazioni\*4turni |
| 35 | Rimuovi assegnazione tavolo (manager) | 96 volte al giorno, in quanto per i stessi motivi dell’operazione precedente si fanno 24 rimozioni per turno (tranne il primo) e a fine giornata si rimuovono tutte e 24 assegnazioni.  24\*4 |
| 36 | Visualizza tavoli (manager) | 70 volte al giorno, 20 per registrare i clienti, 20 per stampare lo scontrino, 30 per controllare la disponibilità prima di registrare un cliente assumendo che 20 poi effettivamente vengono accolti e gli altri 10 no in quanto il ristorante è pieno. |
| 37 | Visualizza assegnazioni tavoli a camerieri (manager) | 24 volte al giorno, 3 per turno. |
| 38 | Espleta ordine pizza (pizzaiolo) | 70 volte al giorno. |
| 39 | Espleta ordine pizza+ (pizzaiolo) | 10 volte al giorno. |
| 40 | Prendi in carico ordine pizza (pizzaiolo) | 70 volte al giorno. |
| 41 | Prendi in carico ordine pizza+ (pizzaiolo) | 10 volte al giorno. |
| 42 | Prendi in carico ordine bevanda (barman) | 80 volte al giorno. |
| 43 | Login | 23 volte al giorno, una per impiegato. |
| 44 | Visualizza ordini pizza+  (pizzaiolo) | Stesse assunzioni dell’operazione 7. |

## Costo delle operazioni

|  |  |  |  |
| --- | --- | --- | --- |
| **Op.** | **Accessi** | **Tipo** | **Costo** |
| 1 | 1 accesso a Cliente | S | 2\*20=40 accessi al giorno. |
| 2 | 1 accesso a Tavolo  1 accesso a Cliente  1 accesso a Scontrino  1 accesso a Tavolo | S  L  S  L | (2+1+2+1)\*20=120 accessi al giorno. |
| 4 | 1 accesso a OrdineB  1 accesso a Tavolo  1 accesso a Bevanda  1 accesso a Bevanda  1 accesso a Scontrino  1 accesso a Scontrino  Gli accessi a bevanda sono in scrittura ed in quanto la prima è relativa alla diminuzione della disponibilità della specifica Bevanda la seconda per la verifica del vincolo dell’ associazione con Ordine.(Sarà cosi anche per le pizze e gli ingredienti), stesso discorso avviene per lo scontrino. | S  L  S  L  L  S | (2+1+2+1+1+2)\*80=720 accessi al giorno. |
| 5 | 1 accesso a OrdineP  1 accesso a Tavolo  1 accesso a Pizza  1 accesso a Pizza  1 accesso a Scontrino  1 accesso a Scontrino | S  L  L  S  L  S | (2+1+1+2+1+2)\*70=630 accessi al giorno. |
| 6 | 1 accesso a OrdineP+  1 accesso a Tavolo  1 accesso a Pizza+  1 accesso a Pizza  1 accesso a Ingrediente  1 accessi ad Aggiunta  1 accesso a Scontrino  1 accesso a Scontrino | S  L  S  S  S  S  S  L | (2+1+2+2+2+2+1+2)\*10=140 accessi al giorno, assumendo che si aggiunga un solo ingrediente. |
| 9 | 1 accesso a Ordine | S | 2\*160=320 accessi al giorno. |
| 11 | 1 accesso a Ordine | S | 2\*160=320 accessi al giorno. |
| 12 | 1 accesso a Turno | S | 2\*49=98 accessi ogni 4 mesi. |
| 13 | 1 accesso a Turno  22 accessi a Usato (media tra i giorni che ne uso 20 e quelli che ne uso 24)  3 accessi a Lavora | S  S    S | 2\*(1+22+3)\*49=2548 accessi ogni 4 mesi. |
| 14 | 1 accesso a Lavora  1 accesso a Impiegato  1 accesso a Turno | S  L  L | (2+1+1)\*277=1108 accessi ogni 4 mesi. |
| 15 | 1 accesso a Lavora | S | 2\*277=554 accessi ogni 4 mesi. |
| 16 | 1 accesso a Tavolo | S | 1 accesso ogni 6 mesi. |
| 17 | Si assume che si rimuova un tavolo libero.  1 accesso a Tavolo  1 accesso a Associato  32 accesso a Usato (assumendo di prendere un tavolo che lavora ogni giorno)  1217 accesso a FattoDa (volume di FattoDa/(2\*24) per prendere il volume semestrale e di un generico tavolo) | S  S  S  S  S | 2\*(1+1+32+1217+1)=2504 accessi ogni 6 mesi. |
| 18 | 1 accesso a Usato  1 accesso a Turno  1 accesso a Tavolo | S  L  L | (2+1+1)\*672=2688 accessi ogni 4 mesi. |
| 19 | 1 accesso a Usato | S | 2\*672=1344 accessi ogni 4 mesi. |
| 20 | 1 accesso a Impiegato | S | 1 accesso ogni 6 mesi. |
| 21 | 1 accesso a Impiegato  7 accessi a Lavora | S  S | (1+7)\*2=16 accessi ogni 6 mesi .  Si sta assumendo che in caso si rimuova un cameriere esso non sia attualmente in turno, e che quindi non sia associato a tavoli. |
| 26 | 1 accesso a Prodotto | S | 1 accesso ogni 4 mesi |
| 27 | Quando si rimuove un prodotto esso può essere una pizza, una bevanda o un ingrediente aggiuntivo per questo i tre casi hanno costi diversi che verranno pesati con la probabilità che sia un certo prodotto (per semplicità uniforme).  Si è partiti dai volumi di OrdinataP/B/P+ si è diviso per 4 per ottenere i volumi quadrimestrali e poi per il numero di Pizze/Bevande/Ingrediente per trovare gli ordini medi della singola pizza/bevanda/ingrediente. Per le pizze e gli ingredienti si assume che una pizza venga usata una volta come base e l’ingrediente 2 volte come aggiunta.  Se viene rimossa una bevanda:  1 accesso a Bevanda  243 accessi a OrdineB  Se viene rimosso un ingrediente:  1 accesso a Ingrediente  1 accesso a Aggiunta  1 accesso a Pizza+  121 accessi a OrdineP+  Se viene rimossa una pizza:  1 accesso a Pizza  255 accessi a OrdineP  121 accessi a OrdineP+  1 accesso a Pizza+  1 accesso a Aggiunta | S  S    S  S  S  S    S  S  S  S  S | 1/3\*(1+243)\*2+  +1/3\*(1+1+1+121)\*2+1/3\*(1+255+  +121+1+1)\*2= 465 accessi ogni 4 mesi. |
| 28 | 1 accesso a Prodotto  (nella frequenza si è assunto che venga fatta singolarmente per ogni prodotto, eventualmente se viene fatta una volta sola essa accede a tutti i prodotti e quindi il costo in accessi non cambia) | S | 2\*90=180 accessi al giorno. |
| 29 | Si è assunto di considerare il caso in cui siano ordinate 4 bevande diverse, 3 pizze diverse e una pizza+  1 accesso a Scontrino  8 accessi a Ordine  3 accessi a Pizza  4 accessi a Bevanda  1 accesso a Pizza+ | S  L  L  L  L | (2+8+3+4+1)\*20=360 accessi al giorno. |
| 34 | 1 accesso ad Associato  1 accesso a Tavolo  1 accesso a Cameriere | S  L  L | (2+1+1)\*80=320 accessi al giorno. |
| 35 | 1 accesso a Associato | S | 2\*96=192 accessi al giorno. |
| 38 | 1 accesso OrdineP | S | 2\*70=140 accessi al giorno. |
| 39 | 1 accesso a OrdineP+ | S | 2\*10=20 accessi al giorno. |
| 40 | 1 accesso a OrdineP | S | 2\*70=140 accessi al giorno. |
| 41 | 1 accesso a OrdineP+ | S | 2\*10=20 accessi al giorno. |
| 42 | 1 accesso a OrdineBevanda | S | 2\*80=160 accessi al giorno. |

## Ristrutturazione dello schema E-R

**Analisi delle ridondanze**

1. Nell’entità “Scontrino” è presente l’attributo “Importo” il quale rappresenta l’importo totale calcolato come la somma del prezzo di tutti i prodotti ordinati da un certo tavolo, questo è derivabile percorrendo a ritroso il percorso che da scontrino porta a prodotto, questa operazione viene già fatta in corso di stampa dello scontrino (ed è necessaria), tuttavia per le operazioni che riguardano la contabilità introdurrebbe un notevole numero di accessi ogni volta che si vuole guardare un entrata, in particolare :

* Per la visualizzazione delle entrate giornaliere:
* 35 accesso a Scontrino di tipo L
* 280 accessi a Contiene di tipo L
* 280 accessi a Ordine di tipo L
* 140 accessi a Bevanda di tipo L
* 35 accesso a Pizza+ di tipo L
* 105 accessi a Pizza di tipo L
* Per la visualizzazione delle entrate mensili (si è considerato un mese di 31 giorni) :
* 620 accesso a Scontrino di tipo L
* 4960 accessi a Contiene di tipo L
* 4960 accessi a Ordine di tipo L
* 2480 accessi a Bevanda di tipo L
* 620 accesso a Pizza+ di tipo L
* 1860 accessi a Pizza di tipo L

Lasciando invece l’attributo “Importo” occorrono molti meno accessi, in particolare :

* Per la visualizzazione delle entrate giornaliere:
* 1 accesso a scontrino
* Per la visualizzazione delle entrate mensili:
* 31 accessi a scontrino

Il prezzo da pagare per questi accessi in meno sono 4\*7300=29200Kbyte (volume annuale, ipotizzando di utilizzare un float per l’importo) che sono trascurabili rispetto agli accessi, inoltre aggiungere importo come attributo aumenta al solidità della consistenza degli scontrini in quanto ogni volta che viene registrato un ordine viene subito aggiornato l’importo dello scontrino in modo che se un prodotto viene eliminato e quindi non si ha più l’informazione sul prezzo lo scontrino tiene comunque tenuto conto nel prezzo del fatto che quel prodotto è stato ordinato.

1. Nell’entità “Pizza+” vi è l’attributo prezzo che è ridondante in quanto come riportato anche nei vincoli esso è pari alla somma del prezzo della pizza usata come base e dei prezzi di tutti gli ingredienti aggiuntivi metterlo però riduce gli accessi in fase di stampa dello scontrino in particolare:

* Togliendo prezzo da Pizza+ gli accessi sono i seguenti in caso di presenza di un ordine di Pizza+ negli ordini (dove k dipende indica il numero di ingredienti aggiuntivi) :
  + 1 accesso a Scontrino
  + 1 accessi a Ordine
  + 1 accesso a Pizza+
  + 1 accesso a Pizza
  + k accessi Aggiunta
  + k accessi a Ingrediente
* Lasciando prezzo a Pizza+ gli accessi sono i seguenti:
  + 1 accesso a Scontrino
  + 1 accessi a Ordine
  + 1 accesso a Pizza+

C’è quindi una riduzione di 2k+1 accessi al costo di 8byte per ogni istanza di Pizza+ che sono pochissimi.

**Eliminazione delle generalizzazioni**

Per la generalizzazione sull’entità “Impiegato” si è scelto di eliminarla e mettere un attributo ruolo a impiegato, per la generalizzazione sull’entità, per “Prodotto” e “Ordine” si è scelto di togliere il padre e mantenere le figlie in quanto nella maggior parte delle operazioni si accede alle figlie separatamente, in particolare le operazioni più frequenti nella giornata su queste entità sono quelle di camerieri, pizzaioli e barman (visualizzare ordini, pizze ecc) mentre le meno frequenti del manager (inventario).

**Scelta identificatori primari**

|  |  |
| --- | --- |
| **Entità** | **Identificatore primario** |
| Cliente | CF |
| Pizza | Nome |
| Bevanda | Nome |
| Ingrediente | Nome |
| Pizza+ | Nome |
| OrdineB | Numero, #Tavolo (identificatore esterno) |
| OrdineP+ | Numero, #Tavolo (identificatore esterno) |
| OrdineP | Numero, #Tavolo (identificatore esterno) |
| Turno | Ora Inizio, Ora Fine, Giorno |
| Impiegato | Matricola |
| Tavolo | #Tavolo |
| Scontrino | IdTransazione |

## Trasformazione di attributi e identificatori

Nelle entità deboli OrdineP, OrdineP+, OrdineB l’identificatore esterno “#Tavolo” viene ridenominato Tavolo per rendere più chiaro da quale entità è preso.

## Traduzione di entità e associazioni

**Nota:**

Gli attributi con vincolo di integrità referenziale sono riportati in corsivo.

Cliente(CF, Nome, Cognome, Commensali)

Tavolo(Numero, Posti, *Cliente*)

Usato(*Tavolo, TurnoOraInizio, TurnoOraFine, TurnoGiorno*);

Associato(*Tavolo, Cameriere*);

Turno(OraInizio, OraFine, Giorno)

Lavora(*TurnoOraInizio, TurnoOraFine, TurnoGiorno, Impiegato*)

Impiegato(Matricola¸ Nome, Cognome, Password, Ruolo)

Scontrino(IdTransazione, Emissione, Importo, *Tavolo*) , in questo caso data ed ora di emissioni sono state sostituite con un unico timestamp comprendente entrambe.

OrdineB(Numero, StatoLavorazione, *Bevanda, Scontrino, Tavolo*)

OrdineP(Numero, StatoLavorazione, *Pizza, Scontrino, Tavolo*)

OrdinePizzaPlus (Numero, StatoLavorazione, *PizzaPlus, Scontrino, Tavolo*)

Pizza(Nome, Prezzo, #Scorte)

Ingrediente(Nome, Prezzo, #Scorte)

Bevanda(Nome, Prezzo, #Scorte)

PizzaPlus (Nome, Prezzo, *Pizza*)

Aggiunta(*Ingrediente, PizzaPlus)*

**Vincoli di integrità referenziale:**

Usato(Tavolo) ⊆ Tavolo(Numero)

Usato(TurnoOraInizio, TurnoOraFine, TurnoGiorno) ⊆ Turno(OraInizio, OraFine, Giorno)

Associato(Tavolo) ⊆ Tavolo(Numero)

Associato(Cameriere) ⊆ Impiegato(Matricola)

Lavora(TurnoOraInizio, TurnoOraFine, TurnoGiorno) ⊆ Turno(OraInizio, OraFine, Giorno)

Lavora(Impiegato) ⊆ Impiegato(Matricola)

Scontrino(Tavolo) ⊆ Tavolo(Numero)

OrdineB(Tavolo) ⊆ Tavolo(Numero)

OrdineP(Tavolo) ⊆ Tavolo(Numero)

OrdinePizzaPlus(Tavolo) ⊆ Tavolo(Numero)

OrdineB(Scontrino) ⊆ Scontrino(Data, Ora)

OrdineP(Scontrino) ⊆ Scontrino(IdTransazione)

OrdinePizzaPlus(Scontrino) ⊆ Scontrino(IdTransazione)

OrdineB(Bevanda) ⊆ Bevanda(Nome)

OrdineP(Pizza) ⊆ Pizza(Nome)

OrdinePizzaPlus(PizzaPlus) ⊆ PizzaPlus(Nome)

PizzaPlus(Pizza) ⊆ Pizza(Nome)

Aggiunta(PizzaPlus) ⊆ PizzaPlus(Nome)

Aggiunta(Ingrediente) ⊆ Ingrediente(Nome)

Immagine che contiene testo, mappa

Descrizione generata automaticamente

## Normalizzazione del modello relazionale

Le relazioni sono tutte in 3NF.

# Progettazione fisica

## Utenti e privilegi

* Cameriere

Il cameriere ha necessita di vedere i tavoli a cui è assegnato per questo ha necessita di accedere ad associato, tavolo e impiegato in lettura, inoltre ha la necessita di registrare gli ordini presi dai clienti dei tavoli a lui assegnati in particolare per come è stata pensato il sistema, esso vede la lista dei vari prodotti e poi li seleziona necessitando perciò i privilegi di lettura dei vari prodotti, ovviamente quando un prodotto viene ordinato la sua disponibilità diminuisce perciò ha bisogno anche dei privilegi di modifica sui prodotti, infine per segnare gli ordini come consegnati ha bisogno anche del privilegio di modifica dell’ordine.

* + Tabella “Associato” lettura
  + Tabella “Impiegato” lettura
  + Tabella “Tavolo” lettura
  + Tabella “OrdineB” lettura, inserimento e modifica
  + Tabella “OrdineP” lettura, inserimento e modifica
  + Tabella “OrdinePizzaPlus” lettura, inserimento e modifica
  + Tabella “PizzaPlus” inserimento e modifica
  + Tabella “Aggiunta” inserimento e modifica
  + Tabella “Pizza” lettura
  + Tabella “Ingrediente” lettura
  + Tabella “Bevanda” lettura
* Pizzaiolo

Il pizzaiolo necessita di vedere quali ordini di pizza e PizzaPlus deve espletare e successivamente deve segnarli come espletati per segnalare al cameriere che sono pronti, per questo motivo ha i privilegi a OrdineP e OrdinePizzaPlus in lettura e modifica.

* + Tabella “OrdineP” lettura, modifica
  + Tabella “OrdinePizzaPlus” lettura, modifica
* Barman

Per gli stessi motivi del pizzaiolo il barman ha i privilegi di lettura e modifica su OrdineB.

* + Tabella “OrdineB” lettura, modifica
* Manager

Il manager ha bisogno di privilegi in tutte le tabelle tuttavia non ha bisogno dei privilegi totali su ogni tabella (per come sono state progettate le operazioni), in particolare,

ha bisogno di inserire clienti quando li accoglie e di rimuoverli quando vanno via per questo ha i privilegi in inserimento e cancellazione su quest’entità poi inoltre deve poter associare i clienti ai tavoli perciò necessita di modificare la tabella tavolo, inoltre deve poter aggiungere e rimuovere tavoli ed associarli a dei turni e a dei camerieri per questo ha i privilegi in lettura, inserimento e cancellazione a tavolo, usato e associato, i turni vanno poi definiti ed associati agli impiegati per questo ha dei privilegi per inserire e rimuovere turni e associarli agli impiegati tramite la tabella lavora, gli impiegati anch’essi vanno registrati all’assunzione e cancellati a fine rapporto di lavoro. Le restanti mansioni del manager sono la stampa dello scontrino motivo per il quale ha i privilegi di inserimento, modifica e cancellazione alla tabella scontrino e di lettura degli ordini e dei prodotti, e l’inserimento e la rimozione di prodotti e l’approvvigionamento di questi ultimi per i quali ha appunto i privilegi di inserimento, modifica e cancellazione.

* + Tabella “Cliente” inserimento e cancellazione
  + Tabella “Tavolo” lettura, inserimento, modifica e cancellazione
  + Tabella “Turno” lettura, inserimento e cancellazione
  + Tabella “Usato” lettura, inserimento e cancellazione
  + Tabella “Associato” lettura, inserimento e cancellazione
  + Tabella “Lavora” lettura, inserimento e cancellazione
  + Tabella “Impiegato” lettura, inserimento e cancellazione
  + Tabella “Pizza” lettura, inserimento, modifica e cancellazione
  + Tabella “Bevanda” lettura, inserimento, modifica e cancellazione
  + Tabella “Ingrediente” lettura, inserimento, modifica e cancellazione
  + Tabella “Scontrino” lettura, inserimento, modifica e cancellazione
  + Tabella “OrdineP” lettura
  + Tabella “OrdinePizzaPlus” lettura
  + Tabella “OrdineB” lettura
  + Tabella “PizzaPizzaPlus” lettura

**Nota:**

Tutti i privilegi sopra riportati si mapperanno su privilegi di esecuzione di apposite store procedure che a questo punto della progettazione non sono ancora state definite.

## Strutture di memorizzazione

**Nota:**

Nelle entità “Pizza”, “Bevanda” ed “Ingrediente” gli attributi “Scorte” e “Prezzo” sono UN questo va a soddisfare le regole aziendali che impongo che il prezzo non possa essere negativo e che un prodotto non può essere aggiunto ad un prodotto se è terminato.

|  |  |  |
| --- | --- | --- |
| **Tabella Bevanda** | | |
| **Attributo** | **Tipo di dato** | **Attributi[[2]](#footnote-2)** |
| Nome | VARCHAR(45) | PK, NN |
| Scorte | INT | NN, UN |
| Prezzo | FLOAT | NN, UN |
| **Tabella Pizza** | | |
| **Attributo** | **Tipo di dato** | **Attributi** |
| Nome | VARCHAR(45) | PK, NN |
| Scorte | INT | NN, UN |
| Prezzo | FLOAT | NN, UN |
| **Tabella Ingrediente** | | |
| **Attributo** | **Tipo di dato** | **Attributi** |
| Nome | VARCHAR(45) | PK, NN |
| Scorte | INT | NN, UN |
| Prezzo | FLOAT | NN, UN |
| **Tabella OrdinePizza** | | |
| **Attributo** | **Tipo di dato** | **Attributi** |
| Numero | BIGINT | PK, NN, UN, AI |
| Tavolo | INT | PK, NN, UN, FK |
| Pizza | VARCHAR(45) | FK |
| StatoLavorazione | ENUM(‘In carico’, 'In lavorazione', 'Espletato', 'Consegnato') | NN |
| Scontrino | BIGINT | UN, FK |
| **Tabella OrdinePizzaPlus** | | |
| **Attributo** | **Tipo di dato** | **Attributi** |
| Numero | BIGINT | PK, NN, UN, AI |
| Tavolo | INT | PK, NN, UN, FK |
| PizzaPlus | VARCHAR(275) (45\*5ingredienti aggiuntivi + 5 caratteri per rendere il nome piu leggibile) | FK |
| StatoLavorazione | ENUM((‘In carico’,'In lavorazione', 'Espletato', 'Consegnato') | NN |
| Scontrino | BIGINT | UN, FK |
| **Tabella OrdineBevanda** | | |
| **Attributo** | **Tipo di dato** | **Attributi** |
| Numero | BIGINT | PK, NN, UN, AI |
| Tavolo | INT | PK, NN, UN, FK |
| Bevanda | VARCHAR(45) | FK |
| StatoLavorazione | ENUM((‘In carico’,'In lavorazione', 'Espletato', 'Consegnato') | NN |
| Scontrino | BIGINT | UN, FK |
| **Tabella PizzaPlus** | | |
| **Attributo** | **Tipo di dato** | **Attributi** |
| Nome | VARCHAR(275) | PK, NN |
| Prezzo | FLOAT | NN, UN |
| Base | VARCHAR(45) | NN, FK |
| **Tabella Aggiunta** | | |
| **Attributo** | **Tipo di dato** | **Attributi** |
| PizzaPlus | VARCHAR(275) | PK, NN, FK |
| Ingrediente | VARCHAR(45) | PK, NN, FK |
| **Tabella Turno** | | |
| **Attributo** | **Tipo di dato** | **Attributi** |
| OraInizio | TIME | PK, NN |
| OraFine | TIME | PK, NN |
| Giorno | ENUM('Domenica', 'Lunedi', 'Martedi', 'Mercoledi', 'Giovedi', 'Venerdi', 'Sabato') | PK, NN |
| **Tabella Lavora** | | |
| **Attributo** | **Tipo di dato** | **Attributi** |
| Impiegato | INT | PK, NN, UN |
| TurnoOraInizio | TIME | PK, NN, FK |
| TurnoOraFine | TIME | PK, NN, FK |
| TurnoGiorno | ENUM('Domenica', 'Lunedi', 'Martedi', 'Mercoledi', 'Giovedi', 'Venerdi', 'Sabato') | PK, NN, FK |
| **Tabella Usato** | | |
| **Attributo** | **Tipo di dato** | **Attributi** |
| Tavolo | INT | PK, NN, UN |
| TurnoOraInizio | TIME | PK, NN, FK |
| TurnoOraFine | TIME | PK, NN, FK |
| TurnoGiorno | ENUM('Domenica', 'Lunedi', 'Martedi', 'Mercoledi', 'Giovedi', 'Venerdi', 'Sabato') | PK, NN, FK |
| **Tabella Associato** | | |
| **Attributo** | **Tipo di dato** | **Attributi** |
| Tavolo | INT | PK, NN, UN, FK |
| Impiegato | INT | PK, NN, UN, FK |
| **Tabella Impiegato** | | |
| **Attributo** | **Tipo di dato** | **Attributi** |
| Matricola | INT | PK, NN, UN |
| Nome | VARCHAR(45) | NN |
| Cognome | VARCHAR(45) | NN |
| Password | VARCHAR(45) | NN |
| Ruolo | ENUM('Manager', 'Cameriere', 'Pizzaiolo', 'Barman') | NN |
| **Tabella Tavolo** | | |
| **Attributo** | **Tipo di dato** | **Attributi** |
| Numero | INT | PK, NN, UN |
| Posti | INT | NN, UN |
| Cliente | VARCHAR(16) | FK |
| **Tabella Scontrino** | | |
| **Attributo** | **Tipo di dato** | **Attributi** |
| IdTransazione | BIGINT | PK, NN, UN, AI |
| Emissione | DATETIME | NN |
| Importo | FLOAT | NN, UN |
| **Tabella Cliente** | | |
| **Attributo** | **Tipo di dato** | **Attributi** |
| CF | VARCHAR(16) | PK, NN |
| Nome | VARCHAR(45) | NN |
| Cognome | VARCHAR(45) | NN |
| Commensali | INT | NN, UN |

## Indici

**Nota:**

La maggior parte degli indici è creata automaticamente in quanto la loro presenza è necessaria per vincoli di foreign key o per la presenza di chiavi primarie, in quelli inseriti per scelta progettuale è riportata la motivazione di tali scelte.

|  |  |
| --- | --- |
| **Tabella Pizza** | |
| **Indice PRIMARY** | **Tipo:** |
| Nome | PR |
| **Tabella Bevanda** | |
| **Indice PRIMARY** | **Tipo:** |
| Nome | PR |
| **Tabella Ingrediente** | |
| **Indice PRIMARY** | **Tipo:** |
| Nome | PR |
| **Tabella Aggiunta** | |
| **Indice PRIMARY** | **Tipo:** |
| PizzaPlus, Ingrediente | **PR** |
| **Indice Aggiunta\_FK\_Ingrediente\_idx** | **Tipo:** |
| Ingrediente | IDX |
| **Tabella PizzaPlus** | |
| **Indice PRIMARY** | **Tipo:** |
| Nome | PR |
| **Indice PizzaPlus\_FK\_Pizza\_idx** | **Tipo:** |
| Base | IDX |
| **Tabella OrdinePizza** | |
| **Indice PRIMARY** | **Tipo:** |
| Numero, Tavolo | PR |
| **Indice OrdinePizza\_FK\_Tavolo\_idx** | **Tipo:** |
| Tavolo | IDX |
| **Indice OrdinePizza\_FK\_Pizza\_idx** | **Tipo:** |
| Pizza | IDX |
| **Indice OrdinePizza\_FK\_Scontrino\_idx** | **Tipo:** |
| Scontrino | IDX |
| **Indice OrdinePizza\_StatoLavorazione\_idx** | **Tipo:** |
| StatoLavorazione  I pizzaioli ed i camerieri effettuano periodicamente un’ operazione di controllo degli ordini, in particolare gli interessano solo gli ordini che hanno un certo stato di lavorazione, al pizzaiolo interessano solo quelli che devono essere ancora espletati mentre il cameriere interessano solo quelli che sono stati espletati, mettendo un indice sull’attributo stato di lavorazione questa operazione viene resa piu veloce. | IDX |
| **Tabella OrdinePizzaPlus** | |
| **Indice PRIMARY** | **Tipo:** |
| Numero, Tavolo | PR |
| **Indice OrdinePizzaPlus\_FK\_Tavolo\_idx** | **Tipo:** |
| Tavolo | IDX |
| **Indice OrdinePizzaPlus\_FK\_Pizza\_idx** | **Tipo:** |
| PizzaPlus | IDX |
| **Indice OrdinePizzaPlus\_FK\_Scontrino\_idx** | **Tipo:** |
| Scontrino | IDX |
| **Indice OrdinePizzaPlus**\_**StatoLavorazione\_idx** | **Tipo:** |
| StatoLavorazione  Stessa motivazione di OrdinePizza. | IDX |
| **Tabella OrdineBevanda** | |
| **Indice PRIMARY** | **Tipo:** |
| Numero, Tavolo | PR |
| **Indice OrdineBevanda\_FK\_Tavolo\_idx** | **Tipo:** |
| Tavolo | IDX |
| **Indice OrdineBevanda\_FK\_Pizza\_idx** | **Tipo:** |
| Bevanda | IDX |
| **Indice OrdineBevanda\_FK\_Scontrino\_idx** | **Tipo:** |
| Scontrino | IDX |
| **Indice OrdineBevanda\_StatoLavorazione\_idx** | **Tipo:** |
| StatoLavorazione  Stessa motivazione di OrdinePizza. | IDX |
| **Tabella Tavolo** | |
| **Indice PRIMARY** | **Tipo:** |
| Numero | PR |
| **Indice Tavolo\_FK\_Cliente\_idx** | **Tipo:** |
| Cliente | IDX |
| **Tabella Cliente** | |
| **Indice PRIMARY** | **Tipo:** |
| CF | PR |
| **Tabella Turno** | |
| **Indice PRIMARY** | **Tipo:** |
| OraInizio, OraFine, Giorno | PR |
| **Tabella Impiegato** | |
| **Indice PRIMARY** | **Tipo:** |
| Matricola | PR |
| **Tabella Usato** | |
| **Indice PRIMARY** | **Tipo:** |
| Tavolo, TurnoOraInizio, TurnoOraFine, TurnoGiorno | PR |
| **Indice Usato\_Turno\_idx** | **Tipo:** |
| TurnoOraInizio, TurnoOraFine, TurnoGiorno | IDX |
| **Tabella Associato** | |
| **Indice PRIMARY** | **Tipo:** |
| Tavolo, Impiegato | PR |
| **Indice Associato\_FK\_Impiegato\_idx** | **Tipo:** |
| Impiegato | IDX |
| **Tabella Lavora** | |
| **Indice PRIMARY** | **Tipo:** |
| Impiegato,TurnoOraInizio, TurnoOraFine, TurnoGiorno | PR |
| **Indice Lavora\_FK\_Turno\_idx** | **Tipo:** |
| TurnoOraInizio, TurnoOraFine, TurnoGiorno | IDX |
| **Tabella Scontrino** | |
| **Indice PRIMARY** | **Tipo:** |
| IdTransazione | PR |
| **Indice Scontrino\_FK\_Tavolo\_idx** | **Tipo:** |
| Tavolo | IDX |
| **Indice Scontrino\_Data\_idx** | **Tipo:** |
| Data  Questo indice è stato inserito in quanto nella specifica sono richieste delle operazioni che permettono di visualizzare le entrate mensili e giornaliere della pizzeria, in particolare un indice sull’attributo data dello scontrino è comodo per velocizzare queste operazioni in quanto gli scontrini vengono filtrati per data. | IDX |

## Trigger

* Quando una pizza o una bevanda vengono ordinate oppure quando una pizza ed un ingrediente vengono scelti per fare una pizzaplus, la loro disponibilità deve essere ridotta di 1, questa è una regola aziendale che viene implementata tramite trigger.
  + Quando si ordina una pizza

CREATE DEFINER = CURRENT\_USER TRIGGER `mydb`.`OrdinePizza\_AFTER\_INSERT` AFTER INSERT ON `OrdinePizza` FOR EACH ROW

BEGIN

update mydb.Pizza

set Scorte = Scorte -1

where myDb.Pizza.Nome=NEW.Pizza ;

END

* + Quando si ordina una bevanda

CREATE DEFINER = CURRENT\_USER TRIGGER `mydb`.`OrdineBevanda\_AFTER\_INSERT` AFTER INSERT ON `OrdineBevanda` FOR EACH ROW

BEGIN

update mydb.Bevanda

set Scorte = Scorte -1

where myDb.Bevanda.Nome=NEW.Bevanda ;

END

* + Quando si vuole aggiungere un ingrediente ad una pizza

CREATE DEFINER = CURRENT\_USER TRIGGER `mydb`.`Aggiunta\_AFTER\_INSERT` AFTER INSERT ON `Aggiunta` FOR EACH ROW

BEGIN

update mydb.Ingrediente

set Scorte = Scorte -1

where myDb.Ingrediente.Nome=NEW.Ingrediente ;

END

* + Quando si vuole usare una pizza come base

CREATE DEFINER = CURRENT\_USER TRIGGER `mydb`.`PizzaPlus\_AFTER\_INSERT` AFTER INSERT ON `PizzaPlus` FOR EACH ROW

BEGIN

update mydb.Pizza

set Scorte = Scorte -1

where myDb.Pizza.Nome=NEW.Pizza ;

END

* Il prezzo di una pizza con ingredienti aggiuntivi deve essere pari alla somma del prezzo della pizza usata come base e dei prezzi dei vari ingredienti aggiuntivi, questa regola aziendale derivativa è implementata tramite due trigger, quando si inserisce una pizzaplus viene posto il suo prezzo pari a quello della pizza usata come base e successivamente ogni volta che viene aggiunto un ingrediente viene sommato al prezzo della pizzaplus il prezzo dell’ingrediente
  + Quando si inserisce la pizzaplus

CREATE DEFINER = CURRENT\_USER TRIGGER `mydb`.`PizzaPlus\_BEFORE\_INSERT\_1` BEFORE INSERT ON `PizzaPlus` FOR EACH ROW

BEGIN

declare tmp float;

select prezzo

from mydb.Pizza

where mydb.Pizza.Nome=NEW.Base

into tmp;

set NEW.Prezzo=tmp;

END

* + Quando si aggiunge un ingrediente

CREATE DEFINER = CURRENT\_USER TRIGGER `mydb`.`Aggiunta\_AFTER\_INSERT\_1` AFTER INSERT ON `Aggiunta` FOR EACH ROW

BEGIN

declare tmp float;

select Prezzo

from mydb.Ingrediente

where mydb.Ingrediente.Nome = NEW.Ingrediente into tmp;

update mydb.PizzaPlus

set Prezzo=Prezzo+tmp

where mydb.PizzaPlus.Nome = NEW.PizzaPlus;

END

## Eventi

Non sono stati previsti eventi.

## Viste

Non sono state previste viste.

## Stored Procedures e transazioni

* Aggiungi\_bevanda

CREATE PROCEDURE `aggiungi\_bevanda` (in bevanda VARCHAR(45), in prezzo FLOAT)

BEGIN

insert into dbPizzeria.Bevanda (dbPizzeria.Bevanda.Nome, dbPizzeria.Bevanda.Prezzo) values (bevanda, prezzo);

END

* Aggiungi\_impiegato

CREATE PROCEDURE `aggiungi\_impiegato` (in matricola INT, in nome VARCHAR(45), in cognome VARCHAR(45), in passwd VARCHAR(45), in ruolo INT)

BEGIN

insert into dbPizzeria.Impiegato values (matricola, nome, cognome, md5(passwd), ruolo);

END

* Aggiungi\_ingrediente

CREATE PROCEDURE `aggiungi\_ingrediente` (in ingrediente VARCHAR(45), in prezzo FLOAT)

BEGIN

insert into dbPizzeria.Ingrediente (dbPizzeria.Ingrediente.Nome, dbPizzeria.Ingrediente.Prezzo) values (ingrediente,prezzo);

END

* Aggiungi\_pizza

CREATE PROCEDURE `aggiungi\_pizza` (in pizza VARCHAR(45), in prezzo FLOAT)

BEGIN

insert into dbPizzeria.Pizza (dbPizzeria.Pizza.Nome, dbPizzeria.Pizza.Prezzo) values (pizza,prezzo);

END

* Aggiungi\_tavolo

CREATE PROCEDURE `aggiungi\_tavolo` (in numero INT, in posti INT)

BEGIN

insert into dbPizzeria.Tavolo (Numero, Posti) values (numero, posti);

END

* Aggiungi\_turno

CREATE PROCEDURE `aggiungi\_turno` (in ora\_inizio TIME, in ora\_fine TIME,in giorno INT)

BEGIN

insert into dbPizzeria.Turno values (ora\_inizio, ora\_fine, giorno);

END

* Assegna\_tavolo\_a\_cameriere

In questo caso si è scelto di controllare se il cameriere inserito è effettivamente un cameriere al fine di evitare eventuali inconsistenze dei dati dovute ad un errore umano, è stata utilizzata una transazione in quanto se per qualche motivo la select del controllo fallisce l’insert non deve essere eseguita, sempre al fine di evitare inconsistenze. Il livello di isolamento è read committed per evitare letture sporche sulla relazione tavolo.

CREATE PROCEDURE `assegna\_tavolo\_a\_cameriere` (in cameriere int,in tavolo int)

BEGIN

declare exit handler for sqlexception

begin

rollback; -- rollback any changes made in the transaction

resignal; -- raise again the sql exception to the caller

end;

set transaction isolation level read committed;

start transaction;

-- dato che solo un cameriere puo' avere associato un tavolo faccio un controllo

if((select Ruolo from dbPizzeria.Impiegato where dbPizzeria.Impiegato.Matricola=cameriere)<>'Cameriere') then

signal sqlstate '45000' set message\_text='L\'impiegato non e\' un cameriere.';

end if;

insert into dbPizzeria.Associato values(tavolo,cameriere);

commit;

END

* Assegna\_tavolo\_a\_cliente

In questo caso prima di assegnare il tavolo ad un cliente, si controlla che esso sia libero al fine di evitare eventuali errori umani, il motivo della scelta di usare una transazione read committed è lo stesso dell’operazione precedente.

CREATE PROCEDURE `assegna\_tavolo\_a\_cliente` (in cliente VARCHAR(16), in tavolo INT)

BEGIN

declare exit handler for sqlexception

begin

rollback; -- rollback any changes made in the transaction

resignal; -- raise again the sql exception to the caller

end;

set transaction isolation level read committed;

start transaction;

-- controllo che il tavolo non sia gia occupato

if((select count(\*) from dbPizzeria.Tavolo where dbPizzeria.Tavolo.Numero=tavolo and dbPizzeria.Tavolo.Cliente is not null)>0) then

signal sqlstate '45002' set message\_text='Il tavolo e\' gia occupato';

else

update dbPizzeria.Tavolo set dbPizzeria.Tavolo.Cliente=cliente where dbPizzeria.Tavolo.Numero=tavolo;

insert into dbPizzeria.Scontrino (dbPizzeria.Scontrino.Tavolo) values (tavolo);

end if;

commit;

END

* Assegna\_turno\_a\_impiegato

CREATE PROCEDURE `assegna\_turno\_a\_impiegato` (in impiegato INT, in ora\_inizio TIME,in ora\_fine TIME, in giorno INT)

BEGIN

insert into dbPizzeria.Lavora values (impiegato, ora\_inizio, ora\_fine, giorno);

END

* Assegna\_turno\_a\_tavolo

CREATE PROCEDURE `assegna\_turno\_a\_tavolo` (in tavolo INT,in ora\_inizio TIME,in ora\_fine TIME, in giorno INT)

BEGIN

insert into dbPizzeria.Usato values (tavolo, ora\_inizio, ora\_fine, giorno);

END

* Aumenta\_scorte\_bevanda

Si è scelto di controllare se la bevanda che viene inserita esiste in quanto altrimenti se l’utente fa un errore di scrittura l’operazione di update risulterebbe avvenuta con successo ma non modificherebbe nessuna riga. Il livello di isolamento della transazione è read committed al fine di evitare che se un cameriere registra un ordine mentre si fa quest’operazione si crei un’inconsistenza (anche se lieve).

CREATE PROCEDURE `aumenta\_scorte\_bevanda`(in bevanda varchar(45), in scorte int)

BEGIN

declare exit handler for sqlexception

begin

rollback; -- rollback any changes made in the transaction

resignal; -- raise again the sql exception to the caller

end;

set transaction isolation level read committed;

start transaction;

-- controllo che la bevanda esista

if((select count(\*) from dbPizzeria.Bevanda where dbPizzeria.Bevanda.Nome=bevanda)<1) then

signal sqlstate '45011' set message\_text='Il prodotto non esiste.';

end if;

UPDATE dbPizzeria.Bevanda

SET

dbPizzeria.Bevanda.Scorte = dbPizzeria.Bevanda.Scorte + scorte

WHERE

dbPizzeria.Bevanda.Nome = bevanda;

commit;

END

* Aumenta\_scorte\_ingrediente

Stessi motivi dell’operazione precedente.

CREATE PROCEDURE `aumenta\_scorte\_ingrediente`(in ingrediente varchar(45), in scorte int)

BEGIN

declare exit handler for sqlexception

begin

rollback; -- rollback any changes made in the transaction

resignal; -- raise again the sql exception to the caller

end;

set transaction isolation level read committed;

start transaction;

-- controllo che l'ingrediente esita

if((select count(\*) from dbPizzeria.Ingrediente where dbPizzeria.Ingrediente.Nome=ingrediente)<1) then

signal sqlstate '45011' set message\_text='Il prodotto non esiste.';

end if;

UPDATE dbPizzeria.Ingrediente

SET

dbPizzeria.Ingrediente.Scorte = dbPizzeria.Ingrediente.Scorte + scorte

WHERE

dbPizzeria.Ingrediente.Nome = ingrediente;

commit;

END

* Aumenta\_scorte\_pizza

Stessi motivi dell’operazione precedente.

CREATE PROCEDURE `aumenta\_scorte\_pizza`(in pizza varchar(45), in scorte int)

BEGIN

declare exit handler for sqlexception

begin

rollback; -- rollback any changes made in the transaction

resignal; -- raise again the sql exception to the caller

end;

set transaction isolation level read committed;

start transaction;

-- controllo che la pizza esista

if((select count(\*) from dbPizzeria.Pizza where dbPizzeria.Pizza.Nome=pizza)<1) then

signal sqlstate '45011' set message\_text='Il prodotto non esiste.';

end if;

UPDATE dbPizzeria.Pizza

SET

dbPizzeria.Pizza.Scorte = dbPizzeria.Pizza.Scorte + scorte

WHERE

dbPizzeria.Pizza.Nome = pizza;

commit;

END

* Aumenta\_scorte\_tutti\_prodotti

Stessi motivi dell’operazione precedente.

CREATE PROCEDURE `aumenta\_scorte\_tutti\_prodotti` (in scorte int)

BEGIN

declare exit handler for sqlexception

begin

rollback; -- rollback any changes made in the transaction

resignal; -- raise again the sql exception to the caller

end;

set transaction isolation level read committed;

start transaction;

UPDATE dbPizzeria.Pizza

SET

dbPizzeria.Pizza.Scorte = dbPizzeria.Pizza.Scorte + scorte;

UPDATE dbPizzeria.Ingrediente

SET

dbPizzeria.Ingrediente.Scorte = dbPizzeria.Ingrediente.Scorte + scorte;

UPDATE dbPizzeria.Bevanda

SET

dbPizzeria.Bevanda.Scorte = dbPizzeria.Bevanda.Scorte + scorte;

commit;

END

* Consegna\_ordine\_bevanda

In questa operazione e in tutte quelle che riguardano il passaggio di stato di lavorazione dell’ordine (in lavorazione>in carico, in carico > espletato, espletato>consegnato) si è scelto di controllare che l’ordine esista per evitare eventuali errori di digitazione umani che generano failure facendo si che l’operazione non dia errori ma non faccia ciò che ci si aspetta,

si è messo poi un controllo che assicura che il passaggio tra gli stati di lavorazione avvenga secondo l’ordine previsto (un ordine in lavorazione non può essere direttamente espletato, perché magari qualcun altro lo sta prendendo in carico), il livello di isolamento è read committed in quanto anche se il lock sulla tupla di interesse viene rilasciato alla select che serve a controllare se la tupla esiste eventuali transazioni concorrenti non creano problemi, se lo stato di lavorazione viene modificato da un'altra transazione semplicemente l’operazione non avrà esito positivo perché vuoldire che l’ordine è stato preso in carico da qualcun altro, questo può causare un leggero rallentamento del servizio del personale della pizzeria in compenso però aumenta le prestazioni dell’esecuzione delle procedure in quanto i lock sono meno stringenti.

CREATE PROCEDURE `consegna\_ordine\_bevanda` (in numero\_ordine INT)

BEGIN

declare exit handler for sqlexception

begin

rollback; -- rollback any changes made in the transaction

resignal; -- raise again the sql exception to the caller

end;

set transaction isolation level repeatable read;

start transaction;

-- controllo che l'ordine esista

if((select count(\*) from dbPizzeria.OrdineBevanda where dbPizzeria.OrdineBevanda.Numero=numero\_ordine)<1) then

signal sqlstate '45014' set message\_text='L\'ordine non esiste.';

end if;

-- controllo che l'ordine segua i passaggi di stato previsti

if((select StatoLavorazione from dbPizzeria.OrdineBevanda where dbPizzeria.OrdineBevanda.Numero=numero\_ordine)<> 'Espletato') then

signal sqlstate '45009' set message\_text='L\'ordine deve essere espletato per poter essere consegnato.';

end if;

UPDATE dbPizzeria.OrdineBevanda

SET

dbPizzeria.OrdineBevanda.StatoLavorazione = 'Consegnato'

WHERE

dbPizzeria.OrdineBevanda.Numero = numero\_ordine;

commit;

END

* Consegna\_ordine\_pizza

CREATE PROCEDURE `consegna\_ordine\_pizza` (in numero\_ordine INT)

BEGIN

declare exit handler for sqlexception

begin

rollback; -- rollback any changes made in the transaction

resignal; -- raise again the sql exception to the caller

end;

set transaction isolation level read committed;

start transaction;

-- controllo che l'ordine esita

if((select count(\*) from dbPizzeria.OrdinePizza where dbPizzeria.OrdinePizza.Numero=numero\_ordine)<1) then

signal sqlstate '45014' set message\_text='L\'ordine non esiste.';

end if;

-- controllo che l'ordine segua i passaggi di stato previsti

if((select StatoLavorazione from dbPizzeria.OrdinePizza where dbPizzeria.OrdinePizza.Numero=numero\_ordine)<> 'Espletato') then

signal sqlstate '45009' set message\_text='L\'ordine deve essere espletato per poter essere consegnato.';

end if;

UPDATE dbPizzeria.OrdinePizza

SET

dbPizzeria.OrdinePizza.StatoLavorazione = 'Consegnato'

WHERE

dbPizzeria.OrdinePizza.Numero = numero\_ordine;

commit;

END

* Consegna\_ordine\_pizzaplus

CREATE PROCEDURE `consegna\_ordine\_pizza\_plus` (in numero\_ordine INT)

BEGIN

declare exit handler for sqlexception

begin

rollback; -- rollback any changes made in the transaction

resignal; -- raise again the sql exception to the caller

end;

set transaction isolation level read committed;

start transaction;

-- controllo che l'ordine esista

if((select count(\*) from dbPizzeria.OrdinePizzaPlus where dbPizzeria.OrdinePlus.Numero=numero\_ordine)<1) then

signal sqlstate '45014' set message\_text='L\'ordine non esiste.';

end if;

-- controllo che l'ordine segua i passaggi di stato previsti

if((select StatoLavorazione from dbPizzeria.OrdinePizzaPlus where dbPizzeria.OrdinePizzaPlus.Numero=numero\_ordine)<> 'Espletato') then

signal sqlstate '45009' set message\_text='L\'ordine deve essere espletato per poter essere consegnato.';

end if;

UPDATE dbPizzeria.OrdinePizzaPlus

SET

dbPizzeria.OrdinePizzaPlus.StatoLavorazione = 'Consegnato'

WHERE

dbPizzeria.OrdinePizzaPlus.Numero = numero\_ordine;

commit;

END

* Espleta\_ordine\_bevanda

CREATE PROCEDURE `espleta\_ordine\_bevanda`(in numero\_ordine bigint)

BEGIN

declare exit handler for sqlexception

begin

rollback; -- rollback any changes made in the transaction

resignal; -- raise again the sql exception to the caller

end;

set transaction isolation level read committed;

start transaction;

-- controllo che l'ordine esista

if((select count(\*) from dbPizzeria.OrdineBevanda where dbPizzeria.OrdineBevanda.Numero=numero\_ordine)<1) then

signal sqlstate '45014' set message\_text='L\'ordine non esiste.';

end if;

-- controllo che l'ordine segua i passaggi di stato previsti

if((select StatoLavorazione from dbPizzeria.OrdineBevanda where dbPizzeria.OrdineBevanda.Numero=numero\_ordine)<> 'In carico') then

signal sqlstate '45008' set message\_text='L\'ordine deve essere in carico per poter essere espletato.';

end if;

UPDATE dbPizzeria.OrdineBevanda

SET

dbPizzeria.OrdineBevanda.StatoLavorazione = 'Espletato'

WHERE

dbPizzeria.OrdineBevanda.Numero = numero\_ordine;

commit;

END

* Espleta\_ordine\_pizza

CREATE PROCEDURE `espleta\_ordine\_pizza`(in numero\_ordine bigint)

BEGIN

declare exit handler for sqlexception

begin

rollback; -- rollback any changes made in the transaction

resignal; -- raise again the sql exception to the caller

end;

set transaction isolation level read committed;

start transaction;

-- controllo che l'ordine esista

if((select count(\*) from dbPizzeria.OrdinePizza where dbPizzeria.OrdinePizza.Numero=numero\_ordine)<1) then

signal sqlstate '45014' set message\_text='L\'ordine non esiste.';

end if;

-- controllo che l'ordine segua i passaggi di stato previsti

if((select StatoLavorazione from dbPizzeria.OrdinePizza where dbPizzeria.OrdinePizza.Numero=numero\_ordine)<> 'In carico') then

signal sqlstate '45008' set message\_text='L\'ordine deve essere in carico per poter essere espletato.';

end if;

UPDATE dbPizzeria.OrdinePizza

SET

dbPizzeria.OrdinePizza.StatoLavorazione = 'Espletato'

WHERE

dbPizzeria.OrdinePizza.Numero = numero\_ordine;

commit;

END

* Espleta\_ordine\_pizzaplus

CREATE PROCEDURE `espleta\_ordine\_pizzaplus`(in numero\_ordine bigint)

BEGIN

declare exit handler for sqlexception

begin

rollback; -- rollback any changes made in the transaction

resignal; -- raise again the sql exception to the caller

end;

set transaction isolation level read committed;

start transaction;

-- controllo che l'ordine esista

if((select count(\*) from dbPizzeria.OrdinePizzaPlus where dbPizzeria.OrdinePizzaPlus.Numero=numero\_ordine)<1) then

signal sqlstate '45014' set message\_text='L\'ordine non esiste.';

end if;

-- controllo che l'ordine segua i passaggi di stato previsti

if((select StatoLavorazione from dbPizzeria.OrdinePizzaPlus where dbPizzeria.OrdinePizzaPlus.Numero=numero\_ordine)<> 'In carico') then

signal sqlstate '45008' set message\_text='L\'ordine deve essere in carico per poter essere espletato.';

end if;

UPDATE dbPizzeria.OrdinePizzaPlus

SET

dbPizzeria.OrdinePizzaPlus.StatoLavorazione = 'Espletato'

WHERE

dbPizzeria.OrdinePizzaPlus.Numero = numero\_ordine;

commit;

END

* Login

CREATE PROCEDURE `login` (in matricola varchar(45), in passwd varchar(45), out var\_role int)

BEGIN

declare var\_user\_role ENUM('Manager', 'Cameriere', 'Pizzaiolo', 'Barman');

SELECT

Ruolo

FROM

dbPizzeria.Impiegato

WHERE

dbPizzeria.Impiegato.Matricola = matricola

AND dbPizzeria.Impiegato.Passwd = MD5(passwd) INTO var\_user\_role;

-- See the corresponding enum in the client

if var\_user\_role = 'Manager' then

set var\_role = 1;

elseif var\_user\_role = 'Cameriere' then

set var\_role = 2;

elseif var\_user\_role = 'Pizzaiolo' then

set var\_role = 3;

elseif var\_user\_role=’Barman’ then

set var\_role=4;

else

set var\_role = 5;

end if;

END

* Prendi\_in\_carico\_ordine\_bevanda

CREATE PROCEDURE `prendi\_in\_carico\_ordine\_bevanda` (in numero\_ordine bigint)

BEGIN

declare exit handler for sqlexception

begin

rollback; -- rollback any changes made in the transaction

resignal; -- raise again the sql exception to the caller

end;

set transaction isolation level read committed;

start transaction;

-- controllo che l'ordine esista

if((select count(\*) from dbPizzeria.OrdineBevanda where dbPizzeria.OrdineBevanda.Numero=numero\_ordine)<1) then

signal sqlstate '45014' set message\_text='L\'ordine non esiste.';

end if;

-- controllo che l'ordine segua i passaggi di stato previsti

if((select StatoLavorazione from dbPizzeria.OrdineBevanda where dbPizzeria.OrdineBevanda.Numero=numero\_ordine)<> 'In lavorazione') then

signal sqlstate '45010' set message\_text='L\'ordine deve essere in lavorazione per poter essere preso in carico.';

end if;

UPDATE dbPizzeria.OrdineBevanda

SET

dbPizzeria.OrdineBevanda.StatoLavorazione = 'In carico'

WHERE

dbPizzeria.OrdineBevanda.Numero = numero\_ordine;

commit;

END

* Prendi\_in\_carico\_ordine\_pizza

CREATE PROCEDURE `prendi\_in\_carico\_ordine\_pizza` (in numero\_ordine bigint)

BEGIN

declare exit handler for sqlexception

begin

rollback; -- rollback any changes made in the transaction

resignal; -- raise again the sql exception to the caller

end;

set transaction isolation level read committed;

start transaction;

-- controllo che l'ordine esista

if((select count(\*) from dbPizzeria.OrdinePizza where dbPizzeria.OrdinePizza.Numero=numero\_ordine)<1) then

signal sqlstate '45014' set message\_text='L\'ordine non esiste.';

end if;

-- controllo che l'ordine segua i passaggi di stato previsti

if((select StatoLavorazione from dbPizzeria.OrdinePizza where dbPizzeria.OrdinePizza.Numero=numero\_ordine)<> 'In lavorazione') then

signal sqlstate '45010' set message\_text='l\'ordine deve essere in lavorazione per poter essere preso in carico.';

end if;

UPDATE dbPizzeria.OrdinePizza

SET

dbPizzeria.OrdinePizza.StatoLavorazione = 'In carico'

WHERE

dbPizzeria.OrdinePizza.Numero = numero\_ordine;

commit;

END

* Prendi\_in\_carico\_ordine\_pizza\_plus

CREATE PROCEDURE `prendi\_in\_carico\_ordine\_pizza\_plus` (in numero\_ordine bigint)

BEGIN

declare exit handler for sqlexception

begin

rollback; -- rollback any changes made in the transaction

resignal; -- raise again the sql exception to the caller

end;

set transaction isolation level read committed;

start transaction;

-- controllo che l'ordine esista

if((select count(\*) from dbPizzeria.OrdinePizzaPlus where dbPizzeria.OrdinePizzaPlus.Numero=numero\_ordine)<1) then

signal sqlstate '45014' set message\_text='L\'ordine non esiste.';

end if;

-- controllo che l'ordine segua i passaggi di stato previsti

if((select StatoLavorazione from dbPizzeria.OrdinePizzaPlus where dbPizzeria.OrdinePizzaPlus.Numero=numero\_ordine)<> 'In lavorazione') then

signal sqlstate '45010' set message\_text='l\'ordine deve essere in lavorazione per poter essere preso in carico.';

end if;

UPDATE dbPizzeria.OrdinePizzaPlus

SET

dbPizzeria.OrdinePizzaPlus.StatoLavorazione = 'In carico'

WHERE

dbPizzeria.OrdinePizzaPlus.Numero = numero\_ordine;

commit;

END

* Registra\_cliente

Si fa un controllo per vedere se il cliente già esiste, se c’è è possibile che sia venuto con un numero di commensali diversi rispetto all’ultima volta, per questo motivo si aggiorna il numero di commensali, se non esiste significa che è un nuovo cliente e viene registrato.

CREATE PROCEDURE `registra\_cliente` (in cf VARCHAR(16),in nome VARCHAR(45),in cognome VARCHAR(45), in commensali INT )

BEGIN

declare exit handler for sqlexception

begin

rollback; -- rollback any changes made in the transaction

resignal; -- raise again the sql exception to the caller

end;

set transaction isolation level read committed;

start transaction;

-- controllo se l'utente gia' esiste, in tal caso potrebbe essere venuto

-- con un numero di commensali diversi percio' aggiorno solo quello

if((select count(\*) from dbPizzeria.Cliente where dbPizzeria.Cliente.CF=cf)>0) then

update dbPizzeria.Cliente set dbPizzeria.Cliente.Commensali=commensali where dbPizzeria.Cliente.CF=cf;

else

insert into dbPizzeria.Cliente values (cf,nome,cognome,commensali);

end if;

commit;

END

* Registra\_ordine\_bevanda

In questo caso si verifica che il tavolo sia effettivamente occupato, in quanto a causa di un errore umano potrebbe verificarsi che viene registrato un ordine che però non ha fatto nessuno, si seleziona poi lo scontrino associato al tavolo (cioè l’unico relativo al tavolo che non ha ancora una data di emissione) e lo si utilizzerà per associare l’ordine allo scontrino e per aumentare l’importo di quest’ultimo, il livello di isolamento è read commited in quanto anche se lo scontrino subisce modifiche a causa di altri camerieri, questa cosa non influenza l’operazione di registrare l’ordine.

CREATE PROCEDURE `registra\_ordine\_bevanda` (in tavolo INT, in bevanda VARCHAR(45))

BEGIN

declare scontrino bigint;

declare prezzo\_bevanda float;

declare exit handler for sqlexception

begin

rollback; -- rollback any changes made in the transaction

resignal; -- raise again the sql exception to the caller

end;

set transaction isolation level read committed;

start transaction;

-- controllo che il tavolo sia occupato, altrimenti potrei avere un ordine che non ha fatto nessuno

if((select Cliente from dbPizzeria.Tavolo where dbPizzeria.Tavolo.Numero=tavolo) is null) then

signal sqlstate '45001' set message\_text='Il tavolo e\' libero.';

end if;

-- seleziono lo scontrino associato al tavolo che fa l'ordine

SELECT

IdTransazione

FROM

dbPizzeria.Scontrino

WHERE

dbPizzeria.Scontrino.Tavolo = tavolo

AND dbPizzeria.Scontrino.Emissione IS NULL INTO scontrino;

-- seleziono il prezzo della bevanda ordinata

SELECT

Prezzo

FROM

dbPizzeria.Bevanda

WHERE

dbPizzeria.Bevanda.Nome = bevanda INTO prezzo\_bevanda;

-- registro l'ordine e lo aggiungo al conto

insert into dbPizzeria.OrdineBevanda (Tavolo, Bevanda, Scontrino) values (tavolo, bevanda, scontrino);

-- aumento l'importo dello scontrino di una quantita' pari al prezzo della bevanda ordinata

UPDATE dbPizzeria.Scontrino

SET

dbPizzeria.Scontrino.Importo = dbPizzeria.Scontrino.Importo + prezzo\_bevanda

WHERE

dbPizzeria.Scontrino.IdTransazione = scontrino;

commit;

END

* Registra\_ordine\_pizza

Stesse motivazioni dell’operazione precedente.

CREATE PROCEDURE `registra\_ordine\_pizza` (in tavolo INT, in pizza VARCHAR(45))

BEGIN

declare scontrino bigint;

declare prezzo\_pizza float;

declare exit handler for sqlexception

begin

rollback; -- rollback any changes made in the transaction

resignal; -- raise again the sql exception to the caller

end;

set transaction isolation level read committed;

start transaction;

-- controllo che il tavolo sia occupato, altrimenti potrei avere un ordine che non ha fatto nessuno

if((select Cliente from dbPizzeria.Tavolo where dbPizzeria.Tavolo.Numero=tavolo) is null) then

signal sqlstate '45001' set message\_text='Il tavolo e\' libero.';

end if;

-- seleziono lo scontrino associato al tavolo

SELECT

IdTransazione

FROM

dbPizzeria.Scontrino

WHERE

dbPizzeria.Scontrino.Tavolo = tavolo

AND dbPizzeria.Scontrino.Emissione IS NULL INTO scontrino;

-- seleziono il prezzo della pizza ordinata

SELECT

Prezzo

FROM

dbPizzeria.Pizza

WHERE

dbPizzeria.Pizza.Nome = pizza INTO prezzo\_pizza;

-- registro l'ordine e lo aggiungo al conto

insert into dbPizzeria.OrdinePizza (Tavolo, Pizza, Scontrino) values (tavolo, pizza, scontrino);

-- aumento l'importo dello scontrino di una quantita' pari al prezzo della bevanda ordinata

UPDATE dbPizzeria.Scontrino

SET

dbPizzeria.Scontrino.Importo = dbPizzeria.Scontrino.Importo + prezzo\_pizza

WHERE

dbPizzeria.Scontrino.IdTransazione = scontrino;

commit;

END

* Registra\_ordine\_pizza\_plus

Quest’operazione di registrazione dell’ordine a differenza delle altre è un po’ più complessa perché include anche la creazione della pizzaplus in particolare viene creato il nome della pizzaplus in base a quanti ingredienti sono presenti, viene poi fatto un controllo sull’esistenza della pizzaplus col nome precedentemente generato, se esiste allora vengono solamente ridotte le scorte degli ingredienti e registrato l’ordine, se non esiste viene creata (e la diminuzione delle scorte avviene tramite trigger) e successivamente registrata, vengono anche fatte le operazioni necessarie all’aggiornamento dello scontrino viste nelle altre operazioni di registrazione dell’ordine. Il livello di isolamento è ancora read commited in quanto altre transazioni concorrenti potrebbero causare solo il fatto che un ingrediente viene terminato, tuttavia se ciò accadesse la transazione andrebbe in abort in quanto l’aggiornamento delle scorte non avrebbe successo.

CREATE PROCEDURE `registra\_ordine\_pizza\_plus` (in tavolo INT, in pizza VARCHAR(45), in ing\_1 VARCHAR(45), in ing\_2 VARCHAR(45), in ing\_3 VARCHAR(45), in ing\_4 VARCHAR(45), in ing\_5 VARCHAR(45) )

BEGIN

declare pizzaplus\_name VARCHAR(275);

declare scontrino bigint;

declare prezzo\_pizzaplus float;

declare exit handler for sqlexception

begin

rollback; -- rollback any changes made in the transaction

resignal; -- raise again the sql exception to the caller

end;

set transaction isolation level read committed;

start transaction;

-- controllo che il tavolo sia occupato, altrimenti potrei avere un ordine che non ha fatto nessuno

if((select Cliente from dbPizzeria.Tavolo where dbPizzeria.Tavolo.Numero=tavolo) is null) then

signal sqlstate '45001' set message\_text='Il tavolo e\' libero.';

end if;

-- seleziono lo scontrino associato al tavolo

SELECT

IdTransazione

FROM

dbPizzeria.Scontrino

WHERE

dbPizzeria.Scontrino.Tavolo = tavolo

AND dbPizzeria.Scontrino.Emissione IS NULL INTO scontrino;

-- costruisco il nome della pizzaplus in base a quanti ingredienti sono stati aggiunti

SELECT CONCAT(pizza, '\_', ing\_1) INTO pizzaplus\_name;

if (ing\_2<>'') then

select concat(pizzaplus\_name, '\_', ing\_2) into pizzaplus\_name;

end if;

if (ing\_3<>'') then

select concat(pizzaplus\_name, '\_', ing\_3) into pizzaplus\_name;

end if;

if (ing\_4<>'') then

select concat(pizzaplus\_name, '\_', ing\_4) into pizzaplus\_name;

end if;

if (ing\_5<>'') then

select concat(pizzaplus\_name, '\_', ing\_5) into pizzaplus\_name;

end if;

-- se la pizza con questi ingredienti esiste gia' vanno solo diminuite le scorte degli ingredienti aggiunti e della pizza usata come base

if((select count(\*) from dbPizzeria.PizzaPlus where dbPizzeria.PizzaPlus.Nome=pizzaplus\_name)>0) then

update dbPizzeria.Pizza set dbPizzeria.Pizza.Scorte=dbPizzeria.Pizza.Scorte-1 where dbPizzeria.Pizza.Nome=pizza;

UPDATE dbPizzeria.Ingrediente

SET

dbPizzeria.Ingrediente.Scorte = dbPizzeria.Ingrediente.Scorte - 1

WHERE

dbPizzeria.Ingrediente.Nome = ing\_1;

-- il primo ingrediente aggiuntivo e' sicuramente presente gli altri non necessariamente perciò faccio un controllo

if(ing\_2 <> '') then

update dbPizzeria.Ingrediente set dbPizzeria.Ingrediente.Scorte=dbPizzeria.Ingrediente.Scorte-1 where dbPizzeria.Ingrediente.Nome=ing\_2;

end if;

if(ing\_3 <> '') then

update dbPizzeria.Ingrediente set dbPizzeria.Ingrediente.Scorte=dbPizzeria.Ingrediente.Scorte-1 where dbPizzeria.Ingrediente.Nome=ing\_3;

end if;

if(ing\_4 <> '') then

update dbPizzeria.Ingrediente set dbPizzeria.Ingrediente.Scorte=dbPizzeria.Ingrediente.Scorte-1 where dbPizzeria.Ingrediente.Nome=ing\_4;

end if;

if(ing\_5 <> '') then

update dbPizzeria.Ingrediente set dbPizzeria.Ingrediente.Scorte=dbPizzeria.Ingrediente.Scorte-1 where dbPizzeria.Ingrediente.Nome=ing\_5;

end if;

-- la pizza con questi ingredienti aggiuntivi non esiste e quindi va creata

else

insert into dbPizzeria.PizzaPlus (dbPizzeria.PizzaPlus.Nome, dbPizzeria.PizzaPlus.Base) values (pizzaplus\_name, pizza);

insert into dbPizzeria.Aggiunta (dbPizzeria.Aggiunta.PizzaPlus, dbPizzeria.Aggiunta.Ingrediente) values (pizzaplus\_name, ing\_1);

-- il primo ingrediente aggiuntivo e' sicuramente presente gli altri non necessariamente perciò faccio un controllo

if(ing\_2 <> '') then

insert into dbPizzeria.Aggiunta (dbPizzeria.Aggiunta.PizzaPlus, dbPizzeria.Aggiunta.Ingrediente) values (pizzaplus\_name, ing\_2);

end if;

if(ing\_3 <> '') then

insert into dbPizzeria.Aggiunta (dbPizzeria.Aggiunta.PizzaPlus, dbPizzeria.Aggiunta.Ingrediente) values (pizzaplus\_name, ing\_3);

end if;

if(ing\_4 <> '') then

insert into dbPizzeria.Aggiunta (dbPizzeria.Aggiunta.PizzaPlus, dbPizzeria.Aggiunta.Ingrediente) values (pizzaplus\_name, ing\_4);

end if;

if(ing\_5 <> '') then

insert into dbPizzeria.Aggiunta (dbPizzeria.Aggiunta.PizzaPlus, dbPizzeria.Aggiunta.Ingrediente) values (pizzaplus\_name, ing\_5);

end if;

end if;

-- seleziono il prezzo della pizzaplus ordinata

SELECT

Prezzo

FROM

dbPizzeria.PizzaPlus

WHERE

dbPizzeria.PizzaPlus.Nome = pizzaplus\_name INTO prezzo\_pizzaplus;

-- registro l'ordine e lo aggiungo al conto

insert into dbPizzeria.OrdinePizzaPlus (Tavolo, PizzaPlus, Scontrino) values (tavolo, pizzaplus\_name, scontrino);

-- aumento l'importo dello scontrino di una quantita' pari al prezzo della bevanda ordinata

UPDATE dbPizzeria.Scontrino

SET

dbPizzeria.Scontrino.Importo = dbPizzeria.Scontrino.Importo + prezzo\_pizzaplus

WHERE

dbPizzeria.Scontrino.IdTransazione = scontrino;

commit;

END

* Rimuovi\_assegnazione\_tavolo\_a\_cameriere

CREATE PROCEDURE `rimuovi\_assegnazione\_tavolo` (in cameriere int, in tavolo int)

BEGIN

delete from dbPizzeria.Associato where dbPizzeria.Associato.Tavolo=tavolo and dbPizzeria.Associato.Impiegato=cameriere;

END

* Rimuovi\_bevanda

Viene controllata che la bevanda esista per evitare che a causa di un errore umano il manager pensi di aver eliminato un prodotto quando non è cosi.

CREATE PROCEDURE `rimuovi\_bevanda` (in bevanda VARCHAR(45))

BEGIN

declare exit handler for sqlexception

begin

rollback; -- rollback any changes made in the transaction

resignal; -- raise again the sql exception to the caller

end;

set transaction isolation level read committed;

start transaction;

-- controllo che la bevanda esista

if((select count(\*) from dbPizzeria.Bevanda where dbPizzeria.Bevanda.Nome=bevanda)<1) then

signal sqlstate '45011' set message\_text='Il prodotto non esiste.';

end if;

DELETE FROM dbPizzeria.Bevanda

WHERE

dbPizzeria.Bevanda.Nome = bevanda;

commit;

END

* Rimuovi\_impiegato

Stesse motivazioni dell’operazione precedente

CREATE PROCEDURE `rimuovi\_impiegato` (in matricola INT)

BEGIN

declare exit handler for sqlexception

begin

rollback; -- rollback any changes made in the transaction

resignal; -- raise again the sql exception to the caller

end;

set transaction isolation level read committed;

start transaction;

-- controllo che l'impiegato esista

if((select count(\*) from dbPizzeria.Impiegato where dbPizzeria.Impiegato.Matricola=matricola)<1) then

signal sqlstate '45013' set message\_text='L\'impiegato non esiste.';

end if;

DELETE FROM dbPizzeria.Impiegato

WHERE

dbPizzeria.Impiegato.Matricola = matricola;

commit;

END

* Rimuovi\_ingrediente

Stesse motivazioni dell’operazione precedente.

CREATE PROCEDURE `rimuovi\_ingrediente` (in ingrediente VARCHAR(45))

BEGIN

declare exit handler for sqlexception

begin

rollback; -- rollback any changes made in the transaction

resignal; -- raise again the sql exception to the caller

end;

set transaction isolation level read committed;

start transaction;

-- controllo che l'ingrediente esista

if((select count(\*) from dbPizzeria.Ingrediente where dbPizzeria.Ingrediente.Nome=ingrediente)<1) then

signal sqlstate '45011' set message\_text='Il prodotto non esiste.';

end if;

DELETE FROM dbPizzeria.Ingrediente

WHERE

dbPizzeria.Ingrediente.Nome = ingrediente;

commit;

END

* Rimuovi\_pizza

Stesse motivazioni dell’operazione precedente

CREATE PROCEDURE `rimuovi\_pizza` (in pizza VARCHAR(45))

BEGIN

declare exit handler for sqlexception

begin

rollback; -- rollback any changes made in the transaction

resignal; -- raise again the sql exception to the caller

end;

set transaction isolation level read committed;

start transaction;

-- controllo che la pizza esista

if((select count(\*) from dbPizzeria.Pizza where dbPizzeria.Pizza.Nome=pizza)<1) then

signal sqlstate '45011' set message\_text='Il prodotto non esiste.';

end if;

DELETE FROM dbPizzeria.Pizza

WHERE

dbPizzeria.Pizza.Nome = pizza;

commit;

END

* Rimuovi\_tavolo

Stesse motivazioni dell’operazione precedente

CREATE PROCEDURE `rimuovi\_tavolo` (in numero INT)

BEGIN

declare exit handler for sqlexception

begin

rollback; -- rollback any changes made in the transaction

resignal; -- raise again the sql exception to the caller

end;

set transaction isolation level read committed;

start transaction;

-- controllo che il tavolo esista

if((select count(\*) from dbPizzeria.Tavolo where dbPizzeria.Tavolo.Numero=numero)<1) then

signal sqlstate '45012' set message\_text='Il tavolo non esiste.';

end if;

DELETE FROM dbPizzeria.Tavolo

WHERE

dbPizzeria.Tavolo.Numero = numero;

commit;

END

* Rimuovi\_turno

Stesse motivazioni dell’operazione precedente

CREATE PROCEDURE `rimuovi\_turno` (in ora\_inizio TIME, in ora\_fine TIME, in giorno INT)

BEGIN

declare exit handler for sqlexception

begin

rollback; -- rollback any changes made in the transaction

resignal; -- raise again the sql exception to the caller

end;

set transaction isolation level read committed;

start transaction;

-- controllo che il turno esista

if((select count(\*) from dbPizzeria.Turno where dbPizzeria.Turno.OraInizio=ora\_inizio and dbPizzeria.Turno.OraFine=ora\_fine and dbPizzeria.Turno.Giorno=giorno)<1) then

signal sqlstate '45013' set message\_text='Il turno non esiste.';

end if;

DELETE FROM dbPizzeria.Turno

WHERE

dbPizzeria.Turno.OraInizio = ora\_inizio

AND dbPizzeria.Turno.OraFine = ora\_fine

AND dbPizzeria.Turno.Giorno = giorno;

commit;

END

* Rimuovi\_turno\_impiegato

CREATE PROCEDURE `rimuovi\_turno\_impiegato` (in impiegato INT, in ora\_inizio TIME,in ora\_fine TIME, in giorno INT)

BEGIN

delete from dbPizzeria.Lavora where dbPizzeria.Impiegato=impiegato and dbPizzeria.TurnoOraInizio=ora\_inizio and dbPizzeria.TurnoOraFine=ora\_fine and dbPizzeria.TurnoGiorno=giorno;

END

* Rimuovi\_turno\_tavolo

CREATE PROCEDURE `rimuovi\_turno\_tavolo` (in tavolo INT,in ora\_inizio TIME,in ora\_fine TIME, in giorno INT)

BEGIN

DELETE FROM dbPizzeria.Usato

WHERE

dbPizzeria.Tavolo.Numero = tavolo

AND dbPizzeria.TurnoOraInizio = ora\_inizio

AND dbPizzeria.TurnoOraFine = ora\_fine

AND dbPizzeria.TurnoGiorno = giorno;

END

* Stampa\_scontrino\_tavolo

Quest’operazione seleziona le informazioni dello scontrino ed i dettagli di tutti gli ordini associati allo scontrino, libera il tavolo e dissocia lo scontrino dal tavolo, essendo un’operazione di contabilità è molto delicata, si è scelto il livello di isolamento repeatable read in quanto essa accede più volte in selezione allo scontrino a livello di tupla.

CREATE PROCEDURE `stampa\_scontrino\_tavolo` (in tavolo INT)

BEGIN

declare id\_scontrino bigint;

declare exit handler for sqlexception

begin

rollback; -- rollback any changes made in the transaction

resignal; -- raise again the sql exception to the caller

end;

set transaction isolation level repeatable read;

start transaction;

-- seleziono lo scontrino associato al tavolo al quale lo sto stampando

SELECT

IdTransazione

FROM

dbPizzeria.Scontrino

WHERE

dbPizzeria.Scontrino.Tavolo = tavolo

AND dbPizzeria.Scontrino.Emissione IS NULL INTO id\_scontrino;

-- aggiungo la data di emissione allo scontrino

UPDATE dbPizzeria.Scontrino

SET

dbPizzeria.Scontrino.Emissione = NOW()

WHERE

dbPizzeria.Scontrino.IdTransazione=id\_scontrino;

-- selezioni le informazioni dello scontrino

SELECT

\*

FROM

dbPizzeria.Scontrino

WHERE

dbPizzeria.Scontrino.IdTransazione = id\_scontrino;

-- seleziono i dettagli di ogni ordine

SELECT

Nome, Prezzo

FROM

dbPizzeria.OrdinePizza

JOIN

dbPizzeria.Pizza ON dbPizzeria.OrdinePizza.Pizza = dbPizzeria.Pizza.Nome

WHERE

dbPizzeria.OrdinePizza.Scontrino = id\_scontrino

UNION ALL SELECT

Nome, Prezzo

FROM

dbPizzeria.OrdinePizzaPlus

JOIN

dbPizzeria.PizzaPlus ON dbPizzeria.OrdinePizzaPlus.PizzaPlus = dbPizzeria.PizzaPlus.Nome

WHERE

dbPizzeria.OrdinePizzaPlus.Scontrino = id\_scontrino

UNION ALL SELECT

Nome, Prezzo

FROM

dbPizzeria.OrdineBevanda

JOIN

dbPizzeria.Bevanda ON dbPizzeria.OrdineBevanda.Bevanda = dbPizzeria.Bevanda.Nome

WHERE

dbPizzeria.OrdineBevanda.Scontrino = id\_scontrino;

-- libero il tavolo

UPDATE dbPizzeria.Tavolo

SET

dbPizzeria.Tavolo.Cliente = NULL

WHERE

dbPizzeria.Tavolo.Numero = tavolo;

commit;

END

* Visualizza\_assegnazioni\_tavoli\_a\_camerieri

CREATE PROCEDURE `visualizza\_assegnazioni\_tavoli\_a\_camerieri` ()

BEGIN

set transaction read only;

set transaction isolation level read committed;

start transaction;

select Matricola,Nome,Cognome,Tavolo

from dbPizzeria.Associato join dbPizzeria.Impiegato on dbPizzeria.Associato.Impiegato=dbPizzeria.Impiegato.Matricola;

commit;

END

* Visualizza\_entrate\_giorno

CREATE PROCEDURE `visualizza\_entrate\_giorno` (in data\_di\_interesse date, out entrate float)

BEGIN

set transaction read only;

set transaction isolation level read committed;

start transaction;

select sum(Importo)

from dbPizzeria.Scontrino

where cast(dbPizzeria.Scontrino.Emissione as date)=data\_di\_interesse into entrate ;

commit;

END

* Visualizza\_entrate\_mese

CREATE PROCEDURE `visualizza\_entrate\_mese` (in data\_di\_interesse date, out entrate float)

BEGIN

set transaction read only;

set transaction isolation level read committed;

start transaction;

select sum(Importo)

from dbPizzeria.Scontrino

where month(dbPizzeria.Scontrino.Emissione)=month(data\_di\_interesse) and year(dbPizzeria.Scontrino.Emissione) = year(data\_di\_interesse) into entrate;

commit;

END

* Visualizza\_menu\_bevande

CREATE PROCEDURE `visualizza\_menu\_bevande` ()

BEGIN

set transaction read only;

set transaction isolation level read committed;

start transaction;

SELECT

\*

FROM

dbPizzeria.Bevanda;

commit;

END

* Visualizza\_menu\_ingredienti

CREATE PROCEDURE `visualizza\_menu\_ingredienti` ()

BEGIN

set transaction read only;

set transaction isolation level read committed;

start transaction;

SELECT

\*

FROM

dbPizzeria.Ingrediente;

commit;

END

* Visualizza\_menu\_pizze

CREATE PROCEDURE `visualizza\_menu\_pizze` ()

BEGIN

set transaction read only;

set transaction isolation level read committed;

start transaction;

SELECT

\*

FROM

dbPizzeria.Pizza;

commit;

END

* Visualizza\_ordini\_espletati\_bevanda

CREATE PROCEDURE `visualizza\_ordini\_espletati\_bevanda` (in cameriere INT)

BEGIN

set transaction read only;

set transaction isolation level read committed;

start transaction;

SELECT

dbPizzeria.OrdineBevanda.Numero,

dbPizzeria.OrdineBevanda.Bevanda,

dbPizzeria.OrdineBevanda.Tavolo

FROM

dbPizzeria.OrdineBevanda

JOIN

dbPizzeria.Associato ON dbPizzeria.OrdineBevanda.Tavolo = dbPizzeria.Associato.Tavolo

WHERE

dbPizzeria.Associato.Impiegato = cameriere

AND dbPizzeria.OrdineBevanda.StatoLavorazione = 'Espletato'

ORDER BY dbPizzeria.OrdineBevanda.Numero ASC;

commit;

END

* Visualizza\_ordini\_espletati\_pizza

CREATE PROCEDURE `visualizza\_ordini\_espletati\_pizza` (in cameriere int)

BEGIN

set transaction read only;

set transaction isolation level read committed;

start transaction;

SELECT

dbPizzeria.OrdinePizza.Numero,

dbPizzeria.OrdinePizza.Pizza,

dbPizzeria.OrdinePizza.Tavolo

FROM

dbPizzeria.OrdinePizza

JOIN

dbPizzeria.Associato ON dbPizzeria.OrdinePizza.Tavolo = dbPizzeria.Associato.Tavolo

WHERE

dbPizzeria.Associato.Impiegato = cameriere

AND dbPizzeria.OrdinePizza.StatoLavorazione = 'Espletato'

ORDER BY dbPizzeria.OrdinePizza.Numero ASC;

commit;

END

* Visualizza\_ordini\_espletati\_pizza\_plus

CREATE PROCEDURE `visualizza\_ordini\_espletati\_pizza\_plus` (in cameriere int)

BEGIN

set transaction read only;

set transaction isolation level read committed;

start transaction;

SELECT

dbPizzeria.OrdinePizzaPlus.Numero,

dbPizzeria.OrdinePizzaPlus.PizzaPlus,

dbPizzeria.OrdinePizzaPlus.Tavolo

FROM

dbPizzeria.OrdinePizzaPlus

JOIN

dbPizzeria.Associato ON dbPizzeria.OrdinePizzaPlus.Tavolo = dbPizzeria.Associato.Tavolo

WHERE

dbPizzeria.Associato.Impiegato = cameriere

AND dbPizzeria.OrdinePizzaPlus.StatoLavorazione = 'Espletato'

ORDER BY dbPizzeria.OrdinePizzaPlus.Numero ASC;

commit;

END

* Visualizza\_ordini\_bevanda\_da\_espletare

CREATE PROCEDURE `visualizza\_ordini\_bevanda\_da\_espletare` ()

BEGIN

set transaction read only;

set transaction isolation level read committed;

start transaction;

SELECT

Bevanda, Numero, Tavolo

FROM

dbPizzeria.OrdineBevanda

WHERE

dbPizzeria.OrdineBevanda.StatoLavorazione = 'In lavorazione'

ORDER BY dbPizzeria.OrdineBevanda.Numero ASC;

commit;

END

* Visualizza\_ordini\_pizza\_da\_espletare

CREATE PROCEDURE `visualizza\_ordini\_pizza\_da\_espletare` ()

BEGIN

set transaction read only;

set transaction isolation level read committed;

start transaction;

SELECT

Pizza, Numero, Tavolo

FROM

dbPizzeria.OrdinePizza

WHERE

dbPizzeria.OrdinePizza.StatoLavorazione = 'In lavorazione'

ORDER BY dbPizzeria.OrdinePizza.Numero ASC;

commit;

END

* Visualizza\_ordini\_pizza\_plus\_da\_espletare

CREATE PROCEDURE `visualizza\_ordini\_pizza\_plus\_da\_espletare` ()

BEGIN

set transaction read only;

set transaction isolation level read committed;

start transaction;

SELECT

PizzaPlus, Numero, Tavolo

FROM

dbPizzeria.OrdinePizzaPlus

WHERE

dbPizzeria.OrdinePizzaPlus.StatoLavorazione = 'In lavorazione'

ORDER BY dbPizzeria.OrdinePizzaPlus.Numero ASC;

commit;

END

* Visualizza\_tavoli

CREATE PROCEDURE `visualizza\_tavoli` ()

BEGIN

set transaction read only;

set transaction isolation level read committed;

start transaction;

SELECT

\*

FROM

dbPizzeria.Tavolo;

commit;

END

* Visualizza\_info\_tavoli\_associati

Questa è l’operazione che fa il cameriere, da specifica è richiesto che possa vedere quali tavoli a lui associati sono occupati, quali sono liberi e quali sono stati serviti, i tavoli serviti vengono riconosciuti guardando se hanno ordini in lavorazione, in carico o espletati, se venisse incluso anche consegnati ogni tavolo a cui si è seduto almeno un cliente nel suo ciclo di vita risulterebbe servito. Si è scelto il livello di isolamento serializable in quanto si accede più volte all’entità tavolo con delle operazioni che selezionano un gruppo di tuple e ciò potrebbe causare un inserimento fantasma.

CREATE PROCEDURE `visualizza\_info\_tavoli\_associati` (in cameriere INT)

BEGIN

set transaction read only;

set transaction isolation level serializable;

start transaction;

-- controllo che l'impiegato esista

if((select count(\*) from dbPizzeria.Impiegato where dbPizzeria.Impiegato.Matricola=cameriere<>'Cameriere')<1) then

signal sqlstate'45013' set message\_text='L\'impiegato non esiste.';

end if;

-- controllo che l'impiegato sia un cameriere

if((select Ruolo from dbPizzeria.Impiegato where dbPizzeria.Impiegato.Matricola=cameriere)<>'Cameriere') then

signal sqlstate '45000' set message\_text='L\'impiegato non e\' un cameriere.';

end if;

-- seleziono i tavoli liberi associati al cameriere

SELECT

Tavolo

FROM

dbPizzeria.Associato

JOIN

dbPizzeria.Tavolo ON dbPizzeria.Associato.Tavolo = dbPizzeria.Tavolo.Numero

WHERE

dbPizzeria.Associato.Impiegato = cameriere

AND dbPizzeria.Tavolo.Cliente IS NULL;

-- seleziono i tavoli occupati associati al cameriere

SELECT

Tavolo

FROM

dbPizzeria.Associato

JOIN

dbPizzeria.Tavolo ON dbPizzeria.Associato.Tavolo = dbPizzeria.Tavolo.Numero

WHERE

dbPizzeria.Associato.Impiegato = cameriere

AND dbPizzeria.Tavolo.Cliente IS NOT NULL;

-- seleziono i tavoli che hanno qualche ordine associato che sia in lavorazione, in carico oppure espletato, cioe' quelli che sono stati serviti

SELECT DISTINCT

dbPizzeria.Tavolo.Numero

FROM

dbPizzeria.Tavolo,

dbPizzeria.OrdineBevanda,

dbPizzeria.OrdinePizza,

dbPizzeria.OrdinePizzaPlus

WHERE

((dbPizzeria.Tavolo.Numero = dbPizzeria.OrdinePizza.Tavolo

AND dbPizzeria.OrdinePizza.StatoLavorazione <> 'Consegnato')

OR (dbPizzeria.Tavolo.Numero = dbPizzeria.OrdineBevanda.Tavolo

AND dbPizzeria.OrdineBevanda.StatoLavorazione <> 'Consegnato')

OR (dbPizzeria.Tavolo.Numero = dbPizzeria.OrdinePizzaPlus.Tavolo

AND dbPizzeria.OrdinePizzaPlus.StatoLavorazione <> 'Consegnato'))

AND dbPizzeria.Tavolo.Numero IN (SELECT

dbPizzeria.Tavolo.Numero

FROM

dbPizzeria.Associato

JOIN

dbPizzeria.Tavolo ON dbPizzeria.Associato.Tavolo = dbPizzeria.Tavolo.Numero

WHERE

dbPizzeria.Associato.Impiegato = cameriere

AND dbPizzeria.Tavolo.Cliente IS NOT NULL);

commit;

END

* Visualizza\_turni\_impiegati

CREATE PROCEDURE `visualizza\_turni\_impiegati` ()

BEGIN

set transaction read only;

set transaction isolation level read committed;

start transaction;

SELECT

Matricola,

Nome,

Cognome,

TurnoOraInizio,

TurnoOraFine,

TurnoGiorno

FROM

dbPizzeria.Impiegato

JOIN

dbPizzeria.Lavora ON dbPizzeria.Impiegato.Matricola = dbPizzeria.Lavora.Impiegato;

commit;

END

* Visualizza\_turni\_tavoli

CREATE PROCEDURE `visualizza\_turni\_tavoli` ()

BEGIN

set transaction read only;

set transaction isolation level read committed;

start transaction;

SELECT

\*

FROM

dbPizzeria.Usato;

commit;

END

* Visualizza turni, questa procedura non è stata identificata in fase di progettazione ma si è rivelata utile al fine di rendere l’applicazione più usabile.

CREATE PROCEDURE `visualizza\_turni` ()

BEGIN

set transaction read only;

set transaction isolation level read committed;

start transaction;

SELECT

\*

FROM

dbPizzeria.Turno;

commit;

END

* Visualizza impiegati, anche questa è stata aggiunta in fase di implementazione per le stesse motivazioni della precedente

CREATE PROCEDURE `visualizza\_impiegati` ()

BEGIN

set transaction read only;

set transaction isolation level read committed;

start transaction;

SELECT

Matricola,

Nome,

Cognome,

Ruolo

FROM

dbPizzeria.Impiegato;

commit;

END

* Visualizza camerieri, aggiunta in fase implementativa per gli stessi motivi delle due precedenti operazioni

CREATE PROCEDURE `visualizza\_camerieri` ()

BEGIN

set transaction read only;

set transaction isolation level read committed;

start transaction;

SELECT

Matricola, Nome, Cognome

FROM

dbPizzeria.Impiegato

WHERE

dbPizzeria.Impiegato.Ruolo = 'cameriere';

commit;

END

# Appendice: Implementazione

## Codice SQL per istanziare il database

**Nota:**

In fondo al codice per istanziare il database è presente una chiamata a stored procedure che inserisce un utente di default con ruolo di manager, matricola 1 e nome, cognome e password “init” necessario al cliente per avere i permessi necessari a configurare il suo account di manager, sarà sua cura successivamente rimuovere tale utente per ovvie motivazioni di sicurezza.

-- MySQL Workbench Forward Engineering

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

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

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

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

-- Schema dbPizzeria

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

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

-- Schema dbPizzeria

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

CREATE SCHEMA IF NOT EXISTS `dbPizzeria` DEFAULT CHARACTER SET utf8mb4 ;

USE `dbPizzeria` ;

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

-- Table `dbPizzeria`.`Cliente`

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

CREATE TABLE IF NOT EXISTS `dbPizzeria`.`Cliente` (

`CF` VARCHAR(16) NOT NULL,

`Nome` VARCHAR(45) NOT NULL,

`Cognome` VARCHAR(45) NOT NULL,

`Commensali` INT UNSIGNED NOT NULL DEFAULT 0,

PRIMARY KEY (`CF`))

ENGINE = InnoDB;

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

-- Table `dbPizzeria`.`Tavolo`

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

CREATE TABLE IF NOT EXISTS `dbPizzeria`.`Tavolo` (

`Numero` INT UNSIGNED NOT NULL,

`Posti` INT UNSIGNED NOT NULL DEFAULT 2,

`Cliente` VARCHAR(16) NULL DEFAULT NULL,

PRIMARY KEY (`Numero`),

INDEX `Tavolo\_FK\_Cliente\_idx` (`Cliente` ASC) VISIBLE,

CONSTRAINT `Tavolo\_FK\_Cliente`

FOREIGN KEY (`Cliente`)

REFERENCES `dbPizzeria`.`Cliente` (`CF`)

ON DELETE SET NULL

ON UPDATE CASCADE)

ENGINE = InnoDB;

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

-- Table `dbPizzeria`.`Turno`

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

CREATE TABLE IF NOT EXISTS `dbPizzeria`.`Turno` (

`OraInizio` TIME NOT NULL,

`OraFine` TIME NOT NULL,

`Giorno` ENUM('Domenica', 'Lunedi', 'Martedi', 'Mercoledi', 'Giovedi', 'Venerdi', 'Sabato') NOT NULL,

PRIMARY KEY (`OraInizio`, `OraFine`, `Giorno`))

ENGINE = InnoDB;

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

-- Table `dbPizzeria`.`Usato`

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

CREATE TABLE IF NOT EXISTS `dbPizzeria`.`Usato` (

`Tavolo` INT UNSIGNED NOT NULL,

`TurnoOraInizio` TIME NOT NULL,

`TurnoOraFine` TIME NOT NULL,

`TurnoGiorno` ENUM('Domenica', 'Lunedi', 'Martedi', 'Mercoledi', 'Giovedi', 'Venerdi', 'Sabato') NOT NULL,

PRIMARY KEY (`Tavolo`, `TurnoOraInizio`, `TurnoOraFine`, `TurnoGiorno`),

INDEX `Usato\_Turno\_idx` (`TurnoOraInizio` ASC, `TurnoOraFine` ASC, `TurnoGiorno` ASC) VISIBLE,

CONSTRAINT `Usato\_FK\_Tavolo`

FOREIGN KEY (`Tavolo`)

REFERENCES `dbPizzeria`.`Tavolo` (`Numero`)

ON DELETE CASCADE

ON UPDATE CASCADE,

CONSTRAINT `Usato\_FK\_Turno`

FOREIGN KEY (`TurnoOraInizio` , `TurnoOraFine` , `TurnoGiorno`)

REFERENCES `dbPizzeria`.`Turno` (`OraInizio` , `OraFine` , `Giorno`)

ON DELETE CASCADE

ON UPDATE CASCADE)

ENGINE = InnoDB;

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

-- Table `dbPizzeria`.`Impiegato`

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

CREATE TABLE IF NOT EXISTS `dbPizzeria`.`Impiegato` (

`Matricola` INT UNSIGNED NOT NULL,

`Nome` VARCHAR(45) NOT NULL,

`Cognome` VARCHAR(45) NOT NULL,

`Passwd` CHAR(45) NOT NULL DEFAULT 'password',

`Ruolo` ENUM('Manager', 'Cameriere', 'Pizzaiolo', 'Barman') NOT NULL,

PRIMARY KEY (`Matricola`))

ENGINE = InnoDB;

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

-- Table `dbPizzeria`.`Lavora`

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

CREATE TABLE IF NOT EXISTS `dbPizzeria`.`Lavora` (

`Impiegato` INT UNSIGNED NOT NULL,

`TurnoOraInizio` TIME NOT NULL,

`TurnoOraFine` TIME NOT NULL,

`TurnoGiorno` ENUM('Domenica', 'Lunedi', 'Martedi', 'Mercoledi', 'Giovedi', 'Venerdi', 'Sabato') NOT NULL,

PRIMARY KEY (`Impiegato`, `TurnoOraInizio`, `TurnoOraFine`, `TurnoGiorno`),

INDEX `Lavora\_FK\_Turno\_idx` (`TurnoOraInizio` ASC, `TurnoOraFine` ASC, `TurnoGiorno` ASC) VISIBLE,

CONSTRAINT `Lavora\_FK\_Turno`

FOREIGN KEY (`TurnoOraInizio` , `TurnoOraFine` , `TurnoGiorno`)

REFERENCES `dbPizzeria`.`Turno` (`OraInizio` , `OraFine` , `Giorno`)

ON DELETE CASCADE

ON UPDATE CASCADE,

CONSTRAINT `Lavora\_FK\_Impiegato`

FOREIGN KEY (`Impiegato`)

REFERENCES `dbPizzeria`.`Impiegato` (`Matricola`)

ON DELETE CASCADE

ON UPDATE CASCADE)

ENGINE = InnoDB;

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

-- Table `dbPizzeria`.`Associato`

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

CREATE TABLE IF NOT EXISTS `dbPizzeria`.`Associato` (

`Tavolo` INT UNSIGNED NOT NULL,

`Impiegato` INT UNSIGNED NOT NULL,

PRIMARY KEY (`Tavolo`, `Impiegato`),

INDEX `Associato\_FK\_Impiegato\_idx` (`Impiegato` ASC) VISIBLE,

CONSTRAINT `Associato\_FK\_Tavolo`

FOREIGN KEY (`Tavolo`)

REFERENCES `dbPizzeria`.`Tavolo` (`Numero`)

ON DELETE CASCADE

ON UPDATE CASCADE,

CONSTRAINT `Associato\_FK\_Impiegato`

FOREIGN KEY (`Impiegato`)

REFERENCES `dbPizzeria`.`Impiegato` (`Matricola`)

ON DELETE CASCADE

ON UPDATE CASCADE)

ENGINE = InnoDB;

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

-- Table `dbPizzeria`.`Scontrino`

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

CREATE TABLE IF NOT EXISTS `dbPizzeria`.`Scontrino` (

`IdTransazione` BIGINT UNSIGNED NOT NULL AUTO\_INCREMENT,

`Emissione` DATETIME NULL,

`Importo` FLOAT UNSIGNED NOT NULL DEFAULT 0.0,

`Tavolo` INT UNSIGNED NULL,

INDEX `Scontrino\_FK\_Tavolo\_idx` (`Tavolo` ASC) VISIBLE,

PRIMARY KEY (`IdTransazione`),

INDEX `Scontrino\_Data\_idx` (`Emissione` ASC) VISIBLE,

CONSTRAINT `Scontrino\_FK\_Tavolo`

FOREIGN KEY (`Tavolo`)

REFERENCES `dbPizzeria`.`Tavolo` (`Numero`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

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

-- Table `dbPizzeria`.`Bevanda`

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

CREATE TABLE IF NOT EXISTS `dbPizzeria`.`Bevanda` (

`Nome` VARCHAR(45) NOT NULL,

`Scorte` INT UNSIGNED NOT NULL DEFAULT 0,

`Prezzo` FLOAT UNSIGNED NOT NULL,

PRIMARY KEY (`Nome`))

ENGINE = InnoDB;

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

-- Table `dbPizzeria`.`Pizza`

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

CREATE TABLE IF NOT EXISTS `dbPizzeria`.`Pizza` (

`Nome` VARCHAR(45) NOT NULL,

`Scorte` INT UNSIGNED NOT NULL DEFAULT 0,

`Prezzo` FLOAT UNSIGNED NOT NULL,

PRIMARY KEY (`Nome`))

ENGINE = InnoDB;

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

-- Table `dbPizzeria`.`Ingrediente`

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

CREATE TABLE IF NOT EXISTS `dbPizzeria`.`Ingrediente` (

`Nome` VARCHAR(45) NOT NULL,

`Scorte` INT UNSIGNED NOT NULL DEFAULT 0,

`Prezzo` FLOAT UNSIGNED NOT NULL,

PRIMARY KEY (`Nome`))

ENGINE = InnoDB;

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

-- Table `dbPizzeria`.`PizzaPlus`

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

CREATE TABLE IF NOT EXISTS `dbPizzeria`.`PizzaPlus` (

`Nome` VARCHAR(275) NOT NULL,

`Prezzo` FLOAT UNSIGNED NOT NULL DEFAULT 0.0,

`Base` VARCHAR(45) NOT NULL,

PRIMARY KEY (`Nome`),

INDEX `Pizza+\_FK\_Pizza\_idx` (`Base` ASC) VISIBLE,

CONSTRAINT `PizzaPlus\_FK\_Pizza`

FOREIGN KEY (`Base`)

REFERENCES `dbPizzeria`.`Pizza` (`Nome`)

ON DELETE CASCADE

ON UPDATE CASCADE)

ENGINE = InnoDB;

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

-- Table `dbPizzeria`.`Aggiunta`

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

CREATE TABLE IF NOT EXISTS `dbPizzeria`.`Aggiunta` (

`PizzaPlus` VARCHAR(275) NOT NULL,

`Ingrediente` VARCHAR(45) NOT NULL,

PRIMARY KEY (`PizzaPlus`, `Ingrediente`),

INDEX `Aggiunta\_FK\_Ingrediente\_idx` (`Ingrediente` ASC) VISIBLE,

CONSTRAINT `Aggiunta\_FK\_PizzaPlus`

FOREIGN KEY (`PizzaPlus`)

REFERENCES `dbPizzeria`.`PizzaPlus` (`Nome`)

ON DELETE CASCADE

ON UPDATE CASCADE,

CONSTRAINT `Aggiunta\_FK\_Ingrediente`

FOREIGN KEY (`Ingrediente`)

REFERENCES `dbPizzeria`.`Ingrediente` (`Nome`)

ON DELETE CASCADE

ON UPDATE CASCADE)

ENGINE = InnoDB;

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

-- Table `dbPizzeria`.`OrdinePizza`

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

CREATE TABLE IF NOT EXISTS `dbPizzeria`.`OrdinePizza` (

`Numero` BIGINT UNSIGNED NOT NULL AUTO\_INCREMENT,

`Tavolo` INT UNSIGNED NOT NULL,

`Pizza` VARCHAR(45) NULL,

`StatoLavorazione` ENUM('In carico', 'In lavorazione', 'Espletato', 'Consegnato', 'Pagato') NOT NULL DEFAULT 'In lavorazione',

`Scontrino` BIGINT UNSIGNED NULL,

PRIMARY KEY (`Numero`, `Tavolo`),

INDEX `OrdinePizza\_FK\_Tavolo\_idx` (`Tavolo` ASC) VISIBLE,

INDEX `OrdinePizza\_FK\_Pizza\_idx` (`Pizza` ASC) VISIBLE,

INDEX `OrdinePizza\_FK\_Scontrino\_idx` (`Scontrino` ASC) VISIBLE,

INDEX `OrdinePizza\_StatoLavorazione\_idx` (`StatoLavorazione` ASC) VISIBLE,

CONSTRAINT `OrdinePizza\_FK\_Tavolo`

FOREIGN KEY (`Tavolo`)

REFERENCES `dbPizzeria`.`Tavolo` (`Numero`)

ON DELETE CASCADE

ON UPDATE CASCADE,

CONSTRAINT `OrdinePizza\_FK\_Pizza`

FOREIGN KEY (`Pizza`)

REFERENCES `dbPizzeria`.`Pizza` (`Nome`)

ON DELETE CASCADE

ON UPDATE CASCADE,

CONSTRAINT `OrdinePizza\_FK\_Scontrino`

FOREIGN KEY (`Scontrino`)

REFERENCES `dbPizzeria`.`Scontrino` (`IdTransazione`)

ON DELETE SET NULL

ON UPDATE CASCADE)

ENGINE = InnoDB;

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

-- Table `dbPizzeria`.`OrdineBevanda`

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

CREATE TABLE IF NOT EXISTS `dbPizzeria`.`OrdineBevanda` (

`Numero` BIGINT UNSIGNED NOT NULL AUTO\_INCREMENT,

`Tavolo` INT UNSIGNED NOT NULL,

`Bevanda` VARCHAR(45) NULL,

`StatoLavorazione` ENUM('In lavorazione', 'In carico', 'Espletato', 'Consegnato', 'Pagato') NOT NULL DEFAULT 'In lavorazione',

`Scontrino` BIGINT UNSIGNED NULL,

PRIMARY KEY (`Numero`, `Tavolo`),

INDEX `Bevanda\_FK\_Bevanda\_idx` (`Bevanda` ASC) VISIBLE,

INDEX `OrdineBevanda\_FK\_Tavolo\_idx` (`Tavolo` ASC) VISIBLE,

INDEX `OrdineBevana\_FK\_Scontrino\_idx` (`Scontrino` ASC) VISIBLE,

INDEX `OrdineBevanda\_StatoLavorazione\_idx` (`StatoLavorazione` ASC) VISIBLE,

CONSTRAINT `OrdineBevanda\_FK\_Bevanda`

FOREIGN KEY (`Bevanda`)

REFERENCES `dbPizzeria`.`Bevanda` (`Nome`)

ON DELETE CASCADE

ON UPDATE CASCADE,

CONSTRAINT `OrdineBevanda\_FK\_Tavolo`

FOREIGN KEY (`Tavolo`)

REFERENCES `dbPizzeria`.`Tavolo` (`Numero`)

ON DELETE CASCADE

ON UPDATE CASCADE,

CONSTRAINT `OrdineBevanda\_FK\_Scontrino`

FOREIGN KEY (`Scontrino`)

REFERENCES `dbPizzeria`.`Scontrino` (`IdTransazione`)

ON DELETE SET NULL

ON UPDATE CASCADE)

ENGINE = InnoDB;

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

-- Table `dbPizzeria`.`OrdinePizzaPlus`

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

CREATE TABLE IF NOT EXISTS `dbPizzeria`.`OrdinePizzaPlus` (

`Numero` BIGINT UNSIGNED NOT NULL AUTO\_INCREMENT,

`Tavolo` INT UNSIGNED NOT NULL,

`PizzaPlus` VARCHAR(275) NULL,

`StatoLavorazione` ENUM('In carico', 'In lavorazione', 'Espletato', 'Consegnato', 'Pagato') NOT NULL DEFAULT 'In lavorazione',

`Scontrino` BIGINT UNSIGNED NULL,

PRIMARY KEY (`Numero`, `Tavolo`),

INDEX `OrdinePizzaPlus\_FK\_Tavolo\_idx` (`Tavolo` ASC) VISIBLE,

INDEX `OrdinePizzaPlus\_FK\_Pizza+\_idx` (`PizzaPlus` ASC) VISIBLE,

INDEX `OrdinePizzaPlus\_FK\_Scontrino\_idx` (`Scontrino` ASC) VISIBLE,

INDEX `OrdinePizzaPlus\_StatoLavorazione\_idx` (`StatoLavorazione` ASC) VISIBLE,

CONSTRAINT `OrdinePizzaPlus\_FK\_Tavolo`

FOREIGN KEY (`Tavolo`)

REFERENCES `dbPizzeria`.`Tavolo` (`Numero`)

ON DELETE CASCADE

ON UPDATE CASCADE,

CONSTRAINT `OrdinePizzaPlus\_FK\_PizzaPlus`

FOREIGN KEY (`PizzaPlus`)

REFERENCES `dbPizzeria`.`PizzaPlus` (`Nome`)

ON DELETE CASCADE

ON UPDATE CASCADE,

CONSTRAINT `OrdinePizzaPlus\_FK\_Scontrino`

FOREIGN KEY (`Scontrino`)

REFERENCES `dbPizzeria`.`Scontrino` (`IdTransazione`)

ON DELETE SET NULL

ON UPDATE CASCADE)

ENGINE = InnoDB;

USE `dbPizzeria` ;

DELIMITER ;

CREATE USER 'Barman' IDENTIFIED BY 'barman';

GRANT EXECUTE ON procedure `dbPizzeria`.`espleta\_ordine\_bevanda` TO 'Barman';

GRANT EXECUTE ON procedure `dbPizzeria`.`visualizza\_ordini\_bevanda\_da\_espletare` TO 'Barman';

GRANT EXECUTE ON procedure `dbPizzeria`.`prendi\_in\_carico\_ordine\_bevanda` TO 'Barman';

CREATE USER 'Pizzaiolo' IDENTIFIED BY 'pizzaiolo';

GRANT EXECUTE ON procedure `dbPizzeria`.`espleta\_ordine\_pizzaplus` TO 'Pizzaiolo';

GRANT EXECUTE ON procedure `dbPizzeria`.`visualizza\_ordini\_pizza\_plus\_da\_espletare` TO 'Pizzaiolo';

GRANT EXECUTE ON procedure `dbPizzeria`.`visualizza\_ordini\_pizza\_da\_espletare` TO 'Pizzaiolo';

GRANT EXECUTE ON procedure `dbPizzeria`.`prendi\_in\_carico\_ordine\_pizza\_plus` TO 'Pizzaiolo';

GRANT EXECUTE ON procedure `dbPizzeria`.`prendi\_in\_carico\_ordine\_pizza` TO 'Pizzaiolo';

GRANT EXECUTE ON procedure `dbPizzeria`.`espleta\_ordine\_pizza` TO 'Pizzaiolo';

CREATE USER 'Manager' IDENTIFIED BY 'manager';

GRANT EXECUTE ON procedure `dbPizzeria`.`aggiungi\_bevanda` TO 'Manager';

GRANT EXECUTE ON procedure `dbPizzeria`.`aggiungi\_impiegato` TO 'Manager';

GRANT EXECUTE ON procedure `dbPizzeria`.`aggiungi\_ingrediente` TO 'Manager';

GRANT EXECUTE ON procedure `dbPizzeria`.`aggiungi\_pizza` TO 'Manager';

GRANT EXECUTE ON procedure `dbPizzeria`.`aggiungi\_tavolo` TO 'Manager';

GRANT EXECUTE ON procedure `dbPizzeria`.`aggiungi\_turno` TO 'Manager';

GRANT EXECUTE ON procedure `dbPizzeria`.`assegna\_tavolo\_a\_cameriere` TO 'Manager';

GRANT EXECUTE ON procedure `dbPizzeria`.`assegna\_tavolo\_a\_cliente` TO 'Manager';

GRANT EXECUTE ON procedure `dbPizzeria`.`assegna\_turno\_a\_impiegato` TO 'Manager';

GRANT EXECUTE ON procedure `dbPizzeria`.`assegna\_turno\_a\_tavolo` TO 'Manager';

GRANT EXECUTE ON procedure `dbPizzeria`.`aumenta\_scorte\_bevanda` TO 'Manager';

GRANT EXECUTE ON procedure `dbPizzeria`.`aumenta\_scorte\_ingrediente` TO 'Manager';

GRANT EXECUTE ON procedure `dbPizzeria`.`visualizza\_turni\_tavoli` TO 'Manager';

GRANT EXECUTE ON procedure `dbPizzeria`.`visualizza\_turni\_impiegati` TO 'Manager';

GRANT EXECUTE ON procedure `dbPizzeria`.`visualizza\_tavoli` TO 'Manager';

GRANT EXECUTE ON procedure `dbPizzeria`.`visualizza\_menu\_pizze` TO 'Manager';

GRANT EXECUTE ON procedure `dbPizzeria`.`visualizza\_menu\_ingredienti` TO 'Manager';

GRANT EXECUTE ON procedure `dbPizzeria`.`visualizza\_menu\_bevande` TO 'Manager';

GRANT EXECUTE ON procedure `dbPizzeria`.`visualizza\_entrate\_mese` TO 'Manager';

GRANT EXECUTE ON procedure `dbPizzeria`.`visualizza\_entrate\_giorno` TO 'Manager';

GRANT EXECUTE ON procedure `dbPizzeria`.`visualizza\_assegnazioni\_tavoli\_a\_camerieri` TO 'Manager';

GRANT EXECUTE ON procedure `dbPizzeria`.`rimuovi\_turno` TO 'Manager';

GRANT EXECUTE ON procedure `dbPizzeria`.`rimuovi\_turno\_impiegato` TO 'Manager';

GRANT EXECUTE ON procedure `dbPizzeria`.`rimuovi\_turno\_tavolo` TO 'Manager';

GRANT EXECUTE ON procedure `dbPizzeria`.`rimuovi\_tavolo` TO 'Manager';

GRANT EXECUTE ON procedure `dbPizzeria`.`rimuovi\_pizza` TO 'Manager';

GRANT EXECUTE ON procedure `dbPizzeria`.`rimuovi\_ingrediente` TO 'Manager';

GRANT EXECUTE ON procedure `dbPizzeria`.`rimuovi\_impiegato` TO 'Manager';

GRANT EXECUTE ON procedure `dbPizzeria`.`rimuovi\_bevanda` TO 'Manager';

GRANT EXECUTE ON procedure `dbPizzeria`.`registra\_cliente` TO 'Manager';

GRANT EXECUTE ON procedure `dbPizzeria`.`rimuovi\_assegnazione\_tavolo` TO 'Manager';

GRANT EXECUTE ON procedure `dbPizzeria`.`aumenta\_scorte\_tutti\_prodotti` TO 'Manager';

GRANT EXECUTE ON procedure `dbPizzeria`.`aumenta\_scorte\_pizza` TO 'Manager';

GRANT EXECUTE ON procedure `dbPizzeria`.`stampa\_scontrino\_tavolo` TO 'Manager';

GRANT EXECUTE ON procedure `dbPizzeria`.`visualizza\_turni` TO 'Manager';

GRANT EXECUTE ON procedure `dbPizzeria`.`visualizza\_impiegati` TO 'Manager';

GRANT EXECUTE ON procedure `dbPizzeria`.`visualizza\_camerieri` TO 'Manager';

CREATE USER 'Cameriere' IDENTIFIED BY 'cameriere';

GRANT EXECUTE ON procedure `dbPizzeria`.`consegna\_ordine\_pizza\_plus` TO 'Cameriere';

GRANT EXECUTE ON procedure `dbPizzeria`.`visualizza\_info\_tavoli\_associati` TO 'Cameriere';

GRANT EXECUTE ON procedure `dbPizzeria`.`visualizza\_ordini\_espletati\_pizza\_plus` TO 'Cameriere';

GRANT EXECUTE ON procedure `dbPizzeria`.`visualizza\_ordini\_espletati\_pizza` TO 'Cameriere';

GRANT EXECUTE ON procedure `dbPizzeria`.`visualizza\_ordini\_espletati\_bevanda` TO 'Cameriere';

GRANT EXECUTE ON procedure `dbPizzeria`.`visualizza\_menu\_pizze` TO 'Cameriere';

GRANT EXECUTE ON procedure `dbPizzeria`.`visualizza\_menu\_ingredienti` TO 'Cameriere';

GRANT EXECUTE ON procedure `dbPizzeria`.`visualizza\_menu\_bevande` TO 'Cameriere';

GRANT EXECUTE ON procedure `dbPizzeria`.`registra\_ordine\_bevanda` TO 'Cameriere';

GRANT EXECUTE ON procedure `dbPizzeria`.`registra\_ordine\_pizza` TO 'Cameriere';

GRANT EXECUTE ON procedure `dbPizzeria`.`registra\_ordine\_pizza\_plus` TO 'Cameriere';

GRANT EXECUTE ON procedure `dbPizzeria`.`consegna\_ordine\_bevanda` TO 'Cameriere';

GRANT EXECUTE ON procedure `dbPizzeria`.`consegna\_ordine\_pizza` TO 'Cameriere';

CREATE USER 'Login' IDENTIFIED BY 'login';

GRANT EXECUTE ON procedure `dbPizzeria`.`login` TO 'Login';

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

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

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

-- default user

call dbPizzeria.aggiungi\_impiegato(1,’init’,’init’,’init’,1);

## Codice del Front-End

**Nota:**

Il codice di appoggio è stato leggermente modificato in alcuni punti per aggiungere il supporto ad altri tipi di dati e modificato a seguito del verificarsi di qualche failure, inoltre dai test è risultato più o meno tutto funzionante tranne due cose che a seguito di svariati tentativi di modifica non sono stati risolti (le stored procedure a seguito di test da mysql shell e da mysql workbench sono risultate corrette quindi il problema è nel client):

* In fase di dump del result set i valori null non vengono riconosciuti (nel controllo sul campo is\_null della MYSQL\_BIND) e appaiono come caratteri strani, inoltre in caso una tupla contenga un campo null e, la tupla precedente abbia invece un valore in quel campo allora esso invece di esserci un carattere strano viene stampato a schermo lo stesso valore della riga superiore (molto evidente in visualizza\_tavoli del manager, quando i tavoli non hanno un cliente associato e pertanto tale campo è null)
* Quando viene registrato un ordine di pizza plus come si può vedere nella stored procedure precedentemente riportata il nome della pizza plus quando creata viene “assemblato” come concatenazione degli ingredienti e della pizza usata come base separati dal carattere “+” questo tramite la funzione CONCAT(…) messa a disposizione da mysql, tuttavia il client visualizza solamente la prima stringa della concatenazione (cioè la pizza usata come base), questo anche utilizzando altri caratteri per dividere le stringhe o utilizzando CONCAT\_WS(),

per vedere se l’errore fosse un eventuale inserimento di un carattere terminatore di stringa ‘\0’ tra le varie stringhe da parte delle due funzioni è stato fatto un while in fase di dump del result set per stampare a schermo carattere per carattere i dati presenti nel buffer della bind in modo da ignorare eventuali terminatore, tuttavia il risultato non è cambiato.

* Main, per il login

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <mysql/mysql.h>

#include "defines.h"

typedef enum {

MANAGER = 1,

CAMERIERE,

PIZZAIOLO,

BARMAN,

FAILED\_LOGIN

} role\_t;

struct configuration conf;

static MYSQL \*conn;

static role\_t attempt\_login(MYSQL \*conn, int matricola, char \*password) {

MYSQL\_STMT \*login\_procedure;

MYSQL\_BIND param[3]; // Used both for input and output

int role = 0;

if(!setup\_prepared\_stmt(&login\_procedure, "call login(?, ?, ?)", conn)) {

print\_stmt\_error(login\_procedure, "Unable to initialize login statement\n");

goto err2;

}

// Prepare parameters

memset(param, 0, sizeof(param));

param[0].buffer\_type = MYSQL\_TYPE\_LONG; // IN

param[0].buffer = &matricola;

param[0].buffer\_length = sizeof(matricola);

param[1].buffer\_type = MYSQL\_TYPE\_VAR\_STRING; // IN

param[1].buffer = password;

param[1].buffer\_length = strlen(password);

param[2].buffer\_type = MYSQL\_TYPE\_LONG; // OUT

param[2].buffer = &role;

param[2].buffer\_length = sizeof(role);

if (mysql\_stmt\_bind\_param(login\_procedure, param) != 0) { // Note \_param

print\_stmt\_error(login\_procedure, "Could not bind parameters for login");

goto err;

}

// Run procedure

if (mysql\_stmt\_execute(login\_procedure) != 0) {

print\_stmt\_error(login\_procedure, "Could not execute login procedure");

goto err;

}

// Prepare output parameters

memset(param, 0, sizeof(param));

param[0].buffer\_type = MYSQL\_TYPE\_LONG; // OUT

param[0].buffer = &role;

param[0].buffer\_length = sizeof(role);

if(mysql\_stmt\_bind\_result(login\_procedure, param)) {

print\_stmt\_error(login\_procedure, "Could not retrieve output parameter");

goto err;

}

// Retrieve output parameter

if(mysql\_stmt\_fetch(login\_procedure)) {

print\_stmt\_error(login\_procedure, "Could not buffer results");

goto err;

}

mysql\_stmt\_close(login\_procedure);

return role;

err:

mysql\_stmt\_close(login\_procedure);

err2:

return FAILED\_LOGIN;

}

int main(void) {

role\_t role;

char user\_mat[10];

if(!parse\_config("users/login.json", &conf)) {

fprintf(stderr, "Unable to load login configuration\n");

exit(EXIT\_FAILURE);

}

conn = mysql\_init (NULL);

if (conn == NULL) {

fprintf (stderr, "mysql\_init() failed (probably out of memory)\n");

exit(EXIT\_FAILURE);

}

if (mysql\_real\_connect(conn, "localhost", "root", "root", "dbPizzeria", 3306, NULL, CLIENT\_MULTI\_STATEMENTS | CLIENT\_MULTI\_RESULTS) == NULL) {

//if (mysql\_real\_connect(conn, conf.host, conf.db\_username, conf.db\_password, conf.database, conf.port, NULL, CLIENT\_MULTI\_STATEMENTS | CLIENT\_MULTI\_RESULTS) == NULL) {

fprintf (stderr, "mysql\_real\_connect() failed\n");

mysql\_close (conn);

exit(EXIT\_FAILURE);

}

printf("Benvenuto, inserisci le tue credenziali\n");

printf("Matricola: ");

getInput(128, user\_mat, false);

conf.username=atoi(user\_mat);

printf("Password: ");

getInput(128, conf.password, true);

role = attempt\_login(conn, conf.username, conf.password);

switch(role) {

case MANAGER:

run\_as\_manager(conn);

break;

case CAMERIERE:

run\_as\_cameriere(conn);

break;

case PIZZAIOLO:

run\_as\_pizzaiolo(conn);

break;

case BARMAN:

run\_as\_barman(conn);

break;

case FAILED\_LOGIN:

fprintf(stderr, "Invalid credentials\n");

exit(EXIT\_FAILURE);

break;

default:

fprintf(stderr, "Invalid condition at %s:%d\n", \_\_FILE\_\_, \_\_LINE\_\_);

abort();

}

printf("Ciao!\n");

mysql\_close (conn);

return 0;

}

* Client del pizzaiolo

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <mysql/mysql.h>

#include "defines.h"

static void visualizza\_ordini\_pizza\_da\_espletare(MYSQL \*conn){

MYSQL\_STMT \*prepared\_stmt;

if(!setup\_prepared\_stmt(&prepared\_stmt, "call visualizza\_ordini\_pizza\_da\_espletare()", conn)) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Unable to initialize visualizza\_ordini\_pizza\_da\_espletare statement\n", false);

}

// Run procedure

if (mysql\_stmt\_execute(prepared\_stmt) != 0) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Could not retrieve visualizza\_ordini\_pizza\_da\_espletare\n", true);

}

// Dump the result set

dump\_result\_set(conn, prepared\_stmt, "\nLista delle pizze da espletare");

mysql\_stmt\_next\_result(prepared\_stmt);

mysql\_stmt\_close(prepared\_stmt);

}

static void espleta\_ordine\_pizza(MYSQL \*conn, long long int ordine){

MYSQL\_STMT \*prepared\_stmt;

MYSQL\_BIND param[1];

char check[2];

// Prepare parameters

memset(param, 0, sizeof(param));

memset(check,'\0', sizeof(check));

param[0].buffer\_type = MYSQL\_TYPE\_LONGLONG;

param[0].buffer = &ordine;

param[0].buffer\_length = sizeof(ordine);

check\_ok\_pizza:

//espleta ordine, viene messa questa scanf in modo da poter prende in carico l'ordine ed espletarlo solo

//quando e' stato effettivamente fatto, evitando al pizzaiolo di dover ricordare l'id dell'ordine

printf("Digitare ok per espletare l'ordine\n");

if(scanf("%s", check)<1){

printf("Errore nella conferma\n");

flush\_stdin();

goto check\_ok\_pizza;

}

flush\_stdin();

if(strcmp(check,"ok")!=0){

printf("Per favore digita ok\n");

goto check\_ok\_pizza;

}

// Prepare stored procedure call

if(!setup\_prepared\_stmt(&prepared\_stmt, "call espleta\_ordine\_pizza(?)", conn)) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Unable to initialize espleta\_ordine\_pizza statement\n", false);

}

// non vi e' reset perche i parametri sono gli stessi

if (mysql\_stmt\_bind\_param(prepared\_stmt, param) != 0) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Could not bind parameters for espleta\_ordine\_pizza\n", true);

}

// Run procedure

if (mysql\_stmt\_execute(prepared\_stmt) != 0) {

print\_stmt\_error (prepared\_stmt, "Errore nell'espletare in carico ordine pizza.\n");

return;

} else {

printf("Ordine pizza espletato correttamente\n");

}

mysql\_stmt\_close(prepared\_stmt);

}

static void prendi\_in\_carico\_ordine\_pizza(MYSQL \*conn) {

MYSQL\_STMT \*prepared\_stmt;

MYSQL\_BIND param[1];

long long int id\_ordine;

// Prepare stored procedure call

if(!setup\_prepared\_stmt(&prepared\_stmt, "call prendi\_in\_carico\_ordine\_pizza(?)", conn)) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Unable to initialize prendi\_in\_carico\_ordine\_pizza statement\n", false);

}

// Prepare parameters

memset(param, 0, sizeof(param));

printf("\nID ordine: ");

if(scanf("%lld", &id\_ordine)<1){

printf("Errore nell'acquisire indice ordine\n");

flush\_stdin();

run\_as\_pizzaiolo(conn);

}

flush\_stdin();

param[0].buffer\_type = MYSQL\_TYPE\_LONGLONG;

param[0].buffer = &id\_ordine;

param[0].buffer\_length = sizeof(id\_ordine);

if (mysql\_stmt\_bind\_param(prepared\_stmt, param) != 0) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Could not bind parameters for prendi\_in\_carico\_ordine\_pizza\n", true);

}

// Run procedure

if (mysql\_stmt\_execute(prepared\_stmt) != 0) {

print\_stmt\_error (prepared\_stmt, "Errore nel prendere in carico ordine pizza.\n");

return;

} else {

printf("Ordine pizza preso in carico correttamente\n");

}

mysql\_stmt\_close(prepared\_stmt);

espleta\_ordine\_pizza(conn, id\_ordine);

}

static void visualizza\_ordini\_pizza\_plus\_da\_espletare(MYSQL \*conn){

MYSQL\_STMT \*prepared\_stmt;

if(!setup\_prepared\_stmt(&prepared\_stmt, "call visualizza\_ordini\_pizza\_plus\_da\_espletare()", conn)) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Unable to initialize visualizza\_ordini\_pizza\_plus\_da\_espletare statement\n", false);

}

// Run procedure

if (mysql\_stmt\_execute(prepared\_stmt) != 0) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Could not retrieve visualizza\_ordini\_pizza\_plus\_da\_espletare\n", true);

}

// Dump the result set

dump\_result\_set(conn, prepared\_stmt, "\nLista delle pizze plus da espletare");

mysql\_stmt\_next\_result(prepared\_stmt);

mysql\_stmt\_close(prepared\_stmt);

}

static void espleta\_ordine\_pizza\_plus(MYSQL \*conn, long long int ordine){

MYSQL\_STMT \*prepared\_stmt;

MYSQL\_BIND param[1];

char check[2];

// Prepare parameters

memset(param, 0, sizeof(param));

memset(check,'\0', sizeof(check));

param[0].buffer\_type = MYSQL\_TYPE\_LONGLONG;

param[0].buffer = &ordine;

param[0].buffer\_length = sizeof(ordine);

check\_ok\_pizza\_plus:

//espleta ordine

printf("Digitare ok per espletare l'ordine\n");

if(scanf("%s", check)<1){

printf("Errore nella conferma\n");

flush\_stdin();

goto check\_ok\_pizza\_plus;

}

flush\_stdin();

if(strcmp(check,"ok")!=0){

printf("Per favore digita ok\n");

goto check\_ok\_pizza\_plus;

}

// Prepare stored procedure call

if(!setup\_prepared\_stmt(&prepared\_stmt, "call espleta\_ordine\_pizzaplus(?)", conn)) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Unable to initialize espleta\_ordine\_pizza\_plusstatement\n", false);

}

// non vi e' reset perche i parametri sono gli stessi

if (mysql\_stmt\_bind\_param(prepared\_stmt, param) != 0) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Could not bind parameters for espleta\_ordine\_pizza\_plus\n", true);

}

// Run procedure

if (mysql\_stmt\_execute(prepared\_stmt) != 0) {

print\_stmt\_error (prepared\_stmt, "Errore nell'espletare in carico ordine pizza plus.\n");

return;

} else {

printf("Ordine pizza plus espletato correttamente\n");

}

mysql\_stmt\_close(prepared\_stmt);

}

static void prendi\_in\_carico\_ordine\_pizza\_plus(MYSQL \*conn) {

MYSQL\_STMT \*prepared\_stmt;

MYSQL\_BIND param[1];

long long int id\_ordine;

// Prepare stored procedure call

if(!setup\_prepared\_stmt(&prepared\_stmt, "call prendi\_in\_carico\_ordine\_pizza\_plus(?)", conn)) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Unable to initialize prendi\_in\_carico\_ordine\_pizza\_plus statement\n", false);

}

//Prepare parameters

memset(param, 0, sizeof(param));

printf("\nID ordine: ");

if(scanf("%lld", &id\_ordine)<1){

printf("Errore nell'acquisire indice ordine\n");

run\_as\_pizzaiolo(conn);

}

flush\_stdin();

param[0].buffer\_type = MYSQL\_TYPE\_LONGLONG;

param[0].buffer = &id\_ordine;

param[0].buffer\_length = sizeof(id\_ordine);

if (mysql\_stmt\_bind\_param(prepared\_stmt, param) != 0) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Could not bind parameters for prendi\_in\_carico\_ordine\_pizza\_plus\n", true);

}

// Run procedure

if (mysql\_stmt\_execute(prepared\_stmt) != 0) {

print\_stmt\_error (prepared\_stmt, "Errore nel prendere in carico ordine pizza plus.\n");

return;

} else {

printf("Ordine pizza plus preso in carico correttamente\n");

}

mysql\_stmt\_close(prepared\_stmt);

espleta\_ordine\_pizza\_plus(conn,id\_ordine);

}

void run\_as\_pizzaiolo(MYSQL \*conn){

char options[6] = {'1','2','3','4','5'};

char op;

printf("Passo al ruolo di pizzaiolo...\n");

if(!parse\_config("users/pizzaiolo.json", &conf)) {

fprintf(stderr, "Unable to load pizzaiolo configuration\n");

exit(EXIT\_FAILURE);

}

if(mysql\_change\_user(conn, conf.db\_username, conf.db\_password, conf.database)) {

fprintf(stderr, "mysql\_change\_user() failed\n");

exit(EXIT\_FAILURE);

}

while(true) {

printf("\033[2J\033[H");

printf("\*\*\* Cosa posso fare per te? \*\*\*\n\n");

printf("1) Prendi in carico ordine pizza\n");

printf("2) Prendi in carico ordine pizzaplus\n");

printf("3) Visualizza ordini pizza da espletare\n");

printf("4) Visualizza ordini pizza plus da espletare\n");

printf("5) Quit\n");

op = multiChoice("Seleziona un opzione", options, 5);

switch(op) {

case '1':

prendi\_in\_carico\_ordine\_pizza(conn);

break;

case '2':

prendi\_in\_carico\_ordine\_pizza\_plus(conn);

break;

case '3':

visualizza\_ordini\_pizza\_da\_espletare(conn);

break;

case '4':

visualizza\_ordini\_pizza\_plus\_da\_espletare(conn);

break;

case '5':

return;

default:

fprintf(stderr, "Invalid condition at %s:%d\n", \_\_FILE\_\_, \_\_LINE\_\_);

abort();

}

getchar();

}

}

* Client del barista

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <mysql/mysql.h>

#include "defines.h"

static void visualizza\_ordini\_bevanda\_da\_espletare(MYSQL \*conn){

MYSQL\_STMT \*prepared\_stmt;

if(!setup\_prepared\_stmt(&prepared\_stmt, "call visualizza\_ordini\_bevanda\_da\_espletare()", conn)) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Unable to initialize visualizza\_ordini\_bevanda\_da\_espletare statement\n", false);

}

// Run procedure

if (mysql\_stmt\_execute(prepared\_stmt) != 0) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Could not retrieve visualizza\_ordini\_bevanda\_da\_espletare\n", true);

}

// Dump the result set

dump\_result\_set(conn, prepared\_stmt, "\nLista delle bevande da espletare");

mysql\_stmt\_next\_result(prepared\_stmt);

mysql\_stmt\_close(prepared\_stmt);

}

static void espleta\_ordine\_bevanda(MYSQL \*conn,long long int ordine){

MYSQL\_STMT \*prepared\_stmt;

MYSQL\_BIND param[1];

char check[2];

// Prepare parameters

memset(param, 0, sizeof(param));

memset(check,'\0', sizeof(check));

param[0].buffer\_type = MYSQL\_TYPE\_LONGLONG;

param[0].buffer = &ordine;

param[0].buffer\_length = sizeof(ordine);

check\_ok\_bevanda:

//espleta ordine, viene messa questa scanf in modo da poter prende in carico l'ordine ed espletarlo solo

//quando e' stato effettivamente fatto, evitando al barman di dover ricordare l'id dell'ordine

printf("Digitare ok per espletare l'ordine\n");

if(scanf("%s", check)<1){

printf("Errore nella conferma\n");

flush\_stdin();

goto check\_ok\_bevanda;

}

flush\_stdin();

if(strcmp(check,"ok")!=0){

printf("Per favore digita ok\n");

goto check\_ok\_bevanda;

}

// Prepare stored procedure call

if(!setup\_prepared\_stmt(&prepared\_stmt, "call espleta\_ordine\_bevanda(?)", conn)) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Unable to initialize espleta\_ordine\_bevanda statement\n", false);

}

// non vi e' reset perche i parametri sono gli stessi

if (mysql\_stmt\_bind\_param(prepared\_stmt, param) != 0) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Could not bind parameters for espleta\_ordine\_bevanda\n", true);

}

// Run procedure

if (mysql\_stmt\_execute(prepared\_stmt) != 0) {

print\_stmt\_error(prepared\_stmt, "Errore nell'espletare ordine bevanda.\n");

} else {

printf("Ordine bevanda espletato correttamente\n");

}

mysql\_stmt\_close(prepared\_stmt);

}

static void prendi\_in\_carico\_ordine\_bevanda(MYSQL \*conn) {

MYSQL\_STMT \*prepared\_stmt;

MYSQL\_BIND param[1];

long long int id\_ordine;

// Prepare stored procedure call

if(!setup\_prepared\_stmt(&prepared\_stmt, "call prendi\_in\_carico\_ordine\_bevanda(?)", conn)) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Unable to initialize prendi\_in\_carico\_ordine\_bevanda statement\n", false);

}

// Prepare parameters

memset(param, 0, sizeof(param));

printf("\nDigitare l'id ordine: ");

if(scanf("%lld", &id\_ordine)<1){

printf("Errore nell'acquisire indice ordine\n");

flush\_stdin();

run\_as\_barman(conn);

}

flush\_stdin();

param[0].buffer\_type = MYSQL\_TYPE\_LONGLONG;

param[0].buffer = &id\_ordine;

param[0].buffer\_length = sizeof(id\_ordine);

if (mysql\_stmt\_bind\_param(prepared\_stmt, param) != 0) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Could not bind parameters for prendi\_in\_carico\_ordine\_bevanda\n", true);

}

// Run procedure

if (mysql\_stmt\_execute(prepared\_stmt) != 0) {

print\_stmt\_error (prepared\_stmt, "Errore nel prendere in carico ordine bevanda.\n");

} else {

printf("Ordine bevanda preso in carico correttamente\n");

}

mysql\_stmt\_close(prepared\_stmt);

espleta\_ordine\_bevanda(conn,id\_ordine);

}

void run\_as\_barman(MYSQL \*conn){

char options[4] = {'1','2','3'};

char op;

printf("Passo al ruolo di barman...\n");

if(!parse\_config("users/barman.json", &conf)) {

fprintf(stderr, "Unable to load barman configuration\n");

exit(EXIT\_FAILURE);

}

if(mysql\_change\_user(conn, conf.db\_username, conf.db\_password, conf.database)) {

fprintf(stderr, "mysql\_change\_user() failed\n");

exit(EXIT\_FAILURE);

}

while(true) {

printf("\033[2J\033[H");

printf("\*\*\* Cosa posso fare per te? \*\*\*\n\n");

printf("1) Prendi in carico ordine bevanda\n");

printf("2) Visualizza ordini bevanda da espletare\n");

printf("3) Quit\n");

op = multiChoice("Seleziona un opzione", options, 3);

switch(op) {

case '1':

prendi\_in\_carico\_ordine\_bevanda(conn);

break;

case '2':

visualizza\_ordini\_bevanda\_da\_espletare(conn);

break;

case '3':

return;

default:

fprintf(stderr, "Invalid condition at %s:%d\n", \_\_FILE\_\_, \_\_LINE\_\_);

abort();

}

getchar();

}

}

* Client del cameriere

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <mysql/mysql.h>

#include "defines.h"

static void visualizza\_menu\_pizze(MYSQL \*conn){

MYSQL\_STMT \*prepared\_stmt;

//Pizza

if(!setup\_prepared\_stmt(&prepared\_stmt, "call visualizza\_menu\_pizze()", conn)) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Unable to initialize visualizza\_menu\_pizze statement\n", false);

}

// Run procedure

if (mysql\_stmt\_execute(prepared\_stmt) != 0) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Could not retrieve visualizza\_menu\_pizze\n", true);

}

// Dump the result set

dump\_result\_set(conn, prepared\_stmt, "\n Menu' pizze");

mysql\_stmt\_next\_result(prepared\_stmt);

mysql\_stmt\_close(prepared\_stmt);

}

static void visualizza\_menu\_ingredienti(MYSQL \*conn){

MYSQL\_STMT \*prepared\_stmt;

if(!setup\_prepared\_stmt(&prepared\_stmt, "call visualizza\_menu\_ingredienti()", conn)) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Unable to initialize visualizza\_menu\_ingredienti statement\n", false);

}

// Run procedure

if (mysql\_stmt\_execute(prepared\_stmt) != 0) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Could not retrieve visualizza\_menu\_ingredienti\n", true);

}

// Dump the result set

dump\_result\_set(conn, prepared\_stmt, "\n Menu' ingredienti");

mysql\_stmt\_next\_result(prepared\_stmt);

mysql\_stmt\_close(prepared\_stmt);

}

static void visualizza\_menu\_bevande(MYSQL \*conn){

MYSQL\_STMT \*prepared\_stmt;

//Bevanda

if(!setup\_prepared\_stmt(&prepared\_stmt, "call visualizza\_menu\_bevande()", conn)) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Unable to initialize visualizza\_menu\_bevande statement\n", false);

}

// Run procedure

if (mysql\_stmt\_execute(prepared\_stmt) != 0) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Could not retrieve visualizza\_menu\_bevande\n", true);

}

// Dump the result set

dump\_result\_set(conn, prepared\_stmt, "\n Menu' bevande");

mysql\_stmt\_next\_result(prepared\_stmt);

mysql\_stmt\_close(prepared\_stmt);

}

static void visualizza\_info\_tavoli\_associati(MYSQL \*conn) {

MYSQL\_STMT \*prepared\_stmt;

MYSQL\_BIND param[1];

int status;

int info=0;

if(!setup\_prepared\_stmt(&prepared\_stmt, "call visualizza\_info\_tavoli\_associati(?)", conn)) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Unable to initialize visualizza\_info\_tavoli\_associati statement\n", false);

}

// Prepare parameters

memset(param, 0, sizeof(param));

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

param[0].buffer = &conf.username;

param[0].buffer\_length = sizeof(conf.username);

if (mysql\_stmt\_bind\_param(prepared\_stmt, param) != 0) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Could not bind parameters for visualizza\_info\_tavoli\_associati\n", true);

}

// Run procedure

if (mysql\_stmt\_execute(prepared\_stmt) != 0) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Could not retrieve visualizza\_info\_tavoli\_associati\n", true);

goto out;

}

// We have multiple result sets here!

do {

// Skip OUT variables (although they are not present in the procedure...)

if(conn->server\_status & SERVER\_PS\_OUT\_PARAMS) {

goto next;

}

if(info==0) {

dump\_result\_set(conn,prepared\_stmt,"\nTavoli liberi:\n");

info++;

}

else if(info==1){

dump\_result\_set(conn,prepared\_stmt,"\nTavoli occupati:\n");

info++;

}

else{

dump\_result\_set(conn,prepared\_stmt,"\nTavoli serviti:\n");

info=0;

}

// more results? -1 = no, >0 = error, 0 = yes (keep looking)

next:

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

if (status > 0)

finish\_with\_stmt\_error(conn, prepared\_stmt, "Unexpected condition", true);

} while (status == 0);

out:

mysql\_stmt\_close(prepared\_stmt);

}

static void registra\_ordine\_pizza(MYSQL \*conn){

MYSQL\_STMT \*prepared\_stmt;

MYSQL\_BIND param[2];

char pizza[45];

int tavolo;

// Prepare parameters

memset(param, 0, sizeof(param));

memset(pizza,'\0', sizeof(pizza));

printf("Inserisci il tavolo:\n");

if(scanf("%d",&tavolo)<1){

printf("Errore nell'acquisire il tavolo\n");

flush\_stdin();

return;

}

flush\_stdin();

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

param[0].buffer = &tavolo;

param[0].buffer\_length = sizeof(tavolo);

printf("Inserisci la pizza:\n");

getInput(45,pizza,false);

param[1].buffer\_type = MYSQL\_TYPE\_STRING;

param[1].buffer = pizza;

param[1].buffer\_length = sizeof(pizza);

if(!setup\_prepared\_stmt(&prepared\_stmt, "call registra\_ordine\_pizza(?,?)", conn)) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Unable to initialize registra\_ordine\_pizza statement\n", false);

}

if (mysql\_stmt\_bind\_param(prepared\_stmt, param) != 0) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Could not bind parameters registra\_ordine\_pizza\n", true);

}

// Run procedure

if (mysql\_stmt\_execute(prepared\_stmt) != 0) {

print\_stmt\_error(prepared\_stmt, "Could not retrieve registra\_ordine\_pizza\n");

}else printf("Ordine pizza registrato correttamente\n");

mysql\_stmt\_close(prepared\_stmt);

}

static void registra\_ordine\_bevanda(MYSQL \*conn){

MYSQL\_STMT \*prepared\_stmt;

MYSQL\_BIND param[2];

char bevanda[45];

int tavolo;

// Prepare parameters

memset(param, 0, sizeof(param));

memset(bevanda,'\0', sizeof(bevanda));

printf("Inserisci il tavolo:\n");

if(scanf("%d",&tavolo)<1){

printf("Errore nell'acquisire il tavolo\n");

flush\_stdin();

return;

}

flush\_stdin();

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

param[0].buffer = &tavolo;

param[0].buffer\_length = sizeof(tavolo);

printf("Inserisci la bevanda:\n");

getInput(45,bevanda,false);

param[1].buffer\_type = MYSQL\_TYPE\_STRING;

param[1].buffer = bevanda;

param[1].buffer\_length = sizeof(bevanda);

if(!setup\_prepared\_stmt(&prepared\_stmt, "call registra\_ordine\_bevanda(?,?)", conn)) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Unable to initialize registra\_ordine\_bevanda statement\n", false);

}

if (mysql\_stmt\_bind\_param(prepared\_stmt, param) != 0) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Could not bind parameters registra\_ordine\_bevanda\n", true);

}

// Run procedure

if (mysql\_stmt\_execute(prepared\_stmt) != 0) {

print\_stmt\_error(prepared\_stmt, "Could not retrieve registra\_ordine\_bevanda\n");

}else printf("Ordine bevanda registrato correttamente\n");

mysql\_stmt\_close(prepared\_stmt);

}

static void registra\_ordine\_pizza\_plus(MYSQL \*conn){

MYSQL\_STMT \*prepared\_stmt;

MYSQL\_BIND param[7];

char pizza[45];

char ing[5][45];

int tavolo;

my\_bool is\_null;

// Prepare parameters

memset(param, 0, sizeof(param));

memset(pizza,'\0', sizeof(pizza));

memset(ing[0],'\0', sizeof(ing[0]));

memset(ing[1],'\0', sizeof(ing[1]));

memset(ing[2],'\0', sizeof(ing[2]));

memset(ing[3],'\0', sizeof(ing[3]));

memset(ing[4],'\0', sizeof(ing[4]));

printf("Inserisci il tavolo:\n");

if(scanf("%d",&tavolo)<1){

printf("Errore nell'acquisire il tavolo\n");

flush\_stdin();

return;

}

flush\_stdin();

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

param[0].buffer = &tavolo;

param[0].buffer\_length = sizeof(tavolo);

printf("Inserisci la pizza da usare come base:\n");

getInput(45,pizza,false);

param[1].buffer\_type = MYSQL\_TYPE\_STRING;

param[1].buffer = pizza;

param[1].buffer\_length = sizeof(pizza);

int i=0;

bool stop=false;

for(i=0;i<5;i++){

if(stop==false){

printf("Inserire ingrediente %d oppure digitare stop se non si devono inserire altri ingredienti\n",i+1);

getInput(45,ing[i],false);

if(strcmp(ing[i],"stop")==0){

param[i+2].is\_null=&is\_null;

stop=true;

continue;

}

param[i+2].buffer\_type = MYSQL\_TYPE\_STRING;

param[i+2].buffer = ing[i];

param[i+2].buffer\_length = sizeof(ing[i]);

}

else param[i+2].is\_null=&is\_null;

}

if(!setup\_prepared\_stmt(&prepared\_stmt, "call registra\_ordine\_pizza\_plus(?,?,?,?,?,?,?)", conn)) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Unable to initialize registra\_ordine\_pizza\_plus statement\n", false);

}

if (mysql\_stmt\_bind\_param(prepared\_stmt, param) != 0) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Could not bind parameters registra\_ordine\_pizza\_plus\n", true);

}

// Run procedure

if (mysql\_stmt\_execute(prepared\_stmt) != 0) {

print\_stmt\_error(prepared\_stmt, "Could not retrieve registra\_ordine\_pizza\_plus\n");

}else printf("Ordine pizza plus registrato correttamente\n");

mysql\_stmt\_close(prepared\_stmt);

}

static void registra\_ordine(MYSQL \*conn) {

char options[8] = {'1','2','3','4','5','6','7'};

char op;

while(true) {

printf("\033[2J\033[H");

printf("\*\*\* Cosa posso fare per te? \*\*\*\n\n");

printf("1) Registra ordine pizza\n");

printf("2) Registra ordine pizza plus\n");

printf("3) Registra ordine bevanda\n");

printf("4) Visualizza menu pizze\n");

printf("5) Visualizza menu bevande\n");

printf("6) Visualizza menu ingredienti\n");

printf("7) Quit\n");

op = multiChoice("Seleziona un opzione", options, 7);

switch(op) {

case '1':

registra\_ordine\_pizza(conn);

break;

case '2':

registra\_ordine\_pizza\_plus(conn);

break;

case '3':

registra\_ordine\_bevanda(conn);

break;

case '4':

visualizza\_menu\_pizze(conn);

break;

case '5':

visualizza\_menu\_bevande(conn);

break;

case '6':

visualizza\_menu\_ingredienti(conn);

break;

case '7':

return;

default:

fprintf(stderr, "Invalid condition at %s:%d\n", \_\_FILE\_\_, \_\_LINE\_\_);

abort();

}

getchar();

}

}

static void visualizza\_ordini\_espletati\_pizza(MYSQL \*conn){

MYSQL\_STMT \*prepared\_stmt;

MYSQL\_BIND param[1];

//Pizza

if(!setup\_prepared\_stmt(&prepared\_stmt, "call visualizza\_ordini\_espletati\_pizza(?)", conn)) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Unable to initialize visualizza\_ordini\_espletati\_pizza statement\n", false);

}

// Prepare parameters

memset(param, 0, sizeof(param));

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

param[0].buffer = &conf.username;

param[0].buffer\_length = sizeof(conf.username);

if (mysql\_stmt\_bind\_param(prepared\_stmt, param) != 0) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Could not bind parameters for visualizza\_ordini\_espletati\_pizza\n", true);

}

// Run procedure

if (mysql\_stmt\_execute(prepared\_stmt) != 0) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Could not retrieve visualizza\_ordini\_espletati\_pizza\n", true);

}

// Dump the result set

dump\_result\_set(conn, prepared\_stmt, "\nLista ordini pizza espletati");

mysql\_stmt\_next\_result(prepared\_stmt);

mysql\_stmt\_close(prepared\_stmt);

}

static void visualizza\_ordini\_espletati\_bevanda(MYSQL \*conn){

MYSQL\_STMT \*prepared\_stmt;

MYSQL\_BIND param[1];

if(!setup\_prepared\_stmt(&prepared\_stmt, "call visualizza\_ordini\_espletati\_bevanda(?)", conn)) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Unable to initialize visualizza\_ordini\_espletati\_bevanda statement\n", false);

}

// Prepare parameters

memset(param, 0, sizeof(param));

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

param[0].buffer = &conf.username;

param[0].buffer\_length = sizeof(conf.username);

if (mysql\_stmt\_bind\_param(prepared\_stmt, param) != 0) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Could not bind parameters for visualizza\_ordini\_espletati\_bevanda\n", true);

}

// Run procedure

if (mysql\_stmt\_execute(prepared\_stmt) != 0) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Could not retrieve visualizza\_ordini\_espletati\_bevanda\n", true);

}

// Dump the result set

dump\_result\_set(conn, prepared\_stmt, "\nLista ordini bevanda espletati");

mysql\_stmt\_next\_result(prepared\_stmt);

mysql\_stmt\_close(prepared\_stmt);

}

static void visualizza\_ordini\_espletati\_pizza\_plus(MYSQL \*conn){

MYSQL\_STMT \*prepared\_stmt;

MYSQL\_BIND param[1];

if(!setup\_prepared\_stmt(&prepared\_stmt, "call visualizza\_ordini\_espletati\_pizza\_plus(?)", conn)) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Unable to initialize visualizza\_ordini\_espletati\_pizza\_plus statement\n", false);

}

// Prepare parameters

memset(param, 0, sizeof(param));

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

param[0].buffer = &conf.username;

param[0].buffer\_length = sizeof(conf.username);

if (mysql\_stmt\_bind\_param(prepared\_stmt, param) != 0) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Could not bind parameters for visualizza\_ordini\_espletati\_pizza\_plus\n", true);

}

// Run procedure

if (mysql\_stmt\_execute(prepared\_stmt) != 0) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Could not retrieve visualizza\_ordini\_espletati\_pizza\_plus\n", true);

}

// Dump the result set

dump\_result\_set(conn, prepared\_stmt, "\nLista ordini pizza plus espletati");

mysql\_stmt\_next\_result(prepared\_stmt);

mysql\_stmt\_close(prepared\_stmt);

}

static void consegna\_ordine\_pizza(MYSQL \*conn){

MYSQL\_STMT \*prepared\_stmt;

MYSQL\_BIND param[1];

long long int id\_ordine;

visualizza\_ordini\_espletati\_pizza(conn);

// Prepare stored procedure call

if(!setup\_prepared\_stmt(&prepared\_stmt, "call consegna\_ordine\_pizza(?)", conn)) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Unable to initialize consegna\_ordine\_pizza statement\n", false);

}

// Prepare parameters

memset(param, 0, sizeof(param));

printf("\nID ordine: ");

if(scanf("%lld", &id\_ordine)<1){

printf("Errore nell'acquisire indice ordine\n");

flush\_stdin();

return;

}

flush\_stdin();

param[0].buffer\_type = MYSQL\_TYPE\_LONGLONG;

param[0].buffer = &id\_ordine;

param[0].buffer\_length = sizeof(id\_ordine);

if (mysql\_stmt\_bind\_param(prepared\_stmt, param) != 0) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Could not bind parameters for consegna\_ordine\_pizza\n", true);

}

// Run procedure

if (mysql\_stmt\_execute(prepared\_stmt) != 0) {

print\_stmt\_error (prepared\_stmt, "Errore nel consegnare ordine pizza.\n");

} else {

printf("Ordine pizza consegnato correttamente\n");

}

mysql\_stmt\_close(prepared\_stmt);

}

static void consegna\_ordine\_bevanda(MYSQL \*conn){

MYSQL\_STMT \*prepared\_stmt;

MYSQL\_BIND param[1];

long long int id\_ordine;

visualizza\_ordini\_espletati\_bevanda(conn);

// Prepare stored procedure call

if(!setup\_prepared\_stmt(&prepared\_stmt, "call consegna\_ordine\_bevanda(?)", conn)) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Unable to initialize consegna\_ordine\_bevanda statement\n", false);

}

// Prepare parameters

memset(param, 0, sizeof(param));

printf("\nID ordine: ");

if(scanf("%lld", &id\_ordine)<1){

printf("Errore nell'acquisire indice ordine\n");

flush\_stdin();

return;

}

flush\_stdin();

param[0].buffer\_type = MYSQL\_TYPE\_LONGLONG;

param[0].buffer = &id\_ordine;

param[0].buffer\_length = sizeof(id\_ordine);

if (mysql\_stmt\_bind\_param(prepared\_stmt, param) != 0) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Could not bind parameters for consegna\_ordine\_bevanda\n", true);

}

// Run procedure

if (mysql\_stmt\_execute(prepared\_stmt) != 0) {

print\_stmt\_error (prepared\_stmt, "Errore nel consegnare ordine bevanda.\n");

} else {

printf("Ordine bevanda consegnato correttamente\n");

}

mysql\_stmt\_close(prepared\_stmt);

}

static void consegna\_ordine\_pizza\_plus(MYSQL \*conn){

MYSQL\_STMT \*prepared\_stmt;

MYSQL\_BIND param[1];

long long int id\_ordine;

visualizza\_ordini\_espletati\_pizza\_plus(conn);

// Prepare stored procedure call

if(!setup\_prepared\_stmt(&prepared\_stmt, "call consegna\_ordine\_pizza\_plus(?)", conn)) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Unable to initialize consegna\_ordine\_pizza\_plus statement\n", false);

}

// Prepare parameters

memset(param, 0, sizeof(param));

printf("\nID ordine: ");

if(scanf("%lld", &id\_ordine)<1){

printf("Errore nell'acquisire indice ordine\n");

flush\_stdin();

return;

}

flush\_stdin();

param[0].buffer\_type = MYSQL\_TYPE\_LONGLONG;

param[0].buffer = &id\_ordine;

param[0].buffer\_length = sizeof(id\_ordine);

if (mysql\_stmt\_bind\_param(prepared\_stmt, param) != 0) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Could not bind parameters for consegnaordine\_pizza\_plus\n", true);

}

// Run procedure

if (mysql\_stmt\_execute(prepared\_stmt) != 0) {

print\_stmt\_error (prepared\_stmt, "Errore nel consegnare ordine pizza plus.\n");

} else {

printf("Ordine pizza plus consegnato correttamente\n");

}

mysql\_stmt\_close(prepared\_stmt);

}

static void consegna\_ordine(MYSQL \*conn) {

char options[7] = {'1','2','3','4'};

char op;

while(true) {

printf("\033[2J\033[H");

printf("\*\*\* Cosa posso fare per te? \*\*\*\n\n");

printf("1) Consegna ordine pizza\n");

printf("2) Consegna ordine pizza plus\n");

printf("3) Consegna ordine bevanda\n");

printf("4) Quit\n");

op = multiChoice("Seleziona un opzione", options, 4);

switch(op) {

case '1':

consegna\_ordine\_pizza(conn);

break;

case '2':

consegna\_ordine\_pizza\_plus(conn);

break;

case '3':

consegna\_ordine\_bevanda(conn);

break;

case '4':

return;

default:

fprintf(stderr, "Invalid condition at %s:%d\n", \_\_FILE\_\_, \_\_LINE\_\_);

abort();

}

getchar();

}

}

void run\_as\_cameriere(MYSQL \*conn){

char options[8] = {'1','2','3','4','5','6','7'};

char op;

printf("Passo al ruolo di cameriere...\n");

if(!parse\_config("users/cameriere.json", &conf)) {

fprintf(stderr, "Unable to load cameriere configuration\n");

exit(EXIT\_FAILURE);

}

if(mysql\_change\_user(conn, conf.db\_username, conf.db\_password, conf.database)) {

fprintf(stderr, "mysql\_change\_user() failed\n");

exit(EXIT\_FAILURE);

}

while(true) {

printf("\033[2J\033[H");

printf("\*\*\* Cosa posso fare per te? \*\*\*\n\n");

printf("1) Visualizza tavoli associati\n");

printf("2) Registra ordine\n");

printf("3) Visualizza ordini pizza espletati\n");

printf("4) Visualizza ordini pizza plus espletati\n");

printf("5) Visualizza ordini bevanda espletati\n");

printf("6) Consegna ordine\n");

printf("7) Quit\n");

op = multiChoice("Seleziona un opzione", options, 7);

switch(op) {

case '1':

visualizza\_info\_tavoli\_associati(conn);

break;

case '2':

registra\_ordine(conn);

break;

case '3':

visualizza\_ordini\_espletati\_pizza(conn);

break;

case '4':

visualizza\_ordini\_espletati\_pizza\_plus(conn);

break;

case '5':

visualizza\_ordini\_espletati\_bevanda(conn);

break;

case '6':

consegna\_ordine(conn);

break;

case '7':

return;

default:

fprintf(stderr, "Invalid condition at %s:%d\n", \_\_FILE\_\_, \_\_LINE\_\_);

abort();

}

getchar();

}

}

* Client del manager

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <mysql/mysql.h>

#include "defines.h"

static void stampa\_scontrino\_tavolo(MYSQL \*conn){

MYSQL\_STMT \*prepared\_stmt;

MYSQL\_BIND param[1];

int tavolo;

int status;

bool first = true;

if(!setup\_prepared\_stmt(&prepared\_stmt, "call stampa\_scontrino\_tavolo(?)", conn)) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Unable to initialize stampa\_scontrino\_tavolo\n", false);

}

// Prepare parameters

memset(param, 0, sizeof(param));

printf("Inserire tavolo:\n");

if(scanf("%d",&tavolo)<1){

printf("Errore inserimento tavolo\n");

flush\_stdin();

return;

}

flush\_stdin();

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

param[0].buffer = &tavolo;

param[0].buffer\_length = sizeof(tavolo);

if (mysql\_stmt\_bind\_param(prepared\_stmt, param) != 0) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Could not bind parameters for stampa\_scontrino\_tavolo\n", true);

}

// Run procedure

if (mysql\_stmt\_execute(prepared\_stmt) != 0) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Could not retrieve stampa\_scontrino\_tavolo\n", true);

goto out;

}

// We have multiple result sets here!

do {

// Skip OUT variables (although they are not present in the procedure...)

if(conn->server\_status & SERVER\_PS\_OUT\_PARAMS) {

goto next;

}

if(first) {

first = false;

dump\_result\_set(conn, prepared\_stmt, "Info scontrino:\n");

} else {

dump\_result\_set(conn, prepared\_stmt, "Dettagli scontrino:\n");

}

// more results? -1 = no, >0 = error, 0 = yes (keep looking)

next:

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

if (status > 0)

finish\_with\_stmt\_error(conn, prepared\_stmt, "Unexpected condition", true);

} while (status == 0);

out:

mysql\_stmt\_close(prepared\_stmt);

}

static void visualizza\_tavoli(MYSQL \*conn){

MYSQL\_STMT \*prepared\_stmt;

//Visualizzazione tavoli

if(!setup\_prepared\_stmt(&prepared\_stmt, "call visualizza\_tavoli()", conn)) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Unable to initialize visualizza\_tavoli statement\n", false);

}

// Run procedure

if (mysql\_stmt\_execute(prepared\_stmt) != 0) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Could not retrieve visualizza\_tavoli\n", true);

}

// Dump the result set

dump\_result\_set(conn, prepared\_stmt, "\n Lista tavoli");

mysql\_stmt\_next\_result(prepared\_stmt);

mysql\_stmt\_close(prepared\_stmt);

}

static void registra\_cliente(MYSQL \*conn) {

MYSQL\_STMT \*prepared\_stmt;

MYSQL\_BIND param[4];

MYSQL\_BIND param1[2];

char cf[16];

char nome[45];

char cognome[45];

int commensali;

int tavolo;

//Registrazione cliente

if(!setup\_prepared\_stmt(&prepared\_stmt, "call registra\_cliente(?,?,?,?)", conn)) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Unable to initialize registra\_cliente statement\n", false);

}

// Prepare parameters

memset(param, 0, sizeof(param));

memset(cf,'\0', sizeof(cf));

memset(nome,'\0', sizeof(nome));

memset(cognome,'\0', sizeof(cognome));

printf("Inserire CF cliente:\n");

getInput(16,cf,false);

printf("Inserire nome cliente:\n");

getInput(45,nome,false);

printf("Inserire cognome cliente:\n");

getInput(45,cognome,false);

printf("Inserire numero commensali:\n");

if(scanf("%d",&commensali)<1){

printf("Errore inserimento numero commensali\n");

flush\_stdin();

return;

}

flush\_stdin();

param[0].buffer\_type = MYSQL\_TYPE\_STRING;

param[0].buffer = cf;

param[0].buffer\_length = sizeof(cf);

param[1].buffer\_type = MYSQL\_TYPE\_STRING;

param[1].buffer = nome;

param[1].buffer\_length = sizeof(nome);

param[2].buffer\_type = MYSQL\_TYPE\_STRING;

param[2].buffer = cognome;

param[2].buffer\_length = sizeof(cognome);

param[3].buffer\_type = MYSQL\_TYPE\_LONG;

param[3].buffer = &commensali;

param[3].buffer\_length = sizeof(commensali);

if (mysql\_stmt\_bind\_param(prepared\_stmt, param) != 0) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Could not bind parameters for registra\_cliente\n", true);

}

// Run procedure

if (mysql\_stmt\_execute(prepared\_stmt) != 0) {

print\_stmt\_error(prepared\_stmt, "Could not run register client\n");

} else printf("Cliente registrato correttamente\n");

mysql\_stmt\_close(prepared\_stmt);

visualizza\_tavoli(conn);

//Registrazione cliente

if(!setup\_prepared\_stmt(&prepared\_stmt, "call assegna\_tavolo\_a\_cliente(?,?)", conn)) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Unable to initialize assegna\_tavolo\_a\_cliente\n", false);

}

// Prepare parameters

memset(param1, 0, sizeof(param1));

printf("\n\nInserire tavolo da assegnare al cliente:\n");

if(scanf("%d",&tavolo)<1){

printf("Errore inserimento tavolo\n");

flush\_stdin();

return;

}

flush\_stdin();

param1[1].buffer\_type = MYSQL\_TYPE\_LONG;

param1[1].buffer = &tavolo;

param1[1].buffer\_length = sizeof(tavolo);

//For table assignment

param1[0].buffer\_type = MYSQL\_TYPE\_STRING;

param1[0].buffer = cf;

param1[0].buffer\_length = sizeof(cf);

if (mysql\_stmt\_bind\_param(prepared\_stmt, param1) != 0) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Could not bind parameters for assegna\_tavolo\_a\_cliente\n", true);

}

// Run procedure

if (mysql\_stmt\_execute(prepared\_stmt) != 0) {

print\_stmt\_error(prepared\_stmt, "Could not assign table\n");

} else printf("Tavolo assegnato correttamente\n");

mysql\_stmt\_close(prepared\_stmt);

}

static void visualizza\_entrate\_giorno(MYSQL \*conn){

MYSQL\_STMT \*prepared\_stmt;

MYSQL\_BIND param[2];

MYSQL\_TIME data;

float entrate;

if(!setup\_prepared\_stmt(&prepared\_stmt, "call visualizza\_entrate\_giorno(?,?)", conn)) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Unable to initialize visualizza\_entrate\_giorno\n", false);

}

// Prepare parameters

memset(param, 0, sizeof(param));

printf("Inserire anno:\n");

if(scanf("%4d",&data.year)<1){

printf("Errore inserimento anno\n");

flush\_stdin();

return;

}

flush\_stdin();

printf("Inserire mese:\n");

if(scanf("%2d",&data.month)<1){

printf("Errore inserimento mese\n");

flush\_stdin();

return;

}

flush\_stdin();

printf("Inserire giorno:\n");

if(scanf("%2d",&data.day)<1){

printf("Errore inserimento giorno\n");

flush\_stdin();

return;

}

flush\_stdin();

param[0].buffer\_type = MYSQL\_TYPE\_DATE;

param[0].buffer = (char \*)&data;

param[0].buffer\_length = sizeof(data);

param[1].buffer\_type = MYSQL\_TYPE\_FLOAT; // OUT

param[1].buffer = &entrate;

param[1].buffer\_length = sizeof(entrate);

if (mysql\_stmt\_bind\_param(prepared\_stmt, param) != 0) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Could not bind parameters for visualizza\_entrate\_giorno\n", true);

}

// Run procedure

if (mysql\_stmt\_execute(prepared\_stmt) != 0) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Could not retrieve visualizza\_entrate\_giorno\n", true);

}

// Prepare output parameters

memset(param, 0, sizeof(param));

param[0].buffer\_type = MYSQL\_TYPE\_FLOAT; // OUT

param[0].buffer = &entrate;

param[0].buffer\_length = sizeof(entrate);

if(mysql\_stmt\_bind\_result(prepared\_stmt, param)) {

print\_stmt\_error(prepared\_stmt, "Could not retrieve output parameter");

return;

}

// Retrieve output parameter

if(mysql\_stmt\_fetch(prepared\_stmt)) {

print\_stmt\_error(prepared\_stmt, "Could not buffer results");

return;

}

printf("Entrate del %02d/%02d/%4d: %f", data.day,data.month, data.year,entrate);

mysql\_stmt\_close(prepared\_stmt);

}

static void visualizza\_entrate\_mese(MYSQL \*conn){

MYSQL\_STMT \*prepared\_stmt;

MYSQL\_BIND param[2];

MYSQL\_TIME data;

data.day=01;

float entrate=0;

if(!setup\_prepared\_stmt(&prepared\_stmt, "call visualizza\_entrate\_mese(?,?)", conn)) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Unable to initialize visualizza\_entrate\_mese\n", false);

}

// Prepare parameters

memset(param, 0, sizeof(param));

printf("Inserire anno:\n");

if(scanf("%4d",&data.year)<1){

printf("Errore inserimento anno\n");

flush\_stdin();

return;

}

flush\_stdin();

printf("Inserire mese:\n");

if(scanf("%2d",&data.month)<1){

printf("Errore inserimento mese\n");

flush\_stdin();

return;

}

flush\_stdin();

param[0].buffer\_type = MYSQL\_TYPE\_DATE;

param[0].buffer = (char \*)&data;

param[0].buffer\_length = sizeof(data);

param[1].buffer\_type = MYSQL\_TYPE\_FLOAT;

param[1].buffer = &entrate;

param[1].buffer\_length = sizeof(entrate);

if (mysql\_stmt\_bind\_param(prepared\_stmt, param) != 0) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Could not bind parameters for visualizza\_entrate\_mese\n", true);

}

// Run procedure

if (mysql\_stmt\_execute(prepared\_stmt) != 0) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Could not retrieve visualizza\_entrate\_mese\n", true);

}

// Prepare output parameters

memset(param, 0, sizeof(param));

param[0].buffer\_type = MYSQL\_TYPE\_FLOAT; // OUT

param[0].buffer = &entrate;

param[0].buffer\_length = sizeof(entrate);

if(mysql\_stmt\_bind\_result(prepared\_stmt, param)) {

print\_stmt\_error(prepared\_stmt, "Could not retrieve output parameter");

return;

}

// Retrieve output parameter

if(mysql\_stmt\_fetch(prepared\_stmt)) {

print\_stmt\_error(prepared\_stmt, "Could not buffer results");

return;

}

printf("Entrate del %02d/%4d: %f", data.month, data.year,entrate);

mysql\_stmt\_close(prepared\_stmt);

}

static void visualizza\_menu\_pizze(MYSQL \*conn){

MYSQL\_STMT \*prepared\_stmt;

//Pizza

if(!setup\_prepared\_stmt(&prepared\_stmt, "call visualizza\_menu\_pizze()", conn)) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Unable to initialize visualizza\_menu\_pizze statement\n", false);

}

// Run procedure

if (mysql\_stmt\_execute(prepared\_stmt) != 0) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Could not retrieve visualizza\_menu\_pizze\n", true);

}

// Dump the result set

dump\_result\_set(conn, prepared\_stmt, "\n Menu' pizze");

mysql\_stmt\_next\_result(prepared\_stmt);

mysql\_stmt\_close(prepared\_stmt);

}

static void visualizza\_menu\_ingredienti(MYSQL \*conn){

MYSQL\_STMT \*prepared\_stmt;

if(!setup\_prepared\_stmt(&prepared\_stmt, "call visualizza\_menu\_ingredienti()", conn)) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Unable to initialize visualizza\_menu\_ingredienti statement\n", false);

}

// Run procedure

if (mysql\_stmt\_execute(prepared\_stmt) != 0) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Could not retrieve visualizza\_menu\_ingredienti\n", true);

}

// Dump the result set

dump\_result\_set(conn, prepared\_stmt, "\n Menu' ingredienti");

mysql\_stmt\_next\_result(prepared\_stmt);

mysql\_stmt\_close(prepared\_stmt);

}

static void visualizza\_menu\_bevande(MYSQL \*conn){

MYSQL\_STMT \*prepared\_stmt;

//Bevanda

if(!setup\_prepared\_stmt(&prepared\_stmt, "call visualizza\_menu\_bevande()", conn)) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Unable to initialize visualizza\_menu\_bevande statement\n", false);

}

// Run procedure

if (mysql\_stmt\_execute(prepared\_stmt) != 0) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Could not retrieve visualizza\_menu\_bevande\n", true);

}

// Dump the result set

dump\_result\_set(conn, prepared\_stmt, "\n Menu' bevande");

mysql\_stmt\_next\_result(prepared\_stmt);

mysql\_stmt\_close(prepared\_stmt);

}

static void aumenta\_scorte\_pizza(MYSQL \*conn){

MYSQL\_STMT \*prepared\_stmt;

MYSQL\_BIND param[2];

char pizza[45];

int scorte;

if(!setup\_prepared\_stmt(&prepared\_stmt, "call aumenta\_scorte\_pizza(?,?)", conn)) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Unable to initialize aumenta\_scorte\_pizza\n", false);

}

// Prepare parameters

memset(param, 0, sizeof(param));

memset(pizza,'\0', sizeof(pizza));

printf("Inserire pizza\n");

getInput(45,pizza,false);

printf("Inserire quantità:\n");

if(scanf("%d",&scorte)<1){

printf("Errore inserimento quantità\n");

flush\_stdin();

return;

}

flush\_stdin();

param[0].buffer\_type = MYSQL\_TYPE\_STRING;

param[0].buffer = pizza;

param[0].buffer\_length = sizeof(pizza);

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

param[1].buffer = &scorte;

param[1].buffer\_length = sizeof(scorte);

if (mysql\_stmt\_bind\_param(prepared\_stmt, param) != 0) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Could not bind parameters for aumenta\_scorte\_pizza\n", true);

}

// Run procedure

if (mysql\_stmt\_execute(prepared\_stmt) != 0) {

print\_stmt\_error(prepared\_stmt, "Could not retrieve refill pizza\n");

}else printf("Scorte aumentate correttamente\n");

mysql\_stmt\_close(prepared\_stmt);

}

static void aumenta\_scorte\_bevanda(MYSQL \*conn){

MYSQL\_STMT \*prepared\_stmt;

MYSQL\_BIND param[2];

char bevanda[45];

int scorte;

if(!setup\_prepared\_stmt(&prepared\_stmt, "call aumenta\_scorte\_bevanda(?,?)", conn)) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Unable to initialize aumenta\_scorte\_bevanda\n", false);

}

// Prepare parameters

memset(param, 0, sizeof(param));

memset(bevanda,'\0', sizeof(bevanda));

printf("Inserire bevanda\n");

getInput(45,bevanda,false);

printf("Inserire quantità:\n");

if(scanf("%d",&scorte)<1){

printf("Errore inserimento quantità\n");

flush\_stdin();

return;

}

flush\_stdin();

param[0].buffer\_type = MYSQL\_TYPE\_STRING;

param[0].buffer = bevanda;

param[0].buffer\_length = sizeof(bevanda);

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

param[1].buffer = &scorte;

param[1].buffer\_length = sizeof(scorte);

if (mysql\_stmt\_bind\_param(prepared\_stmt, param) != 0) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Could not bind parameters for aumenta\_scorte\_bevanda\n", true);

}

// Run procedure

if (mysql\_stmt\_execute(prepared\_stmt) != 0) {

print\_stmt\_error(prepared\_stmt, "Could not retrieve refill bevanda\n");

}else printf("Scorte aumentate correttamente\n");

mysql\_stmt\_close(prepared\_stmt);

}

static void aumenta\_scorte\_ingrediente(MYSQL \*conn){

MYSQL\_STMT \*prepared\_stmt;

MYSQL\_BIND param[2];

char ingrediente[45];

int scorte;

if(!setup\_prepared\_stmt(&prepared\_stmt, "call aumenta\_scorte\_ingrediente(?,?)", conn)) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Unable to initialize aumenta\_scorte\_ingrediente\n", false);

}

// Prepare parameters

memset(param, 0, sizeof(param));

memset(ingrediente,'\0', sizeof(ingrediente));

printf("Inserire ingrediente\n");

getInput(45,ingrediente,false);

printf("Inserire quantità:\n");

if(scanf("%d",&scorte)<1){

printf("Errore inserimento quantità\n");

flush\_stdin();

return;

}

flush\_stdin();

param[0].buffer\_type = MYSQL\_TYPE\_STRING;

param[0].buffer = ingrediente;

param[0].buffer\_length = sizeof(ingrediente);

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

param[1].buffer = &scorte;

param[1].buffer\_length = sizeof(scorte);

if (mysql\_stmt\_bind\_param(prepared\_stmt, param) != 0) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Could not bind parameters for aumenta\_scorte\_ingrediente\n", true);

}

// Run procedure

if (mysql\_stmt\_execute(prepared\_stmt) != 0) {

print\_stmt\_error(prepared\_stmt, "Could not retrieve refill ingrediente\n");

}else printf("Scorte aumentate correttamente\n");

mysql\_stmt\_close(prepared\_stmt);

}

static void aumenta\_scorte\_tutti\_prodotti(MYSQL \*conn){

MYSQL\_STMT \*prepared\_stmt;

MYSQL\_BIND param[1];

int scorte;

if(!setup\_prepared\_stmt(&prepared\_stmt, "call aumenta\_scorte\_tutti\_prodotti(?)", conn)) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Unable to initialize aumenta\_scorte\_tutti\_prodotti\n", false);

}

// Prepare parameters

memset(param, 0, sizeof(param));

printf("Inserire quantità:\n");

if(scanf("%d",&scorte)<1){

printf("Errore inserimento quantità\n");

flush\_stdin();

return;

}

flush\_stdin();

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

param[0].buffer = &scorte;

param[0].buffer\_length = sizeof(scorte);

if (mysql\_stmt\_bind\_param(prepared\_stmt, param) != 0) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Could not bind parameters for aumenta\_scorte\_tutti\_prodotti\n", true);

}

// Run procedure

if (mysql\_stmt\_execute(prepared\_stmt) != 0) {

print\_stmt\_error(prepared\_stmt, "Could not retrieve refill tutti\_prodotti\n");

}else printf("Scorte aumentate correttamente\n");

mysql\_stmt\_close(prepared\_stmt);

}

static void aumenta\_scorte\_prodotto(MYSQL \*conn){

char options[6] = {'1','2','3','4','5'};

char op;

while(true) {

printf("\033[2J\033[H");

printf("\*\*\* Cosa posso fare per te? \*\*\*\n\n");

printf("1) Aumenta scorte pizza\n");

printf("2) Aumenta scorte bevanda\n");

printf("3) Aumenta scorte ingrediente\n");

printf("4) Aumenta scorte di tutti i prodotti\n");

printf("5) Quit\n");

op = multiChoice("Seleziona un opzione", options, 5);

switch(op) {

case '1':

aumenta\_scorte\_pizza(conn);

break;

case '2':

aumenta\_scorte\_bevanda(conn);

break;

case '3':

aumenta\_scorte\_ingrediente(conn);

break;

case '4':

aumenta\_scorte\_tutti\_prodotti(conn);

break;

case '5':

return;

default:

fprintf(stderr, "Invalid condition at %s:%d\n", \_\_FILE\_\_, \_\_LINE\_\_);

abort();

}

getchar();

}

}

static void aggiungi\_pizza(MYSQL \*conn){

MYSQL\_STMT \*prepared\_stmt;

MYSQL\_BIND param[2];

char pizza[45];

float prezzo;

if(!setup\_prepared\_stmt(&prepared\_stmt, "call aggiungi\_pizza(?,?)", conn)) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Unable to initialize aggiungi\_pizza\n", false);

}

// Prepare parameters

memset(param, 0, sizeof(param));

memset(pizza,'\0', sizeof(pizza));

printf("Inserire pizza\n");

getInput(45,pizza,false);

printf("Inserire prezzo\n");

if(scanf("%f",&prezzo)<1){

printf("Inserimento prezzo fallito\n");

flush\_stdin();

return;

}

flush\_stdin();

param[0].buffer\_type = MYSQL\_TYPE\_STRING;

param[0].buffer = pizza;

param[0].buffer\_length = sizeof(pizza);

param[1].buffer\_type = MYSQL\_TYPE\_FLOAT;

param[1].buffer = &prezzo;

param[1].buffer\_length = sizeof(prezzo);

if (mysql\_stmt\_bind\_param(prepared\_stmt, param) != 0) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Could not bind parameters for aggiungi\_pizza\n", true);

}

// Run procedure

if (mysql\_stmt\_execute(prepared\_stmt) != 0) {

print\_stmt\_error(prepared\_stmt, "Could not add pizza\n");

}else printf("Pizza aggiunta correttamente\n");

mysql\_stmt\_close(prepared\_stmt);

}

static void aggiungi\_bevanda(MYSQL \*conn){

MYSQL\_STMT \*prepared\_stmt;

MYSQL\_BIND param[2];

char bevanda[45];

float prezzo;

if(!setup\_prepared\_stmt(&prepared\_stmt, "call aggiungi\_bevanda(?,?)", conn)) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Unable to initialize aggiungi\_bevanda\n", false);

}

// Prepare parameters

memset(param, 0, sizeof(param));

memset(bevanda,'\0', sizeof(bevanda));

printf("Inserire bevanda\n");

getInput(45,bevanda,false);

printf("Inserire prezzo\n");

if(scanf("%f",&prezzo)<1){

printf("Inserimento prezzo fallito\n");

flush\_stdin();

return;

}

flush\_stdin();

param[0].buffer\_type = MYSQL\_TYPE\_STRING;

param[0].buffer = bevanda;

param[0].buffer\_length = sizeof(bevanda);

param[1].buffer\_type = MYSQL\_TYPE\_FLOAT;

param[1].buffer = &prezzo;

param[1].buffer\_length = sizeof(prezzo);

if (mysql\_stmt\_bind\_param(prepared\_stmt, param) != 0) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Could not bind parameters for aggiungi\_bevanda\n", true);

}

// Run procedure

if (mysql\_stmt\_execute(prepared\_stmt) != 0) {

print\_stmt\_error(prepared\_stmt, "Could not add bevanda\n");

}else printf("Bevanda aggiunta correttamente\n");

mysql\_stmt\_close(prepared\_stmt);

}

static void aggiungi\_ingrediente(MYSQL \*conn){

MYSQL\_STMT \*prepared\_stmt;

MYSQL\_BIND param[2];

char ingrediente[45];

float prezzo;

if(!setup\_prepared\_stmt(&prepared\_stmt, "call aggiungi\_ingrediente(?,?)", conn)) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Unable to initialize aggiungi\_ingrediente\n", false);

}

// Prepare parameters

memset(param, 0, sizeof(param));

memset(ingrediente,'\0', sizeof(ingrediente));

printf("Inserire ingrediente\n");

getInput(45,ingrediente,false);

printf("Inserire prezzo\n");

if(scanf("%f",&prezzo)<1){

printf("Inserimento prezzo fallito\n");

flush\_stdin();

return;

}

flush\_stdin();

param[0].buffer\_type = MYSQL\_TYPE\_STRING;

param[0].buffer = ingrediente;

param[0].buffer\_length = sizeof(ingrediente);

param[1].buffer\_type = MYSQL\_TYPE\_FLOAT;

param[1].buffer = &prezzo;

param[1].buffer\_length = sizeof(prezzo);

if (mysql\_stmt\_bind\_param(prepared\_stmt, param) != 0) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Could not bind parameters for aggiungi\_ingrediente\n", true);

}

// Run procedure

if (mysql\_stmt\_execute(prepared\_stmt) != 0) {

print\_stmt\_error(prepared\_stmt, "Could not add ingrediente\n");

}else printf("Ingrediente aggiunta correttamente\n");

mysql\_stmt\_close(prepared\_stmt);

}

static void aggiungi\_prodotto(MYSQL \*conn){

char options[5] = {'1','2','3','4'};

char op;

while(true) {

printf("\033[2J\033[H");

printf("\*\*\* Cosa posso fare per te? \*\*\*\n\n");

printf("1) Aggiungi pizza\n");

printf("2) Aggiungi bevanda\n");

printf("3) Aggiungi ingrediente\n");

printf("4) Quit\n");

op = multiChoice("Seleziona un opzione", options, 4);

switch(op) {

case '1':

aggiungi\_pizza(conn);

break;

case '2':

aggiungi\_bevanda(conn);

break;

case '3':

aggiungi\_ingrediente(conn);

break;

case '4':

return;

default:

fprintf(stderr, "Invalid condition at %s:%d\n", \_\_FILE\_\_, \_\_LINE\_\_);

abort();

}

getchar();

}

}

static void rimuovi\_pizza(MYSQL \*conn){

MYSQL\_STMT \*prepared\_stmt;

MYSQL\_BIND param[1];

char pizza[45];

if(!setup\_prepared\_stmt(&prepared\_stmt, "call rimuovi\_pizza(?)", conn)) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Unable to initialize rimuovi\_pizza\n", false);

}

// Prepare parameters

memset(param, 0, sizeof(param));

memset(pizza,'\0', sizeof(pizza));

printf("Inserire pizza\n");

getInput(45,pizza,false);

param[0].buffer\_type = MYSQL\_TYPE\_STRING;

param[0].buffer = pizza;

param[0].buffer\_length = sizeof(pizza);

if (mysql\_stmt\_bind\_param(prepared\_stmt, param) != 0) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Could not bind parameters for rimuovi\_pizza\n", true);

}

// Run procedure

if (mysql\_stmt\_execute(prepared\_stmt) != 0) {

print\_stmt\_error(prepared\_stmt, "Could not remove pizza\n");

}else printf("Pizza rimossa correttamente\n");

mysql\_stmt\_close(prepared\_stmt);

}

static void rimuovi\_bevanda(MYSQL \*conn){

MYSQL\_STMT \*prepared\_stmt;

MYSQL\_BIND param[1];

char bevanda[45];

if(!setup\_prepared\_stmt(&prepared\_stmt, "call rimuovi\_bevanda(?)", conn)) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Unable to initialize rimuovi\_bevanda\n", false);

}

// Prepare parameters

memset(param, 0, sizeof(param));

memset(bevanda,'\0', sizeof(bevanda));

printf("Inserire bevanda\n");

getInput(45,bevanda,false);

param[0].buffer\_type = MYSQL\_TYPE\_STRING;

param[0].buffer = bevanda;

param[0].buffer\_length = sizeof(bevanda);

if (mysql\_stmt\_bind\_param(prepared\_stmt, param) != 0) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Could not bind parameters for rimuovi\_bevanda\n", true);

}

// Run procedure

if (mysql\_stmt\_execute(prepared\_stmt) != 0) {

print\_stmt\_error(prepared\_stmt, "Could not remove bevanda\n");

}else printf("Bevanda rimossa correttamente\n");

mysql\_stmt\_close(prepared\_stmt);

}

static void rimuovi\_ingrediente(MYSQL \*conn){

MYSQL\_STMT \*prepared\_stmt;

MYSQL\_BIND param[1];

char ingrediente[45];

if(!setup\_prepared\_stmt(&prepared\_stmt, "call rimuovi\_ingrediente(?)", conn)) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Unable to initialize rimuovi\_ingrediente\n", false);

}

// Prepare parameters

memset(param, 0, sizeof(param));

memset(ingrediente,'\0', sizeof(ingrediente));

printf("Inserire ingrediente\n");

getInput(45,ingrediente,false);

param[0].buffer\_type = MYSQL\_TYPE\_STRING;

param[0].buffer = ingrediente;

param[0].buffer\_length = sizeof(ingrediente);

if (mysql\_stmt\_bind\_param(prepared\_stmt, param) != 0) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Could not bind parameters for rimuovi\_ingrediente\n", true);

}

// Run procedure

if (mysql\_stmt\_execute(prepared\_stmt) != 0) {

print\_stmt\_error(prepared\_stmt, "Could not remove ingrediente\n");

}else printf("Ingrediente rimuovi correttamente\n");

mysql\_stmt\_close(prepared\_stmt);

}

static void rimuovi\_prodotto(MYSQL \*conn){

char options[5] = {'1','2','3','4'};

char op;

while(true) {

printf("\033[2J\033[H");

printf("\*\*\* Cosa posso fare per te? \*\*\*\n\n");

printf("1) Rimuovi pizza\n");

printf("2) Rimuovi bevanda\n");

printf("3) Rimuovi ingrediente\n");

printf("4) Quit\n");

op = multiChoice("Seleziona un opzione", options, 4);

switch(op) {

case '1':

rimuovi\_pizza(conn);

break;

case '2':

rimuovi\_bevanda(conn);

break;

case '3':

rimuovi\_ingrediente(conn);

break;

case '4':

return;

default:

fprintf(stderr, "Invalid condition at %s:%d\n", \_\_FILE\_\_, \_\_LINE\_\_);

abort();

}

getchar();

}

}

static void gestisci\_prodotti(MYSQL \*conn){

char options[8] = {'1','2','3','4','5','6','7'};

char op;

while(true) {

printf("\033[2J\033[H");

printf("\*\*\* Cosa posso fare per te? \*\*\*\n\n");

printf("1) Visualizza pizze\n");

printf("2) Visualizza bevande\n");

printf("3) Visualizza ingredienti\n");

printf("4) Aumenta scorte prodotto\n");

printf("5) Aggiungi prodotto\n");

printf("6) Rimuovi prodotto\n");

printf("7) Quit\n");

op = multiChoice("Seleziona un opzione", options, 7);

switch(op) {

case '1':

visualizza\_menu\_pizze(conn);

break;

case '2':

visualizza\_menu\_bevande(conn);

break;

case '3':

visualizza\_menu\_ingredienti(conn);

break;

case '4':

aumenta\_scorte\_prodotto(conn);

break;

case '5':

aggiungi\_prodotto(conn);

break;

case '6':

rimuovi\_prodotto(conn);

break;

case '7':

return;

default:

fprintf(stderr, "Invalid condition at %s:%d\n", \_\_FILE\_\_, \_\_LINE\_\_);

abort();

}

getchar();

}

}

static void aggiungi\_tavolo(MYSQL \*conn){

MYSQL\_STMT \*prepared\_stmt;

MYSQL\_BIND param[2];

int tavolo;

int posti;

if(!setup\_prepared\_stmt(&prepared\_stmt, "call aggiungi\_tavolo(?,?)", conn)) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Unable to initialize aggiungi\_tavolo\n", false);

}

// Prepare parameters

memset(param, 0, sizeof(param));

printf("Inserire numero tavolo\n");

if(scanf("%d",&tavolo)<1){

printf("Errore nell'acquisire numero tavolo\n");

flush\_stdin();

return;

}

flush\_stdin();

printf("Inserire numero posti tavolo\n");

if(scanf("%d",&posti)<1){

printf("Errore nell'acquisire numero posti tavolo\n");

flush\_stdin();

return;

}

flush\_stdin();

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

param[0].buffer = &tavolo;

param[0].buffer\_length = sizeof(tavolo);

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

param[1].buffer = &posti;

param[1].buffer\_length = sizeof(posti);

if (mysql\_stmt\_bind\_param(prepared\_stmt, param) != 0) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Could not bind parameters for aggiungi\_tavolo\n", true);

}

// Run procedure

if (mysql\_stmt\_execute(prepared\_stmt) != 0) {

print\_stmt\_error(prepared\_stmt, "Could not add tavolo\n");

}else printf("Tavolo aggiunto correttamente\n");

mysql\_stmt\_close(prepared\_stmt);

}

static void rimuovi\_tavolo(MYSQL \*conn){

MYSQL\_STMT \*prepared\_stmt;

MYSQL\_BIND param[1];

int tavolo;

if(!setup\_prepared\_stmt(&prepared\_stmt, "call rimuovi\_tavolo(?)", conn)) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Unable to initialize rimuovi\_tavolo\n", false);

}

// Prepare parameters

memset(param, 0, sizeof(param));

printf("Inserire numero tavolo\n");

if(scanf("%d",&tavolo)<1){

printf("Errore nell'acquisire numero tavolo\n");

flush\_stdin();

return;

}

flush\_stdin();

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

param[0].buffer = &tavolo;

param[0].buffer\_length = sizeof(tavolo);

if (mysql\_stmt\_bind\_param(prepared\_stmt, param) != 0) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Could not bind parameters for rimuovi\_tavolo\n", true);

}

// Run procedure

if (mysql\_stmt\_execute(prepared\_stmt) != 0) {

print\_stmt\_error(prepared\_stmt, "Could not remove tavolo\n");

}else printf("Tavolo rimosso correttamente\n");

mysql\_stmt\_close(prepared\_stmt);

}

static void visualizza\_camerieri(MYSQL \*conn){

MYSQL\_STMT \*prepared\_stmt;

//Camerieri

if(!setup\_prepared\_stmt(&prepared\_stmt, "call visualizza\_camerieri()", conn)) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Unable to initialize visualizza\_camerieri statement\n", false);

}

// Run procedure

if (mysql\_stmt\_execute(prepared\_stmt) != 0) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Could not retrieve visualizza\_camerieri\n", true);

}

// Dump the result set

dump\_result\_set(conn, prepared\_stmt, "\n Lista camerieri");

mysql\_stmt\_next\_result(prepared\_stmt);

mysql\_stmt\_close(prepared\_stmt);

}

static void visualizza\_assegnazioni\_tavoli\_camerieri(MYSQL \*conn){

MYSQL\_STMT \*prepared\_stmt;

//Camerieri

if(!setup\_prepared\_stmt(&prepared\_stmt, "call visualizza\_assegnazioni\_tavoli\_a\_camerieri()", conn)) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Unable to initialize visualizza\_assegnazioni\_tavoli\_a\_camerieri statement\n", false);

}

// Run procedure

if (mysql\_stmt\_execute(prepared\_stmt) != 0) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Could not retrieve visualizza\_assegnazioni\_tavoli\_a\_camerieri\n", true);

}

// Dump the result set

dump\_result\_set(conn, prepared\_stmt, "\n Lista assegnazioni");

mysql\_stmt\_next\_result(prepared\_stmt);

mysql\_stmt\_close(prepared\_stmt);

}

static void assegna\_tavolo\_a\_cameriere(MYSQL \*conn){

MYSQL\_STMT \*prepared\_stmt;

MYSQL\_BIND param[2];

int tavolo;

int cameriere;

//Assegnazione

if(!setup\_prepared\_stmt(&prepared\_stmt, "call assegna\_tavolo\_a\_cameriere(?,?)", conn)) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Unable to initialize assegna\_tavolo\_a\_cameriere\n", false);

}

// Prepare parameters

memset(param, 0, sizeof(param));

printf("Inserire numero tavolo\n");

if(scanf("%d",&tavolo)<1){

printf("Errore nell'acquisire numero tavolo\n");

flush\_stdin();

return;

}

flush\_stdin();

printf("Inserire matricola\n");

if(scanf("%d",&cameriere)<1){

printf("Errore nell'acquisire matricola\n");

flush\_stdin();

return;

}

flush\_stdin();

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

param[0].buffer = &cameriere;

param[0].buffer\_length = sizeof(cameriere);

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

param[1].buffer = &tavolo;

param[1].buffer\_length = sizeof(tavolo);

if (mysql\_stmt\_bind\_param(prepared\_stmt, param) != 0) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Could not bind parameters for assegna\_tavolo\_a\_cameriere\n", true);

}

// Run procedure

if (mysql\_stmt\_execute(prepared\_stmt) != 0) {

print\_stmt\_error(prepared\_stmt, "Could not assign table\n");

}else printf("Tavolo assegnato correttamente\n");

mysql\_stmt\_close(prepared\_stmt);

}

static void rimuovi\_tavolo\_a\_cameriere(MYSQL \*conn){

MYSQL\_STMT \*prepared\_stmt;

MYSQL\_BIND param[2];

int tavolo;

int cameriere;

//Rimozione assegnazione

if(!setup\_prepared\_stmt(&prepared\_stmt, "call rimuovi\_assegnazione\_tavolo(?,?)", conn)) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Unable to initialize rimuovi\_assegnazione\_tavolo\n", false);

}

// Prepare parameters

memset(param, 0, sizeof(param));

printf("Inserire numero tavolo\n");

if(scanf("%d",&tavolo)<1){

printf("Errore nell'acquisire numero tavolo\n");

flush\_stdin();

return;

}

flush\_stdin();

printf("Inserire matricola\n");

if(scanf("%d",&cameriere)<1){

printf("Errore nell'acquisire matricola\n");

flush\_stdin();

return;

}

flush\_stdin();

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

param[0].buffer = &cameriere;

param[0].buffer\_length = sizeof(cameriere);

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

param[1].buffer = &tavolo;

param[1].buffer\_length = sizeof(tavolo);

if (mysql\_stmt\_bind\_param(prepared\_stmt, param) != 0) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Could not bind parameters for rimuovi\_assegnazione\_tavolo\n", true);

}

// Run procedure

if (mysql\_stmt\_execute(prepared\_stmt) != 0) {

print\_stmt\_error(prepared\_stmt, "Could not remove table assignment\n");

}else printf("Assegnazione tavolo rimossa correttamente\n");

mysql\_stmt\_close(prepared\_stmt);

}

static void gestisci\_tavoli(MYSQL \*conn){

char options[9] = {'1','2','3','4','5','6','7','8'};

char op;

while(true) {

printf("\033[2J\033[H");

printf("\*\*\* Cosa posso fare per te? \*\*\*\n\n");

printf("1) Visualizza tavoli\n");

printf("2) Aggiungi tavolo\n");

printf("3) Rimuovi tavolo\n");

printf("4) Assegna tavolo a cameriere\n");

printf("5) Rimuovi tavolo a cameriere\n");

printf("6) Visualizza camerieri\n");

printf("7) Visualizza assegnazioni tavoli\n");

printf("8) Quit\n");

op = multiChoice("Seleziona un opzione", options, 8);

switch(op) {

case '1':

visualizza\_tavoli(conn);

break;

case '2':

aggiungi\_tavolo(conn);

break;

case '3':

rimuovi\_tavolo(conn);

break;

case '4':

assegna\_tavolo\_a\_cameriere(conn);

break;

case '5':

rimuovi\_tavolo\_a\_cameriere(conn);

break;

case '6':

visualizza\_camerieri(conn);

break;

case '7':

visualizza\_assegnazioni\_tavoli\_camerieri(conn);

break;

case '8':

return;

default:

fprintf(stderr, "Invalid condition at %s:%d\n", \_\_FILE\_\_, \_\_LINE\_\_);

abort();

}

getchar();

}

}

static int choose\_day(){

char options[9] = {'1','2','3','4','5','6','7','8'};

char op;

while(true) {

printf("\033[2J\033[H");

printf("\*\*\* Selezionare un giorno \*\*\*\n\n");

printf("1) Lunedi'\n");

printf("2) Martedi'\n");

printf("3) Mercoledi'\n");

printf("4) Giovedi'\n");

printf("5) Venerdi'\n");

printf("6) Sabato\n");

printf("7) Domenica\n");

printf("8) Quit\n");

op = multiChoice("Seleziona un opzione", options, 8);

switch(op) {

case '1':

return 2;

case '2':

return 3;

case '3':

return 4;

case '4':

return 5;

case '5':

return 6;

case '6':

return 7;

case '7':

return 1;

case '8':

return -1;

default:

fprintf(stderr, "Invalid condition at %s:%d\n", \_\_FILE\_\_, \_\_LINE\_\_);

abort();

}

getchar();

}

}

static void visualizza\_turni(MYSQL \*conn){

MYSQL\_STMT \*prepared\_stmt;

//Visualizza turni

if(!setup\_prepared\_stmt(&prepared\_stmt, "call visualizza\_turni()", conn)) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Unable to initialize visualizza\_turni\n", false);

}

// Run procedure

if (mysql\_stmt\_execute(prepared\_stmt) != 0) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Could not retrieve visualizza\_turni\n", true);

}

// Dump the result set

dump\_result\_set(conn, prepared\_stmt, "\nTurni:");

mysql\_stmt\_next\_result(prepared\_stmt);

mysql\_stmt\_close(prepared\_stmt);

}

static void visualizza\_impiegati(MYSQL \*conn){

MYSQL\_STMT \*prepared\_stmt;

//Visualizza impiegati

if(!setup\_prepared\_stmt(&prepared\_stmt, "call visualizza\_impiegati()", conn)) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Unable to initialize visualizza\_impiegati\n", false);

}

// Run procedure

if (mysql\_stmt\_execute(prepared\_stmt) != 0) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Could not retrieve visualizza\_impiegati\n", true);

}

// Dump the result set

dump\_result\_set(conn, prepared\_stmt, "\nImpiegati:");

mysql\_stmt\_next\_result(prepared\_stmt);

mysql\_stmt\_close(prepared\_stmt);

}

static void assegna\_turno\_a\_impiegato(MYSQL \*conn){

MYSQL\_STMT \*prepared\_stmt;

MYSQL\_BIND param[4];

int impiegato;

int giorno;

MYSQL\_TIME ora\_inizio;

MYSQL\_TIME ora\_fine;

ora\_inizio.second=0;

ora\_fine.second=0;

//Assegna turno

if(!setup\_prepared\_stmt(&prepared\_stmt, "call assegna\_turno\_a\_impiegato(?,?,?,?)", conn)) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Unable to initialize assegna\_turno\_a\_impiegato\n", false);

}

// Prepare parameters

memset(param, 0, sizeof(param));

memset(&ora\_inizio,0,sizeof(ora\_inizio));

memset(&ora\_fine,0,sizeof(ora\_fine));

printf("Inserisci matricola impiegato\n");

if(scanf("%d",&impiegato)<1){

printf("Errore acquisizione impiegato\n");

flush\_stdin();

return;

}

flush\_stdin();

printf("Inserire ora inizio turno:\n");

if(scanf("%2d",&ora\_inizio.hour)<1){

printf("Errore inserimento ora inizio turno\n");

flush\_stdin();

return;

}

flush\_stdin();

printf("Inserire minuto inizio turno:\n");

if(scanf("%2d",&ora\_inizio.minute)<1){

printf("Errore inserimento minuto inizio turno\n");

flush\_stdin();

return;

}

flush\_stdin();

printf("Inserire ora fine turno:\n");

if(scanf("%2d",&ora\_fine.hour)<1){

printf("Errore inserimento ora fine turno\n");

flush\_stdin();

return;

}

flush\_stdin();

printf("Inserire minuto fine turno:\n");

if(scanf("%2d",&ora\_fine.minute)<1){

printf("Errore inserimento minuto fine turno\n");

flush\_stdin();

return;

}

flush\_stdin();

if((giorno=choose\_day())==-1){

printf("Non e' stato selezionato un giorno\n");

return;

}

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

param[0].buffer = &impiegato;

param[0].buffer\_length = sizeof(impiegato);

param[1].buffer\_type = MYSQL\_TYPE\_TIME;

param[1].buffer = &ora\_inizio;

param[1].buffer\_length = sizeof(ora\_inizio);

param[2].buffer\_type = MYSQL\_TYPE\_TIME;

param[2].buffer = &ora\_fine;

param[2].buffer\_length = sizeof(ora\_fine);

param[3].buffer\_type = MYSQL\_TYPE\_LONG;

param[3].buffer = &giorno;

param[3].buffer\_length = sizeof(giorno);

if (mysql\_stmt\_bind\_param(prepared\_stmt, param) != 0) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Could not bind parameters for assegna\_turno\_a\_impiegato\n", true);

}

// Run procedure

if (mysql\_stmt\_execute(prepared\_stmt) != 0) {

print\_stmt\_error(prepared\_stmt, "Could not assign workshift to employee\n");

}else printf("Turno assegnato correttamente\n");

mysql\_stmt\_close(prepared\_stmt);

}

static void assegna\_turno\_a\_tavolo(MYSQL \*conn){

MYSQL\_STMT \*prepared\_stmt;

MYSQL\_BIND param[4];

int tavolo;

int giorno;

MYSQL\_TIME ora\_inizio;

MYSQL\_TIME ora\_fine;

ora\_inizio.second=0;

ora\_fine.second=0;

if(!setup\_prepared\_stmt(&prepared\_stmt, "call assegna\_turno\_a\_tavolo(?,?,?,?)", conn)) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Unable to initialize assegna\_turno\_a\_tavolo\n", false);

}

// Prepare parameters

memset(param, 0, sizeof(param));

memset(&ora\_inizio,0,sizeof(ora\_inizio));

memset(&ora\_fine,0,sizeof(ora\_fine));

printf("Inserisci numero tavolo\n");

if(scanf("%d",&tavolo)<1){

printf("Errore acquisizione numero tavolo\n");

flush\_stdin();

return;

}

flush\_stdin();

printf("Inserire ora inizio turno:\n");

if(scanf("%2d",&ora\_inizio.hour)<1){

printf("Errore inserimento ora inizio turno\n");

flush\_stdin();

return;

}

flush\_stdin();

printf("Inserire minuto inizio turno:\n");

if(scanf("%2d",&ora\_inizio.minute)<1){

printf("Errore inserimento minuto inizio turno\n");

flush\_stdin();

return;

}

flush\_stdin();

printf("Inserire ora fine turno:\n");

if(scanf("%2d",&ora\_fine.hour)<1){

printf("Errore inserimento ora fine turno\n");

flush\_stdin();

return;

}

flush\_stdin();

printf("Inserire minuto fine turno:\n");

if(scanf("%2d",&ora\_fine.minute)<1){

printf("Errore inserimento minuto fine turno\n");

flush\_stdin();

return;

}

flush\_stdin();

if((giorno=choose\_day())==-1){

printf("Non e' stato selezionato un giorno\n");

return;

}

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

param[0].buffer = &tavolo;

param[0].buffer\_length = sizeof(tavolo);

param[1].buffer\_type = MYSQL\_TYPE\_TIME;

param[1].buffer = &ora\_inizio;

param[1].buffer\_length = sizeof(ora\_inizio);

param[2].buffer\_type = MYSQL\_TYPE\_TIME;

param[2].buffer = &ora\_fine;

param[2].buffer\_length = sizeof(ora\_fine);

param[3].buffer\_type = MYSQL\_TYPE\_LONG;

param[3].buffer = &giorno;

param[3].buffer\_length = sizeof(giorno);

if (mysql\_stmt\_bind\_param(prepared\_stmt, param) != 0) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Could not bind parameters for assegna\_turno\_a\_tavolo\n", true);

}

// Run procedure

if (mysql\_stmt\_execute(prepared\_stmt) != 0) {

print\_stmt\_error(prepared\_stmt, "Could not assign workshift to table\n");

}else printf("Turno assegnato correttamente\n");

mysql\_stmt\_close(prepared\_stmt);

}

static void visualizza\_turni\_impiegati(MYSQL \*conn){

MYSQL\_STMT \*prepared\_stmt;

//Visualizza turni

if(!setup\_prepared\_stmt(&prepared\_stmt, "call visualizza\_turni\_impiegati()", conn)) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Unable to initialize visualizza\_turni\_impiegati\n", false);

}

// Run procedure

if (mysql\_stmt\_execute(prepared\_stmt) != 0) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Could not retrieve visualizza\_turni\_impiegati\n", true);

}

// Dump the result set

dump\_result\_set(conn, prepared\_stmt, "\nTurni impiegati:");

mysql\_stmt\_next\_result(prepared\_stmt);

mysql\_stmt\_close(prepared\_stmt);

}

static void rimuovi\_turno\_a\_impiegato(MYSQL \*conn){

MYSQL\_STMT \*prepared\_stmt;

MYSQL\_BIND param[4];

int impiegato;

int giorno;

MYSQL\_TIME ora\_inizio;

MYSQL\_TIME ora\_fine;

ora\_inizio.second=0;

ora\_fine.second=0;

if(!setup\_prepared\_stmt(&prepared\_stmt, "call rimuovi\_turno\_impiegato(?,?,?,?)", conn)) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Unable to initialize rimuovi\_turno\_impiegato\n", false);

}

// Prepare parameters

memset(param, 0, sizeof(param));

memset(&ora\_inizio,0,sizeof(ora\_inizio));

memset(&ora\_fine,0,sizeof(ora\_fine));

printf("Inserisci matricola impiegato\n");

if(scanf("%d",&impiegato)<1){

printf("Errore acquisizione matricola impiegato\n");

flush\_stdin();

return;

}

flush\_stdin();

printf("Inserire ora inizio turno:\n");

if(scanf("%2d",&ora\_inizio.hour)<1){

printf("Errore inserimento ora inizio turno\n");

flush\_stdin();

return;

}

flush\_stdin();

printf("Inserire minuto inizio turno:\n");

if(scanf("%2d",&ora\_inizio.minute)<1){

printf("Errore inserimento minuto inizio turno\n");

flush\_stdin();

return;

}

flush\_stdin();

printf("Inserire ora fine turno:\n");

if(scanf("%2d",&ora\_fine.hour)<1){

printf("Errore inserimento ora fine turno\n");

flush\_stdin();

return;

}

flush\_stdin();

printf("Inserire minuto fine turno:\n");

if(scanf("%2d",&ora\_fine.minute)<1){

printf("Errore inserimento minuto fine turno\n");

flush\_stdin();

return;

}

flush\_stdin();

if((giorno=choose\_day())==-1){

printf("Non e' stato selezionato un giorno\n");

return;

}

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

param[0].buffer = &impiegato;

param[0].buffer\_length = sizeof(impiegato);

param[1].buffer\_type = MYSQL\_TYPE\_TIME;

param[1].buffer = &ora\_inizio;

param[1].buffer\_length = sizeof(ora\_inizio);

param[2].buffer\_type = MYSQL\_TYPE\_TIME;

param[2].buffer = &ora\_fine;

param[2].buffer\_length = sizeof(ora\_fine);

param[3].buffer\_type = MYSQL\_TYPE\_LONG;

param[3].buffer = &giorno;

param[3].buffer\_length = sizeof(giorno);

if (mysql\_stmt\_bind\_param(prepared\_stmt, param) != 0) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Could not bind parameters for rimuovi\_turno\_impiegato\n", true);

}

// Run procedure

if (mysql\_stmt\_execute(prepared\_stmt) != 0) {

print\_stmt\_error(prepared\_stmt, "Could not remove employee's workshift\n");

}else printf("Turno rimosso correttamente\n");

mysql\_stmt\_close(prepared\_stmt);

}

static void visualizza\_turni\_tavoli(MYSQL \*conn){

MYSQL\_STMT \*prepared\_stmt;

//Visualizza turni

if(!setup\_prepared\_stmt(&prepared\_stmt, "call visualizza\_turni\_tavoli()", conn)) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Unable to initialize visualizza\_turni\_tavoli\n", false);

}

// Run procedure

if (mysql\_stmt\_execute(prepared\_stmt) != 0) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Could not retrieve visualizza\_turni\_tavoli\n", true);

}

// Dump the result set

dump\_result\_set(conn, prepared\_stmt, "\nTurni tavoli:");

mysql\_stmt\_next\_result(prepared\_stmt);

mysql\_stmt\_close(prepared\_stmt);

}

static void rimuovi\_turno\_a\_tavolo(MYSQL \*conn){

MYSQL\_STMT \*prepared\_stmt;

MYSQL\_BIND param[4];

int tavolo;

int giorno;

MYSQL\_TIME ora\_inizio;

MYSQL\_TIME ora\_fine;

ora\_inizio.second=0;

ora\_fine.second=0;

if(!setup\_prepared\_stmt(&prepared\_stmt, "call rimuovi\_turno\_tavolo(?,?,?,?)", conn)) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Unable to initialize rimuovi\_turno\_tavolo\n", false);

}

// Prepare parameters

memset(param, 0, sizeof(param));

memset(&ora\_inizio,0,sizeof(ora\_inizio));

memset(&ora\_fine,0,sizeof(ora\_fine));

printf("Inserisci numero tavolo\n");

if(scanf("%d",&tavolo)<1){

printf("Errore acquisizione numero tavolo\n");

flush\_stdin();

return;

}

flush\_stdin();

printf("Inserire ora inizio turno:\n");

if(scanf("%2d",&ora\_inizio.hour)<1){

printf("Errore inserimento ora inizio turno\n");

flush\_stdin();

return;

}

flush\_stdin();

printf("Inserire minuto inizio turno:\n");

if(scanf("%2d",&ora\_inizio.minute)<1){

printf("Errore inserimento minuto inizio turno\n");

flush\_stdin();

return;

}

flush\_stdin();

printf("Inserire ora fine turno:\n");

if(scanf("%2d",&ora\_fine.hour)<1){

printf("Errore inserimento ora fine turno\n");

flush\_stdin();

return;

}

flush\_stdin();

printf("Inserire minuto fine turno:\n");

if(scanf("%2d",&ora\_fine.minute)<1){

printf("Errore inserimento minuto fine turno\n");

flush\_stdin();

return;

}

flush\_stdin();

if((giorno=choose\_day())==-1){

printf("Non e' stato selezionato un giorno\n");

return;

}

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

param[0].buffer = &tavolo;

param[0].buffer\_length = sizeof(tavolo);

param[1].buffer\_type = MYSQL\_TYPE\_TIME;

param[1].buffer = &ora\_inizio;

param[1].buffer\_length = sizeof(ora\_inizio);

param[2].buffer\_type = MYSQL\_TYPE\_TIME;

param[2].buffer = &ora\_fine;

param[2].buffer\_length = sizeof(ora\_fine);

param[3].buffer\_type = MYSQL\_TYPE\_LONG;

param[3].buffer = &giorno;

param[3].buffer\_length = sizeof(giorno);

if (mysql\_stmt\_bind\_param(prepared\_stmt, param) != 0) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Could not bind parameters for rimuovi\_turno\_tavolo\n", true);

}

// Run procedure

if (mysql\_stmt\_execute(prepared\_stmt) != 0) {

print\_stmt\_error(prepared\_stmt, "Could not remove table's workshift\n");

}else printf("Turno rimosso correttamente\n");

mysql\_stmt\_close(prepared\_stmt);

}

static void aggiungi\_turno(MYSQL \*conn){

MYSQL\_STMT \*prepared\_stmt;

MYSQL\_BIND param[3];

int giorno;

MYSQL\_TIME ora\_inizio;

MYSQL\_TIME ora\_fine;

ora\_inizio.second=0;

ora\_fine.second=0;

if(!setup\_prepared\_stmt(&prepared\_stmt, "call aggiungi\_turno(?,?,?)", conn)) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Unable to initialize aggiungi\_turno\n", false);

}

// Prepare parameters

memset(param, 0, sizeof(param));

memset(&ora\_inizio,0,sizeof(ora\_inizio));

memset(&ora\_fine,0,sizeof(ora\_fine));

printf("Inserire ora inizio turno:\n");

if(scanf("%02d",&ora\_inizio.hour)<1){

printf("Errore inserimento ora inizio turno\n");

flush\_stdin();

return;

}

flush\_stdin();

printf("Inserire minuto inizio turno:\n");

if(scanf("%02d",&ora\_inizio.minute)<1){

printf("Errore inserimento minuto inizio turno\n");

flush\_stdin();

return;

}

flush\_stdin();

printf("Inserire ora fine turno:\n");

if(scanf("%02d",&ora\_fine.hour)<1){

printf("Errore inserimento ora fine turno\n");

flush\_stdin();

return;

}

flush\_stdin();

printf("Inserire minuto fine turno:\n");

if(scanf("%02d",&ora\_fine.minute)<1){

printf("Errore inserimento minuto fine turno\n");

flush\_stdin();

return;

}

flush\_stdin();

if((giorno=choose\_day())==-1){

printf("Non e' stato selezionato un giorno\n");

return;

}

param[0].buffer\_type = MYSQL\_TYPE\_TIME;

param[0].buffer = (char \*)&ora\_inizio;

param[0].buffer\_length = sizeof(ora\_inizio);

param[1].buffer\_type = MYSQL\_TYPE\_TIME;

param[1].buffer = &ora\_fine;

param[1].buffer\_length = sizeof(ora\_fine);

param[2].buffer\_type = MYSQL\_TYPE\_LONG;

param[2].buffer = &giorno;

param[2].buffer\_length = sizeof(giorno);

if (mysql\_stmt\_bind\_param(prepared\_stmt, param) != 0) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Could not bind parameters for aggiungi\_turno\n", true);

}

// Run procedure

if (mysql\_stmt\_execute(prepared\_stmt) != 0) {

print\_stmt\_error(prepared\_stmt, "Could not add workshift\n");

}else printf("Turno aggiunto correttamente\n");

mysql\_stmt\_close(prepared\_stmt);

}

static void rimuovi\_turno(MYSQL \*conn){

MYSQL\_STMT \*prepared\_stmt;

MYSQL\_BIND param[3];

int giorno;

MYSQL\_TIME ora\_inizio;

MYSQL\_TIME ora\_fine;

ora\_inizio.second=0;

ora\_fine.second=0;

//rimuovi turno

if(!setup\_prepared\_stmt(&prepared\_stmt, "call rimuovi\_turno(?,?,?)", conn)) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Unable to initialize rimuovi\_turno\n", false);

}

// Prepare parameters

memset(param, 0, sizeof(param));

memset(&ora\_inizio,0,sizeof(ora\_inizio));

memset(&ora\_fine,0,sizeof(ora\_fine));

printf("Inserire ora inizio turno:\n");

if(scanf("%2d",&ora\_inizio.hour)<1){

printf("Errore inserimento ora inizio turno\n");

flush\_stdin();

return;

}

flush\_stdin();

printf("Inserire minuto inizio turno:\n");

if(scanf("%2d",&ora\_inizio.minute)<1){

printf("Errore inserimento minuto inizio turno\n");

flush\_stdin();

return;

}

flush\_stdin();

printf("Inserire ora fine turno:\n");

if(scanf("%2d",&ora\_fine.hour)<1){

printf("Errore inserimento ora fine turno\n");

flush\_stdin();

return;

}

flush\_stdin();

printf("Inserire minuto fine turno:\n");

if(scanf("%2d",&ora\_fine.minute)<1){

printf("Errore inserimento minuto fine turno\n");

flush\_stdin();

return;

}

flush\_stdin();

if((giorno=choose\_day())==-1){

printf("Non e' stato selezionato un giorno\n");

return;

}

param[0].buffer\_type = MYSQL\_TYPE\_TIME;

param[0].buffer = (char \*)&ora\_inizio;

param[0].buffer\_length = sizeof(ora\_inizio);

param[1].buffer\_type = MYSQL\_TYPE\_TIME;

param[1].buffer = (char \*)&ora\_fine;

param[1].buffer\_length = sizeof(ora\_fine);

param[2].buffer\_type = MYSQL\_TYPE\_LONG;

param[2].buffer = &giorno;

param[2].buffer\_length = sizeof(giorno);

if (mysql\_stmt\_bind\_param(prepared\_stmt, param) != 0) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Could not bind parameters for rimuovi\_turno\n", true);

}

// Run procedure

if (mysql\_stmt\_execute(prepared\_stmt) != 0) {

print\_stmt\_error(prepared\_stmt, "Could not remove workshift\n");

}else printf("Turno rimosso correttamente\n");

mysql\_stmt\_close(prepared\_stmt);

}

static void gestisci\_turni\_tavoli(MYSQL \*conn){

char options[6] = {'1','2','3','4','5'};

char op;

while(true) {

printf("\033[2J\033[H");

printf("\*\*\* Cosa posso fare per te? \*\*\*\n\n");

printf("1) Visualizza turni tavoli\n");

printf("2) Visualizza turni\n");

printf("3) Assegna turno a tavolo\n");

printf("4) Rimuovi turno a tavolo\n");

printf("5) Quit\n");

op = multiChoice("Seleziona un opzione", options, 5);

switch(op) {

case '1':

visualizza\_turni\_tavoli(conn);

break;

case '2':

visualizza\_turni(conn);

break;

case '3':

assegna\_turno\_a\_tavolo(conn);

break;

case '4':

rimuovi\_turno\_a\_tavolo(conn);

break;

case '5':

return;

default:

fprintf(stderr, "Invalid condition at %s:%d\n", \_\_FILE\_\_, \_\_LINE\_\_);

abort();

}

getchar();

}

}

static void gestisci\_turni\_impiegati(MYSQL \*conn){

char options[6] = {'1','2','3','4','5'};

char op;

while(true) {

printf("\033[2J\033[H");

printf("\*\*\* Cosa posso fare per te? \*\*\*\n\n");

printf("1) Visualizza turni impiegati\n");

printf("2) Visualizza turni\n");

printf("3) Assegna turno a impiegato\n");

printf("4) Rimuovi turno a impiegato\n");

printf("5) Quit\n");

op = multiChoice("Seleziona un opzione", options, 5);

switch(op) {

case '1':

visualizza\_turni\_impiegati(conn);

break;

case '2':

visualizza\_turni(conn);

break;

case '3':

assegna\_turno\_a\_impiegato(conn);

break;

case '4':

rimuovi\_turno\_a\_impiegato(conn);

break;

case '5':

return;

default:

fprintf(stderr, "Invalid condition at %s:%d\n", \_\_FILE\_\_, \_\_LINE\_\_);

abort();

}

getchar();

}

}

static void gestisci\_turni(MYSQL \*conn){

char options[7] = {'1','2','3','4','5','6'};

char op;

while(true) {

printf("\033[2J\033[H");

printf("\*\*\* Cosa posso fare per te? \*\*\*\n\n");

printf("1) Gestisci turni impiegati\n");

printf("2) Gestisci turni tavoli\n");

printf("3) Visualizza turni\n");

printf("4) Aggiungi turno\n");

printf("5) Rimuovi turno\n");

printf("6) Quit\n");

op = multiChoice("Seleziona un opzione", options, 6);

switch(op) {

case '1':

gestisci\_turni\_impiegati(conn);

break;

case '2':

gestisci\_turni\_tavoli(conn);

break;

case '3':

visualizza\_turni(conn);

break;

case '4':

aggiungi\_turno(conn);

break;

case '5':

rimuovi\_turno(conn);

break;

case '6':

return;

default:

fprintf(stderr, "Invalid condition at %s:%d\n", \_\_FILE\_\_, \_\_LINE\_\_);

abort();

}

getchar();

}

}

static int choose\_role(){

char options[6] = {'1','2','3','4','5'};

char op;

while(true) {

printf("\033[2J\033[H");

printf("\*\*\* Selezionare un ruolo \*\*\*\n\n");

printf("1) Manager\n");

printf("2) Cameriere\n");

printf("3) Pizzaiolo\n");

printf("4) Barman\n");

printf("5) Quit\n");

op = multiChoice("Seleziona un opzione", options, 5);

switch(op) {

case '1':

return 1;

case '2':

return 2;

case '3':

return 3;

case '4':

return 4;

case '5':

return -1;

default:

fprintf(stderr, "Invalid condition at %s:%d\n", \_\_FILE\_\_, \_\_LINE\_\_);

abort();

}

getchar();

}

}

static void aggiungi\_impiegato(MYSQL \*conn){

MYSQL\_STMT \*prepared\_stmt;

MYSQL\_BIND param[5];

int matricola;

char nome[45];

char cognome[45];

char passwd[45];

int ruolo;

if(!setup\_prepared\_stmt(&prepared\_stmt, "call aggiungi\_impiegato(?,?,?,?,?)", conn)) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Unable to initialize aggiungi\_impiegato\n", false);

}

// Prepare parameters

memset(param, 0, sizeof(param));

memset(nome,'\0', sizeof(nome));

memset(cognome,'\0', sizeof(cognome));

memset(passwd,'\0', sizeof(passwd));

printf("Attenzione le seguenti informazioni sono definitive\n");

printf("Inserire matricola impiegato:\n");

if(scanf("%i",&matricola)<1){

printf("Errore inserimento matricola\n");

flush\_stdin();

return;

}

flush\_stdin();

printf("Inserire nome impiegato:\n");

getInput(45,nome,false);

printf("Inserire cognome impiegato:\n");

getInput(45,cognome,false);

printf("Inserire password impiegato\n");

getInput(45,passwd,true);

if((ruolo=choose\_role())==-1){

printf("Non è stato selezionato un ruolo\n");

return;

}

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

param[0].buffer = &matricola;

param[0].buffer\_length = sizeof(matricola);

param[1].buffer\_type = MYSQL\_TYPE\_STRING;

param[1].buffer = nome;

param[1].buffer\_length = sizeof(nome);

param[2].buffer\_type = MYSQL\_TYPE\_STRING;

param[2].buffer = cognome;

param[2].buffer\_length = sizeof(cognome);

param[3].buffer\_type = MYSQL\_TYPE\_STRING;

param[3].buffer = passwd;

param[3].buffer\_length = sizeof(passwd);

param[4].buffer\_type = MYSQL\_TYPE\_LONG;

param[4].buffer = &ruolo;

param[4].buffer\_length = sizeof(ruolo);

if (mysql\_stmt\_bind\_param(prepared\_stmt, param) != 0) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Could not bind parameters for aggiungi\_impiegato\n", true);

}

// Run procedure

if (mysql\_stmt\_execute(prepared\_stmt) != 0) {

print\_stmt\_error(prepared\_stmt, "Could not add employee\n");

}else printf("Impiegato aggiunto correttamente\n");

mysql\_stmt\_close(prepared\_stmt);

}

static void rimuovi\_impiegato(MYSQL \*conn){

MYSQL\_STMT \*prepared\_stmt;

MYSQL\_BIND param[1];

int matricola;

if(!setup\_prepared\_stmt(&prepared\_stmt, "call rimuovi\_impiegato(?)", conn)) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Unable to initialize aggiungi\_rimuovi\n", false);

}

// Prepare parameters

memset(param, 0, sizeof(param));

printf("Inserire matricola impiegato:\n");

if(scanf("%d",&matricola)<1){

printf("Errore inserimento matricola\n");

flush\_stdin();

return;

}

flush\_stdin();

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

param[0].buffer = &matricola;

param[0].buffer\_length = sizeof(matricola);

if (mysql\_stmt\_bind\_param(prepared\_stmt, param) != 0) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Could not bind parameters for rimuovi\_impiegato\n", true);

}

// Run procedure

if (mysql\_stmt\_execute(prepared\_stmt) != 0) {

print\_stmt\_error(prepared\_stmt, "Could not remove employee\n");

}else printf("Impiegato rimosso correttamente\n");

mysql\_stmt\_close(prepared\_stmt);

}

static void gestisci\_impiegati(MYSQL \*conn){

char options[5] = {'1','2','3','4'};

char op;

while(true) {

printf("\033[2J\033[H");

printf("\*\*\* Cosa posso fare per te? \*\*\*\n\n");

printf("1) Aggiungi impiegato\n");

printf("2) Rimuovi impiegato\n");

printf("3) Visualizza impiegati\n");

printf("4) Quit\n");

op = multiChoice("Seleziona un opzione", options, 4);

switch(op) {

case '1':

aggiungi\_impiegato(conn);

break;

case '2':

rimuovi\_impiegato(conn);

break;

case '3':

visualizza\_impiegati(conn);

break;

case '4':

return;

default:

fprintf(stderr, "Invalid condition at %s:%d\n", \_\_FILE\_\_, \_\_LINE\_\_);

abort();

}

getchar();

}

}

void run\_as\_manager(MYSQL \*conn){

char options[10] = {'1','2','3','4','5','6','7','8','9'};

char op;

printf("Passo al ruolo di manager...\n");

if(!parse\_config("users/manager.json", &conf)) {

fprintf(stderr, "Unable to load manager configuration\n");

exit(EXIT\_FAILURE);

}

if(mysql\_change\_user(conn, conf.db\_username, conf.db\_password, conf.database)) {

fprintf(stderr, "mysql\_change\_user() failed\n");

exit(EXIT\_FAILURE);

}

while(true) {

printf("\033[2J\033[H");

printf("\*\*\* Cosa posso fare per te? \*\*\*\n\n");

printf("1) Registra cliente\n");

printf("2) Stampa scontrino\n");

printf("3) Gestisci prodotti\n");

printf("4) Gestisci tavoli\n");

printf("5) Gestisci impiegati\n");

printf("6) Gestisci turni\n");

printf("7) Visualizza entrate giornaliere\n");

printf("8) Visualizza entrate mensili\n");

printf("9) Quit\n");

op = multiChoice("Seleziona un opzione", options, 9);

switch(op) {

case '1':

registra\_cliente(conn);

break;

case '2':

stampa\_scontrino\_tavolo(conn);

break;

case '3':

gestisci\_prodotti(conn);

break;

case '4':

gestisci\_tavoli(conn);

break;

case '5':

gestisci\_impiegati(conn);

break;

case '6':

gestisci\_turni(conn);

break;

case '7':

visualizza\_entrate\_giorno(conn);

break;

case '8':

visualizza\_entrate\_mese(conn);

break;

case '9':

return;

default:

fprintf(stderr, "Invalid condition at %s:%d\n", \_\_FILE\_\_, \_\_LINE\_\_);

abort();

}

getchar();

}

}

* Codice di appoggio, defines.h libreria per l’accesso ai moduli di appoggio

#pragma once

#include <stdbool.h>

#include <mysql/mysql.h>

struct configuration {

char \*host;

char \*db\_username;

char \*db\_password;

unsigned int port;

char \*database;

int username;

char password[128];

};

extern struct configuration conf;

extern int parse\_config(char \*path, struct configuration \*conf);

extern char \*getInput(unsigned int lung, char \*stringa, bool hide);

extern bool yesOrNo(char \*domanda, char yes, char no, bool predef, bool insensitive);

extern char multiChoice(char \*domanda, char choices[], int num);

extern void print\_error (MYSQL \*conn, char \*message);

extern void print\_stmt\_error (MYSQL\_STMT \*stmt, char \*message);

extern void finish\_with\_error(MYSQL \*conn, char \*message);

extern void finish\_with\_stmt\_error(MYSQL \*conn, MYSQL\_STMT \*stmt, char \*message, bool close\_stmt);

extern bool setup\_prepared\_stmt(MYSQL\_STMT \*\*stmt, char \*statement, MYSQL \*conn);

extern void dump\_result\_set(MYSQL \*conn, MYSQL\_STMT \*stmt, char \*title);

extern void run\_as\_manager(MYSQL \*conn);

extern void run\_as\_pizzaiolo(MYSQL \*conn);

extern void run\_as\_barman(MYSQL \*conn);

extern void run\_as\_cameriere(MYSQL \*conn);

extern void flush\_stdin();

* Codice di appoggio, utils.c per il corretto interfacciamento e utilizzo delle api, utile a togliere ridondanza dal codice principale del client.

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <ctype.h>

#include "defines.h"

void flush\_stdin(){

int c;

while ((c = getchar()) != '\n' && c != EOF) { }

}

void print\_stmt\_error (MYSQL\_STMT \*stmt, char \*message)

{

fprintf (stderr, "%s\n", message);

if (stmt != NULL) {

fprintf (stderr, "Error %u (%s): %s\n",

mysql\_stmt\_errno (stmt),

mysql\_stmt\_sqlstate(stmt),

mysql\_stmt\_error (stmt));

}

}

void print\_error(MYSQL \*conn, char \*message)

{

fprintf (stderr, "%s\n", message);

if (conn != NULL) {

#if MYSQL\_VERSION\_ID >= 40101

fprintf (stderr, "Error %u (%s): %s\n",

mysql\_errno (conn), mysql\_sqlstate(conn), mysql\_error (conn));

#else

fprintf (stderr, "Error %u: %s\n",

mysql\_errno (conn), mysql\_error (conn));

#endif

}

}

bool setup\_prepared\_stmt(MYSQL\_STMT \*\*stmt, char \*statement, MYSQL \*conn)

{

my\_bool update\_length = true;

\*stmt = mysql\_stmt\_init(conn);

if (\*stmt == NULL)

{

print\_error(conn, "Could not initialize statement handler");

return false;

}

if (mysql\_stmt\_prepare (\*stmt, statement, strlen(statement)) != 0) {

print\_stmt\_error(\*stmt, "Could not prepare statement");

return false;

}

mysql\_stmt\_attr\_set(\*stmt, STMT\_ATTR\_UPDATE\_MAX\_LENGTH, &update\_length);

return true;

}

void finish\_with\_error(MYSQL \*conn, char \*message)

{

print\_error(conn, message);

mysql\_close(conn);

exit(EXIT\_FAILURE);

}

void finish\_with\_stmt\_error(MYSQL \*conn, MYSQL\_STMT \*stmt, char \*message, bool close\_stmt)

{

print\_stmt\_error(stmt, message);

if(close\_stmt) mysql\_stmt\_close(stmt);

mysql\_close(conn);

exit(EXIT\_FAILURE);

}

static void print\_dashes(MYSQL\_RES \*res\_set)

{

MYSQL\_FIELD \*field;

unsigned int i, j;

mysql\_field\_seek(res\_set, 0);

putchar('+');

for (i = 0; i < mysql\_num\_fields(res\_set); i++) {

field = mysql\_fetch\_field(res\_set);

for (j = 0; j < field->length + 2; j++)

putchar('-');

putchar('+');

}

putchar('\n');

}

static void dump\_result\_set\_header(MYSQL\_RES \*res\_set)

{

MYSQL\_FIELD \*field;

unsigned long col\_len;

unsigned int i;

/\* determine column display widths -- requires result set to be \*/

/\* generated with mysql\_store\_result(), not mysql\_use\_result() \*/

mysql\_field\_seek (res\_set, 0);

for (i = 0; i < mysql\_num\_fields (res\_set); i++) {

field = mysql\_fetch\_field (res\_set);

col\_len = strlen(field->name);

if (col\_len < field->length)

col\_len = field->length;

if (col\_len < 4 && !IS\_NOT\_NULL(field->flags))

col\_len = 4; /\* 4 = length of the word "NULL" \*/

field->length = col\_len; /\* reset column info \*/

}

print\_dashes(res\_set);

putchar('|');

mysql\_field\_seek (res\_set, 0);

for (i = 0; i < mysql\_num\_fields(res\_set); i++) {

field = mysql\_fetch\_field(res\_set);

printf(" %-\*s |", (int)field->length, field->name);

}

putchar('\n');

print\_dashes(res\_set);

}

void dump\_result\_set(MYSQL \*conn, MYSQL\_STMT \*stmt, char \*title)

{

int i;

int status;

int num\_fields; /\* number of columns in result \*/

MYSQL\_FIELD \*fields; /\* for result set metadata \*/

MYSQL\_BIND \*rs\_bind; /\* for output buffers \*/

MYSQL\_RES \*rs\_metadata;

MYSQL\_TIME \*date;

size\_t attr\_size;

/\* Prefetch the whole result set. This in conjunction with

\* STMT\_ATTR\_UPDATE\_length set in `setup\_prepared\_stmt`

\* updates the result set metadata which are fetched in this

\* function, to allow to compute the actual max length of

\* the columns.

\*/

if (mysql\_stmt\_store\_result(stmt)) {

fprintf(stderr, " mysql\_stmt\_execute(), 1 failed\n");

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

exit(0);

}

/\* the column count is > 0 if there is a result set \*/

/\* 0 if the result is only the final status packet \*/

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

if (num\_fields > 0) {

/\* there is a result set to fetch \*/

printf("%s\n", title);

if((rs\_metadata = mysql\_stmt\_result\_metadata(stmt)) == NULL) {

finish\_with\_stmt\_error(conn, stmt, "Unable to retrieve result metadata\n", true);

}

dump\_result\_set\_header(rs\_metadata);

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

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

if (!rs\_bind) {

finish\_with\_stmt\_error(conn, stmt, "Cannot allocate output buffers\n", true);

}

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

/\* set up and bind result set output buffers \*/

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

// Properly size the parameter buffer

switch(fields[i].type) {

case MYSQL\_TYPE\_DATE:

case MYSQL\_TYPE\_TIMESTAMP:

case MYSQL\_TYPE\_DATETIME:

case MYSQL\_TYPE\_TIME:

attr\_size = sizeof(MYSQL\_TIME);

break;

case MYSQL\_TYPE\_FLOAT:

attr\_size = sizeof(float);

break;

case MYSQL\_TYPE\_DOUBLE:

attr\_size = sizeof(double);

break;

case MYSQL\_TYPE\_TINY:

attr\_size = sizeof(signed char);

break;

case MYSQL\_TYPE\_SHORT:

case MYSQL\_TYPE\_YEAR:

attr\_size = sizeof(short int);

break;

case MYSQL\_TYPE\_LONG:

case MYSQL\_TYPE\_INT24:

attr\_size = sizeof(int);

break;

case MYSQL\_TYPE\_LONGLONG:

attr\_size = sizeof(long long int);

break;

default:

attr\_size = fields[i].length;

break;

}

// Setup the binding for the current parameter

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

rs\_bind[i].buffer = malloc(attr\_size + 1);

rs\_bind[i].buffer\_length = attr\_size + 1;

if(rs\_bind[i].buffer == NULL) {

finish\_with\_stmt\_error(conn, stmt, "Cannot allocate output buffers\n", true);

}

}

if(mysql\_stmt\_bind\_result(stmt, rs\_bind)) {

finish\_with\_stmt\_error(conn, stmt, "Unable to bind output parameters\n", true);

}

/\* fetch and display result set rows \*/

while (true) {

status = mysql\_stmt\_fetch(stmt);

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

break;

putchar('|');

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

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

printf (" %-\*s |", (int)fields[i].length, "NULL");

continue;

}

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

case MYSQL\_TYPE\_VAR\_STRING:

printf(" %-\*s |", (int)fields[i].length, (char \*)rs\_bind[i].buffer);

break;

case MYSQL\_TYPE\_DATETIME:

case MYSQL\_TYPE\_TIMESTAMP:

date = (MYSQL\_TIME \*)rs\_bind[i].buffer;

printf("%02d/%02d/%4d-%02d:%02d:%02d |", date->day, date->month, date->year,date->hour,date->minute,date->second);

break;

case MYSQL\_TYPE\_TIME:

date = (MYSQL\_TIME \*)rs\_bind[i].buffer;

printf(" %02d-", date->hour);

printf("%02d-", date->minute);

printf("%-\*.02d |",((int)fields[i].length-6) ,date->second);

break;

case MYSQL\_TYPE\_DATE:

date = (MYSQL\_TIME \*)rs\_bind[i].buffer;

printf(" %4d-%02d-%02d |", date->year, date->month, date->day);

break;

case MYSQL\_TYPE\_STRING:

printf(" %-\*s |", (int)fields[i].length, (char \*)rs\_bind[i].buffer);

break;

case MYSQL\_TYPE\_FLOAT:

case MYSQL\_TYPE\_DOUBLE:

printf(" %-\*.02f |", (int)fields[i].length,\*(float \*)rs\_bind[i].buffer);

break;

case MYSQL\_TYPE\_LONG:

case MYSQL\_TYPE\_SHORT:

case MYSQL\_TYPE\_TINY:

printf(" %-\*d |", (int)fields[i].length, \*(int \*)rs\_bind[i].buffer);

break;

case MYSQL\_TYPE\_LONGLONG:

printf(" %-\*lld |", (int)fields[i].length, \*(long long int \*)rs\_bind[i].buffer);

break;

case MYSQL\_TYPE\_NEWDECIMAL:

printf(" %-\*.02lf |", (int)fields[i].length, \*(float\*) rs\_bind[i].buffer);

break;

default:

printf("ERROR: Unhandled type (%d)\n", rs\_bind[i].buffer\_type);

abort();

}

}

putchar('\n');

print\_dashes(rs\_metadata);

}

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

/\* free output buffers \*/

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

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

}

free(rs\_bind);

}

}

* Codice di appoggio, parse.c per il parsing dei json delle varie tipologie di utente

#include <stddef.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include "defines.h"

#define BUFF\_SIZE 4096

// The final config struct will point into this

static char config[BUFF\_SIZE];

/\*\*

\* JSON type identifier. Basic types are:

\* o Object

\* o Array

\* o String

\* o Other primitive: number, boolean (true/false) or null

\*/

typedef enum {

JSMN\_UNDEFINED = 0,

JSMN\_OBJECT = 1,

JSMN\_ARRAY = 2,

JSMN\_STRING = 3,

JSMN\_PRIMITIVE = 4

} jsmntype\_t;

enum jsmnerr {

/\* Not enough tokens were provided \*/

JSMN\_ERROR\_NOMEM = -1,

/\* Invalid character inside JSON string \*/

JSMN\_ERROR\_INVAL = -2,

/\* The string is not a full JSON packet, more bytes expected \*/

JSMN\_ERROR\_PART = -3

};

/\*\*

\* JSON token description.

\* type type (object, array, string etc.)

\* start start position in JSON data string

\* end end position in JSON data string

\*/

typedef struct {

jsmntype\_t type;

int start;

int end;

int size;

#ifdef JSMN\_PARENT\_LINKS

int parent;

#endif

} jsmntok\_t;

/\*\*

\* JSON parser. Contains an array of token blocks available. Also stores

\* the string being parsed now and current position in that string

\*/

typedef struct {

unsigned int pos; /\* offset in the JSON string \*/

unsigned int toknext; /\* next token to allocate \*/

int toksuper; /\* superior token node, e.g parent object or array \*/

} jsmn\_parser;

/\*\*

\* Allocates a fresh unused token from the token pool.

\*/

static jsmntok\_t \*jsmn\_alloc\_token(jsmn\_parser \*parser, jsmntok\_t \*tokens, size\_t num\_tokens) {

jsmntok\_t \*tok;

if (parser->toknext >= num\_tokens) {

return NULL;

}

tok = &tokens[parser->toknext++];

tok->start = tok->end = -1;

tok->size = 0;

#ifdef JSMN\_PARENT\_LINKS

tok->parent = -1;

#endif

return tok;

}

/\*\*

\* Fills token type and boundaries.

\*/

static void jsmn\_fill\_token(jsmntok\_t \*token, jsmntype\_t type,

int start, int end) {

token->type = type;

token->start = start;

token->end = end;

token->size = 0;

}

/\*\*

\* Fills next available token with JSON primitive.

\*/

static int jsmn\_parse\_primitive(jsmn\_parser \*parser, const char \*js,

size\_t len, jsmntok\_t \*tokens, size\_t num\_tokens) {

jsmntok\_t \*token;

int start;

start = parser->pos;

for (; parser->pos < len && js[parser->pos] != '\0'; parser->pos++) {

switch (js[parser->pos]) {

#ifndef JSMN\_STRICT

/\* In strict mode primitive must be followed by "," or "}" or "]" \*/

case ':':

#endif

case '\t' : case '\r' : case '\n' : case ' ' :

case ',' : case ']' : case '}' :

goto found;

}

if (js[parser->pos] < 32 || js[parser->pos] >= 127) {

parser->pos = start;

return JSMN\_ERROR\_INVAL;

}

}

#ifdef JSMN\_STRICT

/\* In strict mode primitive must be followed by a comma/object/array \*/

parser->pos = start;

return JSMN\_ERROR\_PART;

#endif

found:

if (tokens == NULL) {

parser->pos--;

return 0;

}

token = jsmn\_alloc\_token(parser, tokens, num\_tokens);

if (token == NULL) {

parser->pos = start;

return JSMN\_ERROR\_NOMEM;

}

jsmn\_fill\_token(token, JSMN\_PRIMITIVE, start, parser->pos);

#ifdef JSMN\_PARENT\_LINKS

token->parent = parser->toksuper;

#endif

parser->pos--;

return 0;

}

/\*\*

\* Fills next token with JSON string.

\*/

static int jsmn\_parse\_string(jsmn\_parser \*parser, const char \*js,

size\_t len, jsmntok\_t \*tokens, size\_t num\_tokens) {

jsmntok\_t \*token;

int start = parser->pos;

parser->pos++;

/\* Skip starting quote \*/

for (; parser->pos < len && js[parser->pos] != '\0'; parser->pos++) {

char c = js[parser->pos];

/\* Quote: end of string \*/

if (c == '\"') {

if (tokens == NULL) {

return 0;

}

token = jsmn\_alloc\_token(parser, tokens, num\_tokens);

if (token == NULL) {

parser->pos = start;

return JSMN\_ERROR\_NOMEM;

}

jsmn\_fill\_token(token, JSMN\_STRING, start+1, parser->pos);

#ifdef JSMN\_PARENT\_LINKS

token->parent = parser->toksuper;

#endif

return 0;

}

/\* Backslash: Quoted symbol expected \*/

if (c == '\\' && parser->pos + 1 < len) {

int i;

parser->pos++;

switch (js[parser->pos]) {

/\* Allowed escaped symbols \*/

case '\"': case '/' : case '\\' : case 'b' :

case 'f' : case 'r' : case 'n' : case 't' :

break;

/\* Allows escaped symbol \uXXXX \*/

case 'u':

parser->pos++;

for(i = 0; i < 4 && parser->pos < len && js[parser->pos] != '\0'; i++) {

/\* If it isn't a hex character we have an error \*/

if(!((js[parser->pos] >= 48 && js[parser->pos] <= 57) || /\* 0-9 \*/

(js[parser->pos] >= 65 && js[parser->pos] <= 70) || /\* A-F \*/

(js[parser->pos] >= 97 && js[parser->pos] <= 102))) { /\* a-f \*/

parser->pos = start;

return JSMN\_ERROR\_INVAL;

}

parser->pos++;

}

parser->pos--;

break;

/\* Unexpected symbol \*/

default:

parser->pos = start;

return JSMN\_ERROR\_INVAL;

}

}

}

parser->pos = start;

return JSMN\_ERROR\_PART;

}

/\*\*

\* Parse JSON string and fill tokens.

\*/

static int jsmn\_parse(jsmn\_parser \*parser, const char \*js, size\_t len, jsmntok\_t \*tokens, unsigned int num\_tokens) {

int r;

int i;

jsmntok\_t \*token;

int count = parser->toknext;

for (; parser->pos < len && js[parser->pos] != '\0'; parser->pos++) {

char c;

jsmntype\_t type;

c = js[parser->pos];

switch (c) {

case '{': case '[':

count++;

if (tokens == NULL) {

break;

}

token = jsmn\_alloc\_token(parser, tokens, num\_tokens);

if (token == NULL)

return JSMN\_ERROR\_NOMEM;

if (parser->toksuper != -1) {

tokens[parser->toksuper].size++;

#ifdef JSMN\_PARENT\_LINKS

token->parent = parser->toksuper;

#endif

}

token->type = (c == '{' ? JSMN\_OBJECT : JSMN\_ARRAY);

token->start = parser->pos;

parser->toksuper = parser->toknext - 1;

break;

case '}': case ']':

if (tokens == NULL)

break;

type = (c == '}' ? JSMN\_OBJECT : JSMN\_ARRAY);

#ifdef JSMN\_PARENT\_LINKS

if (parser->toknext < 1) {

return JSMN\_ERROR\_INVAL;

}

token = &tokens[parser->toknext - 1];

for (;;) {

if (token->start != -1 && token->end == -1) {

if (token->type != type) {

return JSMN\_ERROR\_INVAL;

}

token->end = parser->pos + 1;

parser->toksuper = token->parent;

break;

}

if (token->parent == -1) {

if(token->type != type || parser->toksuper == -1) {

return JSMN\_ERROR\_INVAL;

}

break;

}

token = &tokens[token->parent];

}

#else

for (i = parser->toknext - 1; i >= 0; i--) {

token = &tokens[i];

if (token->start != -1 && token->end == -1) {

if (token->type != type) {

return JSMN\_ERROR\_INVAL;

}

parser->toksuper = -1;

token->end = parser->pos + 1;

break;

}

}

/\* Error if unmatched closing bracket \*/

if (i == -1) return JSMN\_ERROR\_INVAL;

for (; i >= 0; i--) {

token = &tokens[i];

if (token->start != -1 && token->end == -1) {

parser->toksuper = i;

break;

}

}

#endif

break;

case '\"':

r = jsmn\_parse\_string(parser, js, len, tokens, num\_tokens);

if (r < 0) return r;

count++;

if (parser->toksuper != -1 && tokens != NULL)

tokens[parser->toksuper].size++;

break;

case '\t' : case '\r' : case '\n' : case ' ':

break;

case ':':

parser->toksuper = parser->toknext - 1;

break;

case ',':

if (tokens != NULL && parser->toksuper != -1 &&

tokens[parser->toksuper].type != JSMN\_ARRAY &&

tokens[parser->toksuper].type != JSMN\_OBJECT) {

#ifdef JSMN\_PARENT\_LINKS

parser->toksuper = tokens[parser->toksuper].parent;

#else

for (i = parser->toknext - 1; i >= 0; i--) {

if (tokens[i].type == JSMN\_ARRAY || tokens[i].type == JSMN\_OBJECT) {

if (tokens[i].start != -1 && tokens[i].end == -1) {

parser->toksuper = i;

break;

}

}

}

#endif

}

break;

#ifdef JSMN\_STRICT

/\* In strict mode primitives are: numbers and booleans \*/

case '-': case '0': case '1' : case '2': case '3' : case '4':

case '5': case '6': case '7' : case '8': case '9':

case 't': case 'f': case 'n' :

/\* And they must not be keys of the object \*/

if (tokens != NULL && parser->toksuper != -1) {

jsmntok\_t \*t = &tokens[parser->toksuper];

if (t->type == JSMN\_OBJECT ||

(t->type == JSMN\_STRING && t->size != 0)) {

return JSMN\_ERROR\_INVAL;

}

}

#else

/\* In non-strict mode every unquoted value is a primitive \*/

default:

#endif

r = jsmn\_parse\_primitive(parser, js, len, tokens, num\_tokens);

if (r < 0) return r;

count++;

if (parser->toksuper != -1 && tokens != NULL)

tokens[parser->toksuper].size++;

break;

#ifdef JSMN\_STRICT

/\* Unexpected char in strict mode \*/

default:

return JSMN\_ERROR\_INVAL;

#endif

}

}

if (tokens != NULL) {

for (i = parser->toknext - 1; i >= 0; i--) {

/\* Unmatched opened object or array \*/

if (tokens[i].start != -1 && tokens[i].end == -1) {

return JSMN\_ERROR\_PART;

}

}

}

return count;

}

/\*\*

\* Creates a new parser based over a given buffer with an array of tokens

\* available.

\*/

static void jsmn\_init(jsmn\_parser \*parser) {

parser->pos = 0;

parser->toknext = 0;

parser->toksuper = -1;

}

static int jsoneq(const char \*json, jsmntok\_t \*tok, const char \*s)

{

if (tok->type == JSMN\_STRING

&& (int) strlen(s) == tok->end - tok->start

&& strncmp(json + tok->start, s, tok->end - tok->start) == 0) {

return 0;

}

return -1;

}

static size\_t load\_file(char \*filename)

{

FILE \*f = fopen(filename, "rb");

if(f == NULL) {

fprintf(stderr, "Unable to open file %s\n", filename);

exit(1);

}

fseek(f, 0, SEEK\_END);

size\_t fsize = ftell(f);

fseek(f, 0, SEEK\_SET); //same as rewind(f);

if(fsize >= BUFF\_SIZE) {

fprintf(stderr, "Configuration file too large\n");

abort();

}

fread(config, fsize, 1, f);

fclose(f);

config[fsize] = 0;

return fsize;

}

int parse\_config(char \*path, struct configuration \*conf)

{

int i;

int r;

jsmn\_parser p;

jsmntok\_t t[128]; /\* We expect no more than 128 tokens \*/

load\_file(path);

jsmn\_init(&p);

r = jsmn\_parse(&p, config, strlen(config), t, sizeof(t)/sizeof(t[0]));

if (r < 0) {

printf("Failed to parse JSON: %d\n", r);

return 0;

}

/\* Assume the top-level element is an object \*/

if (r < 1 || t[0].type != JSMN\_OBJECT) {

printf("Object expected\n");

return 0;

}

/\* Loop over all keys of the root object \*/

for (i = 1; i < r; i++) {

if (jsoneq(config, &t[i], "host") == 0) {

/\* We may use strndup() to fetch string value \*/

conf->host = strndup(config + t[i+1].start, t[i+1].end-t[i+1].start);

i++;

} else if (jsoneq(config, &t[i], "username") == 0) {

conf->db\_username = strndup(config + t[i+1].start, t[i+1].end-t[i+1].start);

i++;

} else if (jsoneq(config, &t[i], "password") == 0) {

conf->db\_password = strndup(config + t[i+1].start, t[i+1].end-t[i+1].start);

i++;

} else if (jsoneq(config, &t[i], "port") == 0) {

conf->port = strtol(config + t[i+1].start, NULL, 10);

i++;

} else if (jsoneq(config, &t[i], "database") == 0) {

conf->database = strndup(config + t[i+1].start, t[i+1].end-t[i+1].start);

i++;

} else {

printf("Unexpected key: %.\*s\n", t[i].end-t[i].start, config + t[i].start);

}

}

return 1;

}

* Codice di appoggio, inout.c per l’interazione con l’utente

#include <unistd.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <ctype.h>

#include <termios.h>

#include <sys/ioctl.h>

#include <pthread.h>

#include <signal.h>

#include <stdbool.h>

#include "defines.h"

// Per la gestione dei segnali

static volatile sig\_atomic\_t signo;

typedef struct sigaction sigaction\_t;

static void handler(int s);

char \*getInput(unsigned int lung, char \*stringa, bool hide)

{

char c;

unsigned int i;

// Dichiara le variabili necessarie ad un possibile mascheramento dell'input

sigaction\_t sa, savealrm, saveint, savehup, savequit, saveterm;

sigaction\_t savetstp, savettin, savettou;

struct termios term, oterm;

if(hide) {

// Svuota il buffer

(void) fflush(stdout);

// Cattura i segnali che altrimenti potrebbero far terminare il programma, lasciando l'utente senza output sulla shell

sigemptyset(&sa.sa\_mask);

sa.sa\_flags = SA\_INTERRUPT; // Per non resettare le system call

sa.sa\_handler = handler;

(void) sigaction(SIGALRM, &sa, &savealrm);

(void) sigaction(SIGINT, &sa, &saveint);

(void) sigaction(SIGHUP, &sa, &savehup);

(void) sigaction(SIGQUIT, &sa, &savequit);

(void) sigaction(SIGTERM, &sa, &saveterm);

(void) sigaction(SIGTSTP, &sa, &savetstp);

(void) sigaction(SIGTTIN, &sa, &savettin);

(void) sigaction(SIGTTOU, &sa, &savettou);

// Disattiva l'output su schermo

if (tcgetattr(fileno(stdin), &oterm) == 0) {

(void) memcpy(&term, &oterm, sizeof(struct termios));

term.c\_lflag &= ~(ECHO|ECHONL);

(void) tcsetattr(fileno(stdin), TCSAFLUSH, &term);

} else {

(void) memset(&term, 0, sizeof(struct termios));

(void) memset(&oterm, 0, sizeof(struct termios));

}

}

// Acquisisce da tastiera al più lung - 1 caratteri

for(i = 0; i < lung; i++) {

(void) fread(&c, sizeof(char), 1, stdin);

if(c == '\n') {

stringa[i] = '\0';

break;

} else

stringa[i] = c;

// Gestisce gli asterischi

if(hide) {

if(c == '\b') // Backspace

(void) write(fileno(stdout), &c, sizeof(char));

else

(void) write(fileno(stdout), "\*", sizeof(char));

}

}

// Controlla che il terminatore di stringa sia stato inserito

if(i == lung - 1)

stringa[i] = '\0';

// Se sono stati digitati più caratteri, svuota il buffer della tastiera

if(strlen(stringa) >= lung) {

// Svuota il buffer della tastiera

do {

c = getchar();

} while (c != '\n');

}

if(hide) {

//L'a capo dopo l'input

(void) write(fileno(stdout), "\n", 1);

// Ripristina le impostazioni precedenti dello schermo

(void) tcsetattr(fileno(stdin), TCSAFLUSH, &oterm);

// Ripristina la gestione dei segnali

(void) sigaction(SIGALRM, &savealrm, NULL);

(void) sigaction(SIGINT, &saveint, NULL);

(void) sigaction(SIGHUP, &savehup, NULL);

(void) sigaction(SIGQUIT, &savequit, NULL);

(void) sigaction(SIGTERM, &saveterm, NULL);

(void) sigaction(SIGTSTP, &savetstp, NULL);

(void) sigaction(SIGTTIN, &savettin, NULL);

(void) sigaction(SIGTTOU, &savettou, NULL);

// Se era stato ricevuto un segnale viene rilanciato al processo stesso

if(signo)

(void) raise(signo);

}

return stringa;

}

// Per la gestione dei segnali

static void handler(int s) {

signo = s;

}

bool yesOrNo(char \*domanda, char yes, char no, bool predef, bool insensitive)

{

// I caratteri 'yes' e 'no' devono essere minuscoli

yes = tolower(yes);

no = tolower(no);

// Decide quale delle due lettere mostrare come predefinite

char s, n;

if(predef) {

s = toupper(yes);

n = no;

} else {

s = yes;

n = toupper(no);

}

// Richiesta della risposta

while(true) {

// Mostra la domanda

printf("%s [%c/%c]: ", domanda, s, n);

char c;

getInput(1, &c, false);

// Controlla quale risposta è stata data

if(c == '\0') { // getInput() non può restituire '\n'!

return predef;

} else if(c == yes) {

return true;

} else if(c == no) {

return false;

} else if(c == toupper(yes)) {

if(predef || insensitive) return true;

} else if(c == toupper(yes)) {

if(!predef || insensitive) return false;

}

}

}

char multiChoice(char \*domanda, char choices[], int num)

{

// Genera la stringa delle possibilità

char \*possib = malloc(2 \* num \* sizeof(char));

int i, j = 0;

for(i = 0; i < num; i++) {

possib[j++] = choices[i];

possib[j++] = '/';

}

possib[j-1] = '\0'; // Per eliminare l'ultima '/'

// Chiede la risposta

while(true) {

// Mostra la domanda

printf("%s [%s]: ", domanda, possib);

char c;

getInput(1, &c, false);

// Controlla se è un carattere valido

for(i = 0; i < num; i++) {

if(c == choices[i])

return c;

}

}

}

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