Basi di Dati e Conoscenza

Progetto A.A. 2019/2020

Sistema di gestione di una pizzeria

0244242

Matteo Conti

Indice

1. Descrizione del Minimondo

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.

24	Il manager ha la possibilità di stampare lo scontrino di un ordine. Inoltre, per motivi
25	statistici, ha la possibilità di visualizzare le entrate giornaliere e/o mensili.

2. 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.		Ordine.

	L'insieme dei prodotti definisce il menù.		
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.

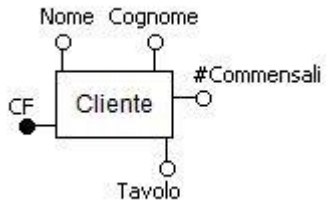
Frasi relative al Cliente:
All’ingresso di un cliente, il manager lo riceve e lo registra, segnando nome, cognome e numero di commensali, assegnandogli un tavolo disponibile in grado di ospitarli tutti.
Frasi relative al Turno:
La pizzeria opera 24/7, ma per motivi di risparmio, in alcuni turni sono 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

3. Progettazione concettuale

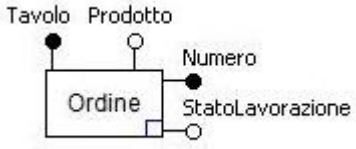
Costruzione dello schema E-R

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):

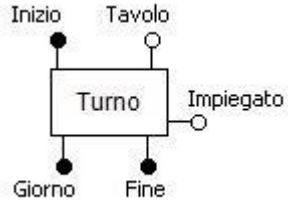
1. Per il cliente è importante sapere a quale tavolo viene assegnato in una prima fase è stato perciò messo un attributo che verrà in seguito reificato.



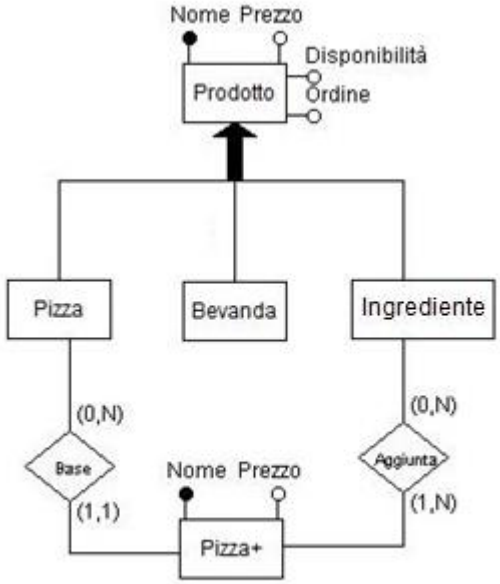
2. Come per il cliente anche qui il prodotto e il tavolo che sono strettamente legati all'ordine sono riportati come attributi che verranno poi reificati.



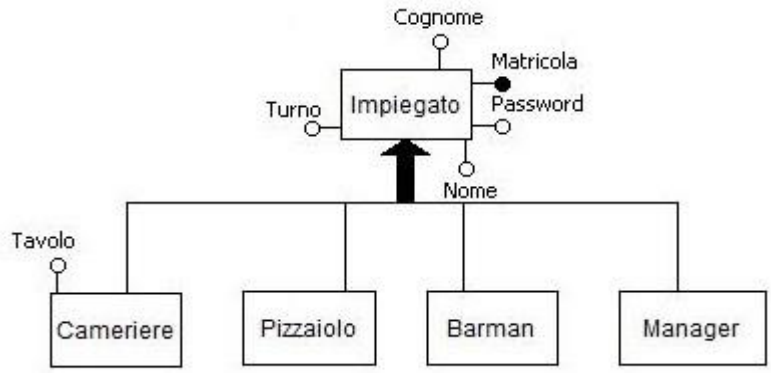
3. Per il turno si è scelto di mettere orario di inizio, orario di fine e giorno in quanto è prevedibile che i turni di diverse tipologie di impiegato inizino allo stesso orario ma finiscano in orari diversi (o viceversa) cosa che solo con giorno ed orario di inizio (o ora di fine) non sarebbe rappresentabile.



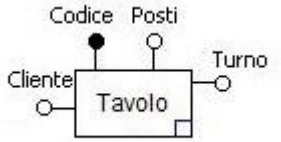
4. Per il concetto di prodotto si è scelto di specificarlo in 3 tipologie di prodotti che sono bevande, ingredienti aggiuntivi per le pizze per le pizze e pizze. Gli ultimi due formano quella che viene chiamata “Pizza+” che rappresenta una pizza con eventuali ingredienti aggiuntivi.



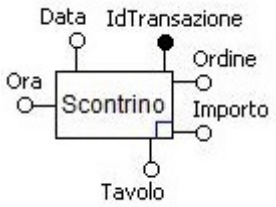
5. Per l’impiegato si è scelto di mettere una matricola piuttosto che il codice fiscale in quanto in genere in un luogo di lavoro ad un impiegato viene associata una matricola per l’identificazione. Per esplicitare concettualmente le tipologie di utenti che utilizzano l’applicazione si è scelto di mettere 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.



6. 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).

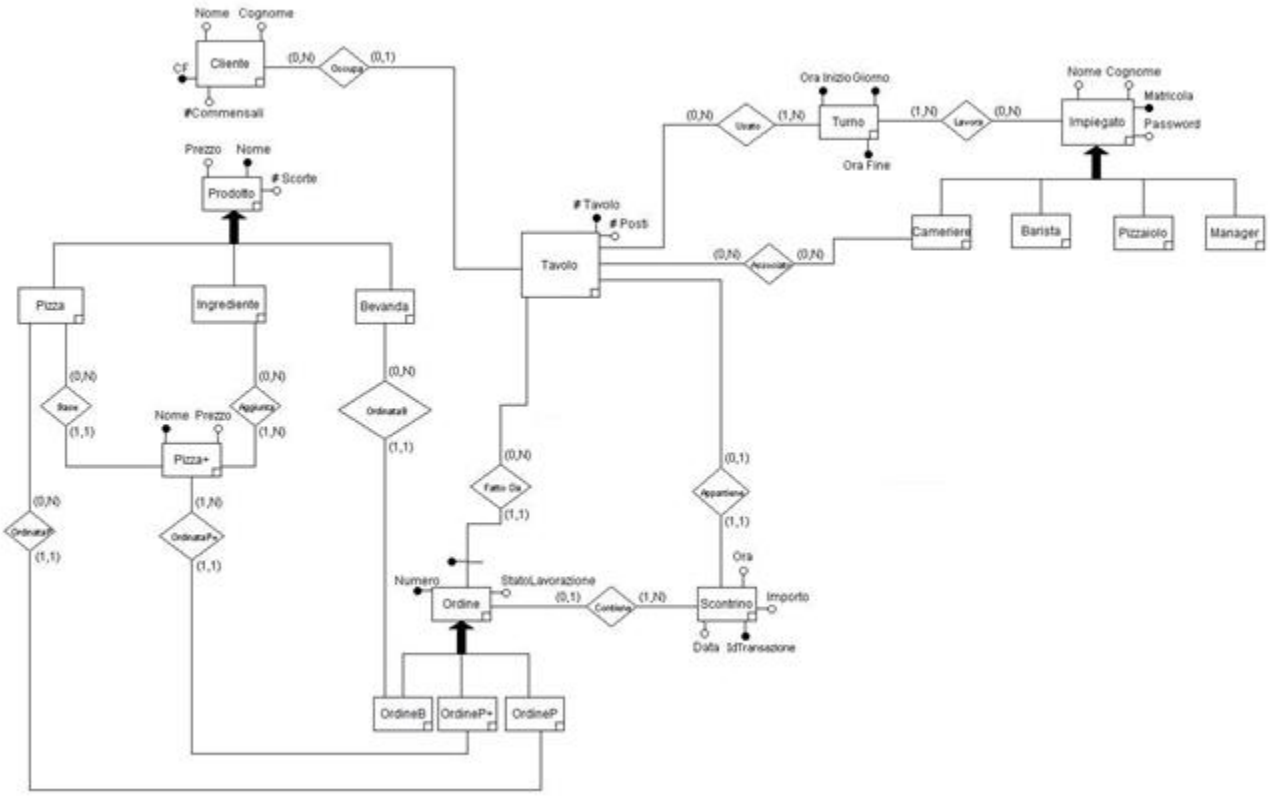


7. Nella specifica si parla di voler stampare uno scontrino, esso è utile anche ai fini di contabilità che è un altro elemento chiave della specifica per questo motivo è stato modellato come entità, si è scelto anche se "superfluo" di mettere un identificatore fittizio in quanto tipicamente gli scontrino 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).



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

4. 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]	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

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

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	S L S S S	(2+1+2+2+2+2+1+2)*10=140 accessi al giorno, assumendo che si aggiunga un solo ingrediente.

	1 accessi ad Aggiunta 1 accesso a Scontrino 1 accesso a Scontrino	S S L	
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.

2. 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)

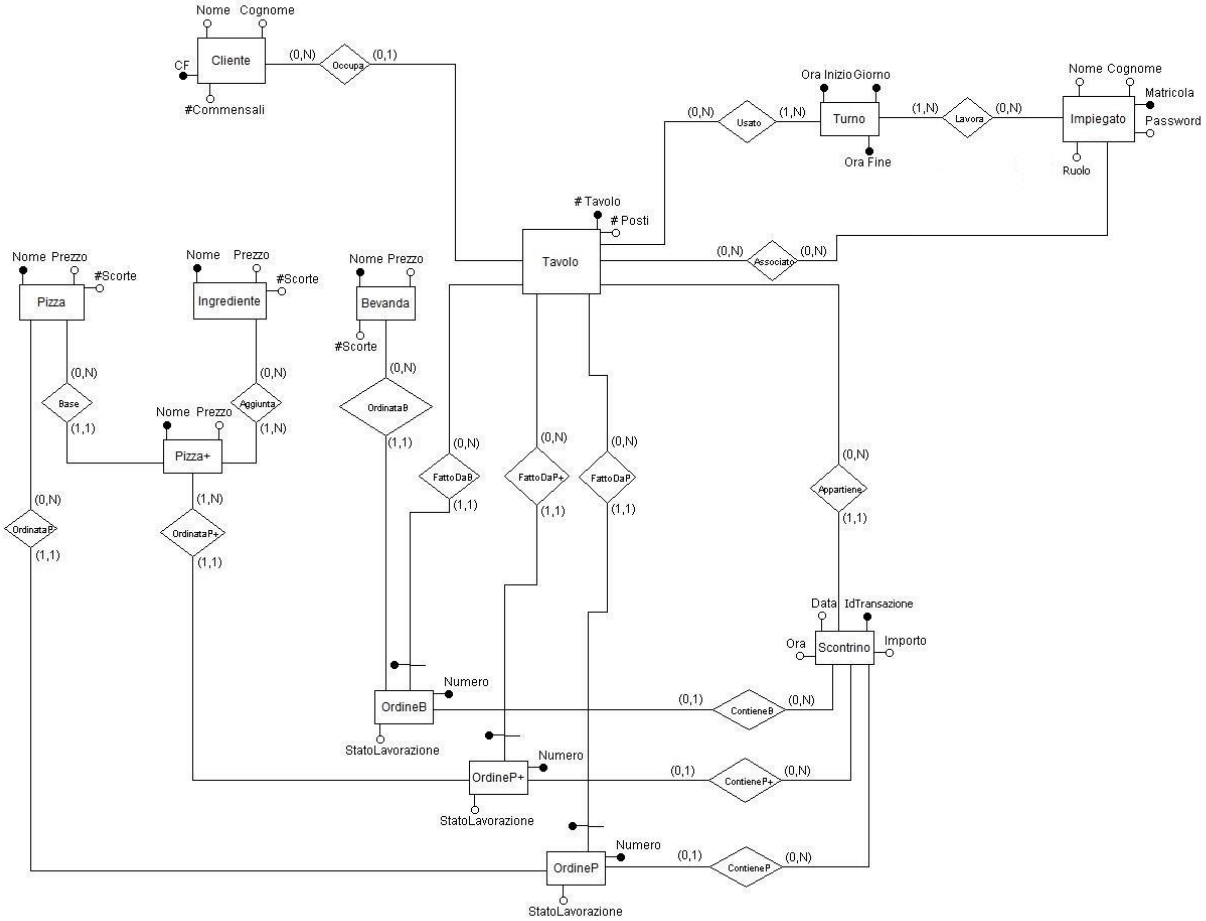
OrdineP(Pizza) ⊆ Pizza(Nome)

OrdinePizzaPlus(PizzaPlus) ⊆ PizzaPlus(Nome)

PizzaPlus(Pizza) ⊆ Pizza(Nome)

Aggiunta(PizzaPlus) ⊆ PizzaPlus(Nome)

Aggiunta(Ingrediente) ⊆ Ingrediente(Nome)



Normalizzazione del modello relazionale

Le relazioni sono tutte in 3NF.

5. 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]
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

[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.

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	IDX

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.	
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;
                }
        }
}
```