Basi di Dati
Progetto A.A. 2021/2022

Basi di Dati:
Pizzeria

0270988
Matteo Federico

Indice

1. Descrizione del Minimondo

Sistema di gestione di una pizzeria.
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, pizzaiolo hanno sempre la possibilità di visualizzare cosa debbono preparare, in ordine di ricezione della comanda. Quando hanno preparato una bevanda o una pizza, il cameriere può visualizzare cosa è pronto (in relazione agli ordini) e sapere cosa deve consegnare a quale tavolo.

La pizzeria opera 24/7, ma per motivi di risparmio, in alcuni giorni sono disponibili un numero differente di camerieri e vengono utilizzati un numero differente di tavoli. Il manager può definire quali camerieri lavorano in quali turni e quali tavoli sono utilizzati in quali turni. Il menu è unico per tutti i turni e definito dal manager, con i rispettivi prezzi. Nel menu è necessario anche prevedere aggiunte per le pizze (ad esempio, un cliente potrebbe voler aggiungere del tonno ad una pizza quattro formaggi), con i relativi costi.

Allo stesso modo, il manager ha la possibilità di tenere traccia delle disponibilità dei singoli prodotti. In questo modo, se viene ordinato ad un cameriere da un cliente un prodotto che non è disponibile, questo non potrà essere aggiunto all'ordine.

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

2. Analisi dei Requisiti

Identificazione dei termini ambigui e correzioni possibili

Linea	Termine	Nuovo termine	Motivo correzione
15	Cosa	Comanda	Il cameriere può vedere quando una comanda è pronta per essere consegnata
23	Prodotti	Ingredienti	Può causare confusione tra il prodotto che un cliente può ordinare, cioè quello sul menu, e l'ingrediente che compone il prodotto del menu

Specifica disambiguata
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, 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 le comande pronte (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 Ingredienti.
In questo modo, se viene ordinato ad un cameriere da un cliente un prodotto che non è disponibile, questo non potrà essere aggiunto all'ordine.

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

Glossario dei Termini

Termine	Descrizione	Sinonimi	Collegamenti

Cliente	Persona che occupa un tavolo, e ordina pizze e bevande, in un determinato momento		Tavolo, Cameriere
Manager	Lavoratore della pizzeria che si occupa di tener traccia degli ingredienti, di assegnare i turni ai camerieri e ai tavoli e di accogliere i clienti quando arrivano		Cliente, Cameriere, tavolo, Ingredienti
Cameriere	Lavoratore della pizzeria che si occupa di servire i clienti dei tavoli a lui assegnati durante un determinato turno		Tavolo, Turno
Tavolo	Tavolo posseduto dalla pizzeria che può essere usato o meno durante un determinato turno		Cameriere, Cliente, Turno
Turno	Turni di lavoro della pizzeria decisi dal manager		Cameriere, Manager, Tavolo
Comanda	L'insieme dei prodotti ordinati da uno stesso cliente in un determinato momento della cena	ordine	Scontrino, Cliente, Prodotti
Scontrino	L'insieme delle comande ordinate da un cliente		Comande, Clienti
Prodotti	Prodotti sul menu che sono venduti dalla pizzeria		Ingredienti, Pizze, Bevande
Ingredienti	Ingredienti che vengono usati dalla pizzeria per creare i Prodotti sul menu		Prodotti, Aggiunte
Aggiunte	Ingredienti che possono essere aggiunti alle pizze con un costo extra		Ingredienti, Pizze
Pizza	Tipo di Prodotto venduta dalla pizzeria cui può essere modificato con delle aggiunte		Prodotto, Comanda, Aggiunta
Bevanda	Tipo di Prodotto venduta dalla pizzeria		Prodotto, Comanda

Raggruppamento dei requisiti in insiemi omogenei

Frasi relative a Cliente
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.

Frasi relative a Cameriere
Un cameriere ha sempre la possibilità di visualizzare quali tavoli a lui assegnati sono occupati e quali sono stati serviti.

Il cameriere può visualizzare cosa è pronto (in relazione agli ordini) e sapere cosa deve consegnare a quale tavolo.
In alcuni giorni sono disponibili un numero differente di camerieri

Frasi relative a Tavolo
All'ingresso di un cliente, il Manager […] assegnando un tavolo disponibile in grado di ospitarli tutti.
In alcuni giorni […] vengono utilizzati un numero differente di tavoli

Frasi relative a Comanda
Barista, pizzaiolo hanno sempre la possibilità di visualizzare cosa debbono preparare, in ordine di ricezione della comanda.

Frasi relative a Manager
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.
All'ingresso di un cliente, il manager lo riceve e lo registra
Il manager può definire quali camerieri lavorano in quali turni e quali tavoli sono utilizzati in quali turni
Il manager ha la possibilità di tenere traccia delle disponibilità dei singoli Ingredienti.

Frasi relative a Ingredienti
possibilità di tenere traccia delle disponibilità dei singoli Ingredienti.

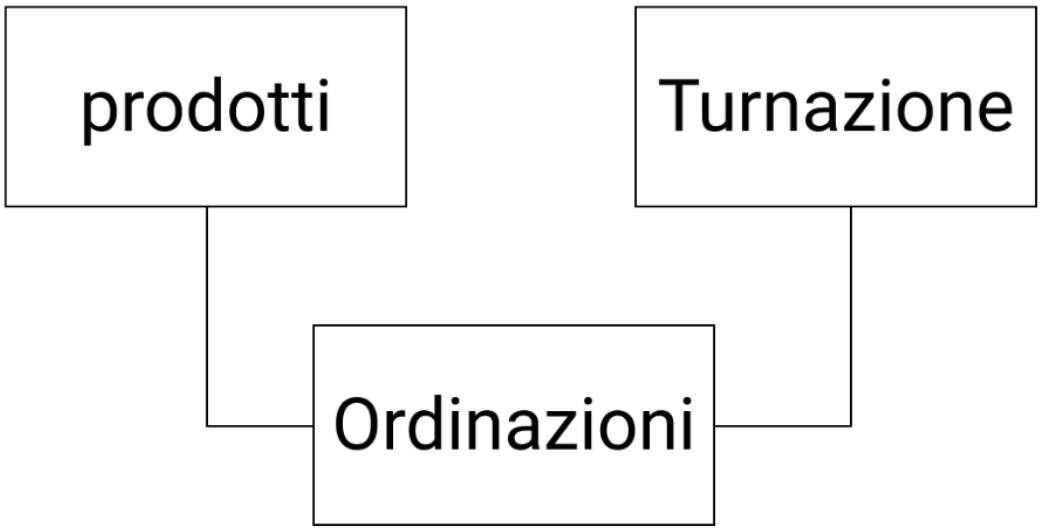
Frasi relative a Aggiunte
Nel menu è necessario anche prevedere aggiunte per le pizze (ad esempio, un cliente potrebbe voler aggiungere del tonno ad una pizza quattro formaggi), con i relativi costi.

Frasi relative a Prodotti
Se viene ordinato ad un cameriere da un cliente un prodotto che non è disponibile, questo non potrà essere aggiunto all'ordine.

3. Progettazione concettuale

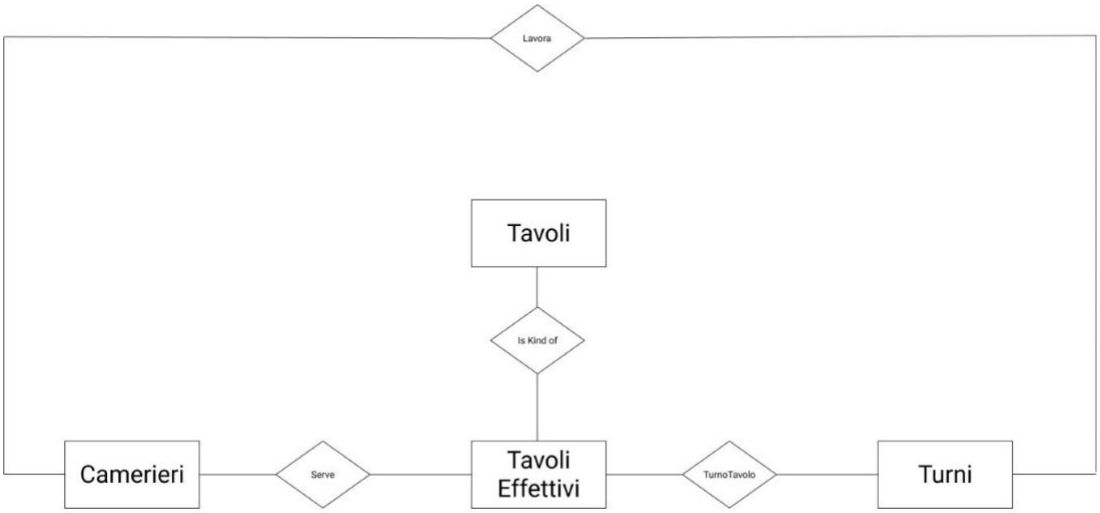
Costruzione dello schema E-R

Per la creazione dello Schema ER ho adoperato una strategia mista cercando quindi di creare un iniziale schema scheletro individuando tre macro-concetti: i prodotti che offre la pizzeria, le ordinazioni dei clienti e la turnazione dei camerieri e dei tavoli.

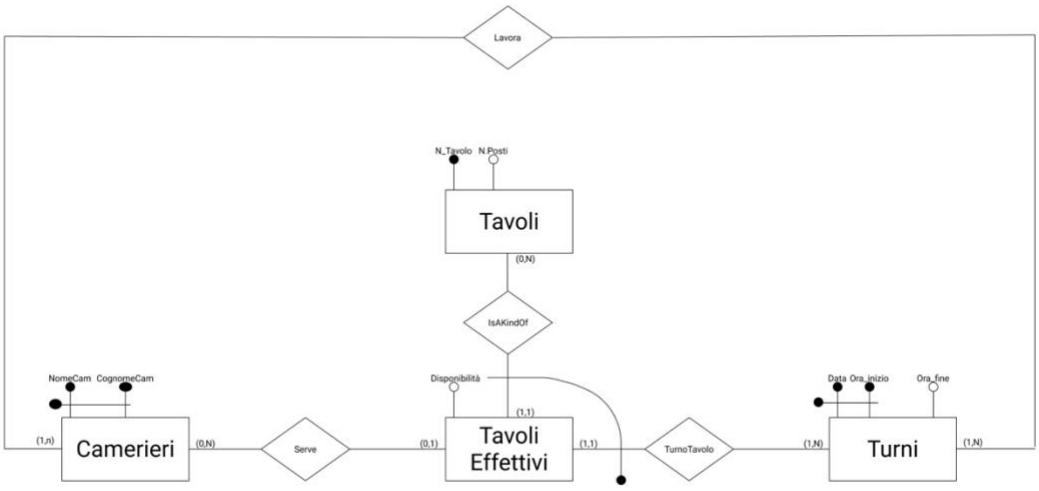


Ho quindi diviso il lavoro in 3 fasi per ogni Macro-Concetto:
La prima fase è creare una sorta di schema scheletro per il concetto contenente le entità che rappresentavano il concetto e le relazioni che collegavano le varie entità.
La seconda fase è stata che per ogni concetto ho raffinato ulteriormente lo schema andando a concentrarmi sugli attributi delle entità e sulle cardinalità delle relazioni.
L'ultima fase è stata quella di unire i vari schemi creati cercando di ottenere un unico schema che rappresentasse a pieno la specifica fornita

Il primo macro-concetto è la turnazione, nella specifica ho individuato due Concetti fondamentali a cui bisogna associare il concetto di turno: i camerieri e i tavoli. Poiché sia i camerieri sia i tavoli hanno necessità di avere un turno e un cameriere può avere più tavoli assegnati durante uno stesso turno.

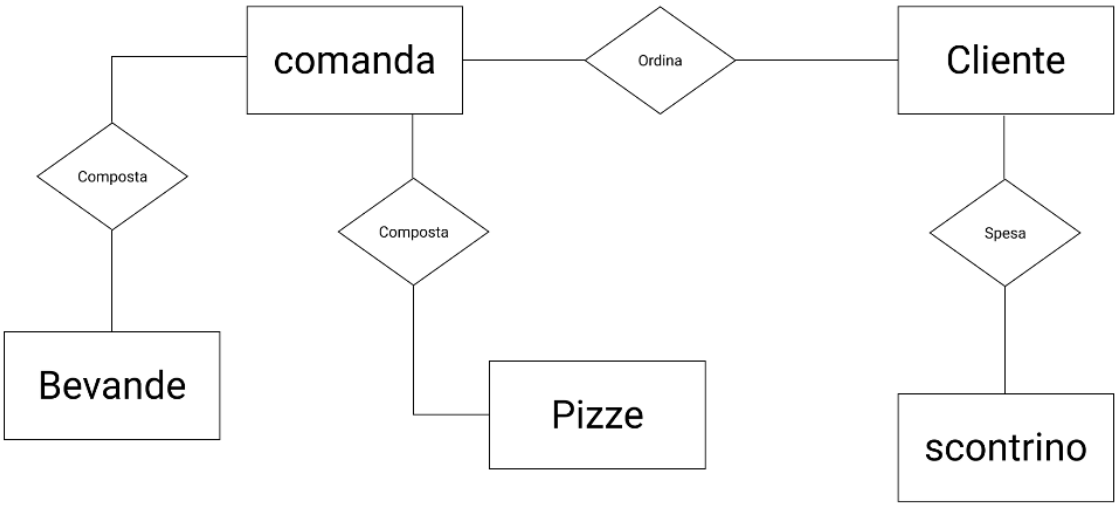


Ovviamente una delle regole aziendali deve imporre il vincolo che il cameriere è di turno nello stesso turno al quale sono assegnati i suoi tavoli effettivi.
In un raffinamento ulteriore si individua che il cameriere deve essere individuato dal suo nome e dal suo cognome e che i tavoli devono avere un attributo fittizio per distinguerli. In più, ogni tavolo è caratterizzato dal numero di posti che può contenere. Il turno deve essere individuato dalla data e l'ora di inizio. Il tavolo effettivo che rappresenta il tavolo in un determinato turno deve essere caratterizzato dalla sua disponibilità e dal tavolo e dal turno a cui si riferisce.

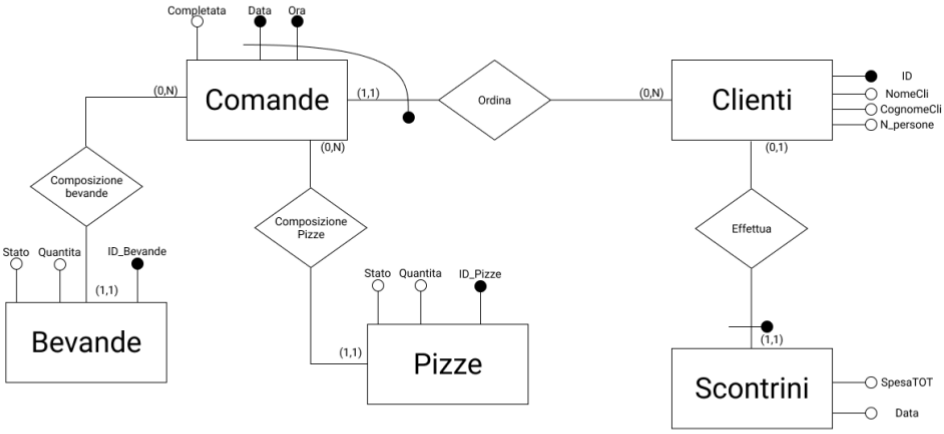


In seguito, sono andato a raffinare le ordinazioni, elemento che raggruppa in sé il concetto di cliente, di comanda, di scontrino e i due concetti di pizze e bevande che vanno a comporre la comanda. Ho

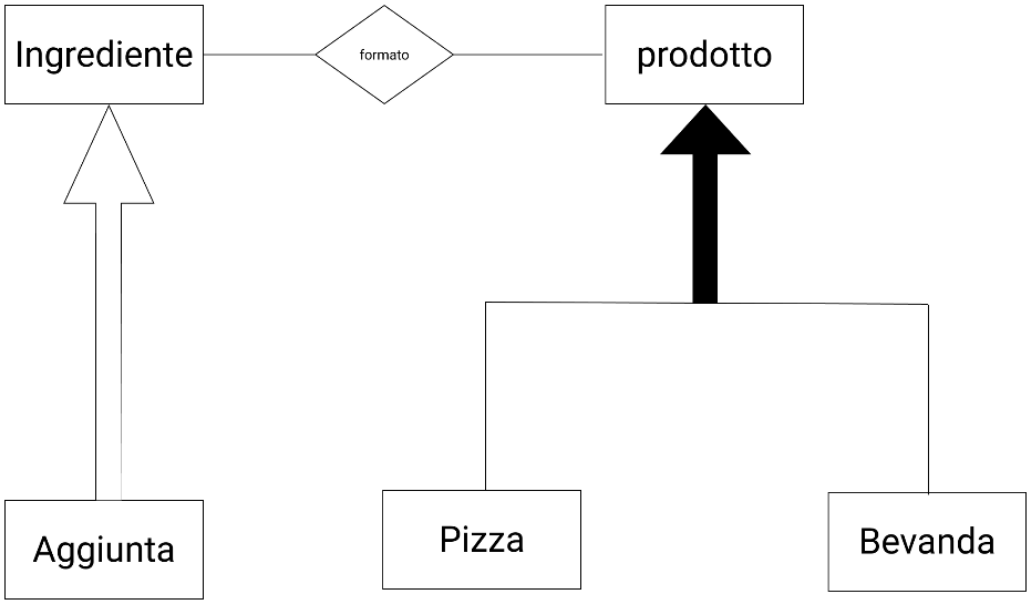
deciso di interpretare la comanda come un'ordinazione singola del cliente, ogni cliente quindi può fare più comande.



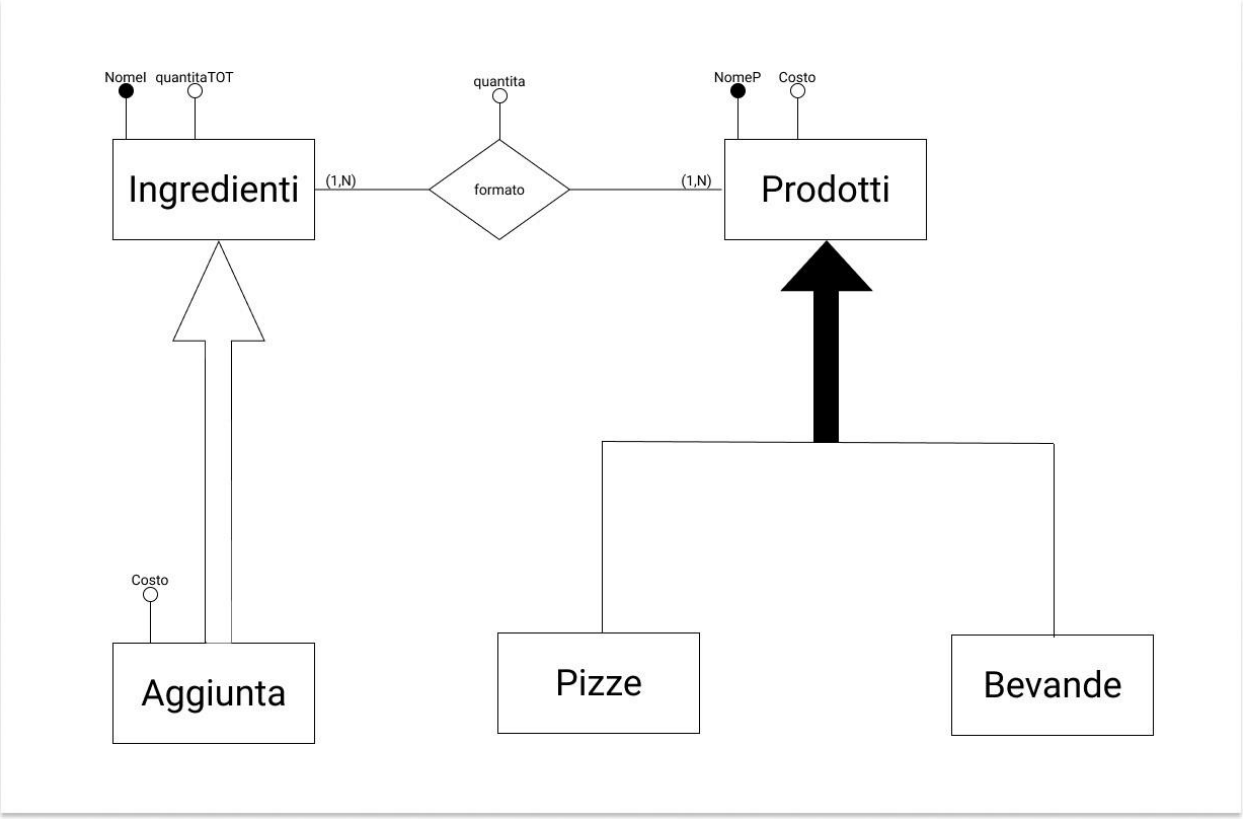
Andando a raffinare ulteriormente questo schema ho deciso di caratterizzare l'entità cliente con il nome, il cognome e il numero di commensali, dato che un cliente può tornare più volte a mangiare alla pizzeria ho deciso di identificare il cliente tramite un attributo fittizio ID. La comanda è caratterizzata dal suo stato, cioè se è stata completata o meno, dalla data e dall'ora, quest'ultimi insieme al cliente che effettua l'ordine la identificano. L'entità scontrino è stata caratterizzata dal totale della spesa e dalla data di emissione e viene identificata dal cliente che effettua la cena, quindi ogni cliente può effettuare un solo scontrino.
La comanda è composta dalle bevande e dalle pizze, tutte e due queste entità sono caratterizzate da uno stato, dalla quantità e da un identificatore.



L'ultimo macro-concetto è prodotti di cui bisogna tener traccia degli ingredienti che li compongono e di quali ingredienti sono aggiunte, inoltre bisogna differenziare i tipi di prodotti che si hanno in Bevande e in Pizze.

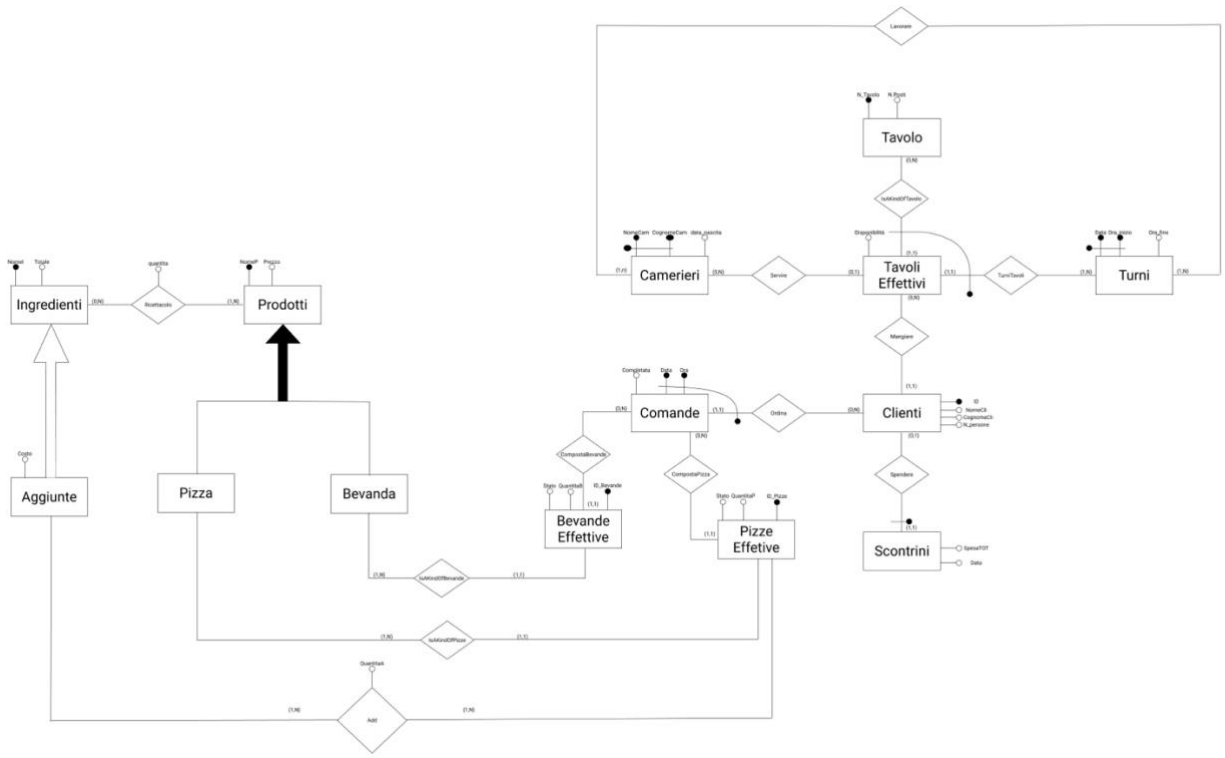


Nella raffinazione successiva ho dato all'entità ingredienti un nome, che identifica lo specifico ingrediente, e la quantità che rimane in pizzeria, ho identificato il prodotto con un nome, il suo costo e la descrizione degli allergeni che può contenere, le aggiunte sono estensioni degli ingredienti che hanno anche un costo, ogni prodotto è composto da N ingredienti con quantità diverse in base alla ricetta del prodotto.



Integrazione finale

Una volta finito il raffinamento delle tre macro-entità ho iniziato a capire come le entità raffinate potessero essere associate tra loro. Ovviamente inizio con il collegare l'entità pizza e l'entità bevanda della macro entità prodotti con le pizze e le bevande della macro Entità ordinazione (che verranno rinominate per non creare confusione con il nome di Pizze effettive e bevande Effettive) facendo ben attenzione a considerarle entità separate poiché in un caso rappresentano i prodotti astratti che sono nel menù, nell'altro rappresentano la concretizzazione di essi che viene portata al cliente, inoltre, il cliente può fare delle aggiunte alle pizze, per tanto serve un'associazione anche tra aggiunte e pizze effettive. Invece tra il macro-concetto di Turnazione e il macro-concetto di ordinazione ci deve essere un'associazione tra il tavolo effettivo e il cliente. Un cliente consuma su un tavolo effettivo e un tavolo in un determinato turno può avere un cliente che lo usa. Perciò dopo tutte queste considerazioni lo schema E-R finale che ho elaborato è il seguente:



Regole aziendali

Laddove la specifica non sia catturata in maniera completa dallo schema E-R, corredare lo stesso in questo paragrafo con l'insieme delle regole aziendali necessarie a completare la progettazione concettuale.

1. I Tavoli di Turno possono essere serviti solo dai Camerieri che lavorano durante quel turno
2. I Clienti possono mangiare solo su tavoli al quale vi è assegnato un cameriere

Dizionario dei dati

Entità	Descrizione	Attributi	Identificatori
Ingredienti	Gli ingredienti presenti dentro la pizzeria	NomeI Totale	NomeI
Aggiunta	Gli ingredienti che possono essere anche usate come aggiunte per le pizze se il cliente lo richiede	costo	NomeI
Prodotto	Prodotto presente nel menù della pizzeria	NomeP Prezzo	NomeP
Cliente	Persona che prenota e consuma in pizzeria	Nome, Cognome, ID, N_Persone	ID
Scontrino	Entità che permette di immagazzinare il prezzo totale spesso da un cliente	SpesaToT, Data	ID
Pizza	Prodotti di tipo pizza che la pizzeria può cucinare e presenti nel menù		NomeP

Bevanda	Prodotti di tipo Bevanda che la pizzeria può preparare e presenti nel menù		NomeP
Comanda	Insieme degli ordini fatti da un Cliente	Data, Ora, Completata	ID,Data,Ora
PizzaEffettiva	Le pizze effettivamente prodotte dalla pizzeria	QuantitaP, Stato	ID_Pizze
BevandaEffettiva	Le bevande effettivamente preparate dalla pizzeria	QuantitaB, Stato	ID_Bevande
Tavolo	Tavolo presente nella pizzeria che può essere messo a disposizione durante i turni	N_Tavolo, N_persone	N_Tavolo
Cameriere	Persona che lavora dentro la pizzeria	NomeC, CognomeC,	NomeC, CognomeC
Turno	Turni di lavoro della pizzeria, gli stessi turni valgono sia per i camerieri che per i tavoli	Data, OraInzio OraFine	Data, OraInizio
TavoloEffettivo	Tavolo Effettivamente utilizzato durante uno specifico turno dove può accomodarsi un cliente se disponibile	Disponibilità	N_Tavolo, Data, OraInizio,

4. Progettazione logica

Volume dei dati

Ho supposto di avere un totale di 90-110 clienti al giorno, ogni cliente porta con sé 3 persone e ogni cliente ordina in media 3 pizze e 3 bevande, ogni cliente fa circa una media di 1 comanda e mezza e ogni cliente fa uno e un solo scontrino.
Per i turni ho considerato 3 turni al giorno di 8 ore e che siano gli stessi tra i tavoli e i camerieri, inoltre ho deciso di tener traccia dei turni passati per un anno e di memorizzare le comande e le pizze per non più di un mese.
Gli scontrini e i clienti vengono memorizzati per un anno, i prodotti, i tavoli, i camerieri, gli ingredienti e le aggiunte non si dispone una data di eliminazione perché suppongo che rimangano invariati come espressamente detto dal testo.

Concetto nello schema	Tipo[1]	Volume atteso
Ingredienti	E	60-70
Prodotto	E	20-40
Aggiunte	E	10-20
Pizze	E	10-20
Bevande	E	10-20
PizzeEffettive	E	8400-10300
BevandeEffettive	E	8400-10300
Comande	E	4900-5200
Cliente	E	34000-41000
Scontrini	E	34000-41000
TavoliEffettivi	E	33000-44000
Tavoli	E	30-40
Turni	E	1100
Camerieri	E	15-20
Formato	R	80-160
isAKindOfPizze	R	8400-10300
isAKindOfBevande	R	8400-10300
add	R	3000-6000
CompostaB-C	R	8400-10300
CompostaP-C	R	8400-10300
Spesa	R	34000-41000
Ordina	R	4900-5200
Serve	R	33000-44000
Cameriere-Turno	R	4900-5200
IsAKindOfTavolo	R	33000-44000

Tavola delle operazioni

Rappresentare nella tabella sottostante tutte le operazioni sulla base di dati che devono essere supportate dall'applicazione, con la frequenza attesa. Le operazioni da supportare devono essere desunte dalle specifiche raccolte.

Op1. Registrazione di un nuovo cliente
Op2. Registrazione di una nuova comanda
Op3. Visualizzare i tavoli assegnati ad uno specifico cameriere
Op4. Visualizzare le pizze da preparare in ordine d'arrivo

Op5. Visualizzare le bevande da preparare in ordine d'arrivo
Op6. Visualizzare le comande pronte da portare al cliente
Op7. Visualizzare il conto di un cliente
Op8. Visualizzare il guadagno di un giorno
Op9. Visualizzare il guadagno di un mese
Op10. Visualizzare gli ingredienti
Op11. Visualizzare aggiunte per le pizze

Cod.	Descrizione	Frequenza attesa
OP1	Registrazione di un nuovo cliente da parte del manager all'interno del sistema e assegnazione di un tavolo disponibile	90/giorno-110/giorno
OP2	Inserimento all'interno del sistema di una nuova comanda da parte di un cameriere	130/giorno-150/giorno
OP3	Visualizzazione da parte di un cameriere dei tavoli a lui assegnati	40/giorno-60/giorno
OP4	Visualizzazione da parte del pizzaiolo delle pizze che deve preparare in ordine di arrivo	130/giorno-150/giorno
OP5	Visualizzazione da parte del barista delle bevande che deve preparare in ordine di arrivo	130/giorno-150/giorno
OP6	Visualizzazione da parte dei camerieri delle comande pronte che devono essere portata ai tavoli	600/giorno-800/giorno
OP7	Visualizzazione dello scontrino di un cliente	90/giorno-110/giorno
OP8	Visualizzazione da parte del manager del guadagno di un giorno	1/giorno
OP9	Visualizzazione da parte del manager del guadagno di un mese	1/mese
OP10	Visualizzazione da parte del manager dello stato delle quantità degli ingredienti da comprare	1/settimana
OP11	Visualizzare da parte del cameriere le possibili aggiunte da mettere sopra le pizze	130/giorno-150/giorno

Costo delle operazioni

Operazione 1
Considerando il caso il nostro sistema debba trovare un tavolo disponibile nel giorno corrente in un determinato turno e va bene scegliere il primo libero abbastanza capiente per contenere il nuovo cliente, allora se la tabella di tavoli effettivi ha un volume atteso di 38 000 elementi ci vorranno in media 19000 accessi in lettura e 1 accesso in scrittura per registrare un cliente.

Operazione 1			
Concetto	Costr	Accesso	Tipo
TavoliEffettivi	E	19000	L
Cliente	E	1	S

Totale:1710000-2090000 accessi al giorno

Operazione 2
Considerando il caso in cui abbiamo una media di 3 pizze e 3 bevande a comanda e che solo una pizza su 3 abbia la possibile aggiunta.

Operazione 2			
Concetto	Costr	Accesso	Tipo
Comanda	E	1	S
PizzeEffettive	E	3	S
BevandeEffettive	E	3	S
Add	R	1	S

Totale:1040-1200 accessi al giorno

Operazione 3
Considerando, come nell'operazione 1, che in media ci siano 38000 elementi in tavoli effettivi e che ogni cameriere abbia assegnato a turno circa 7 tavoli per trovarli tutti deve scorrere completamente la lista dei Tavoli Effettivi

Operazione 3			
Concetto	Costr	Accesso	Tipo
TavoliEffettivi	E	38000	L

Totale:1520000-2280000 accessi al giorno

Operazione 4
Considerando che il volume medio delle pizze da preparare è di 9400 elementi e che per ogni pizza bisogna vedere anche le possibili aggiunte abbiamo che bisogna accedere sia a tutte le pizze effettive per vedere se siano da preparare e sia vedere se le pizze da preparare abbiano qualche aggiunta o meno

Operazione 4			
Concetto	Costr	Accesso	Tipo
PizzeEffettive	E	9400	L
Add	R	4500	L

Totale:1807000-2085000 accessi al giorno

Operazione 5
Considerando che il volume medio delle bevande da preparare è di 9400 elementi

Operazione 5			
Concetto	Costr	Accesso	Tipo
BevandeEffettive	E	9400	L

Totale:1034000-1410000 accessi al giorno

Operazione 6
Si stima che in media il volume atteso delle comande totali sia di 5000 elementi, quindi bisogna verificare lo stato di ogni singola comanda per vedere se è pronta oppure no

Operazione 6			
Concetto	Costr	Accesso	Tipo
Comande	E	5000	L

Totale:2000000-3000000 accessi al giorno

Operazione 7
Considerando che in media ogni cliente ordina 3 pizze e 3 bevande e che solo una di esse ha una sola aggiunta bisogna vedere il cliente seduto a quale tavolo, leggere le comande a lui relative e vedere le pizze, le aggiunte e le bevande da lui consumate e inserire la spesa totale nel sistema

Operazione 7			
Concetto	Costr	Accesso	Tipo

Cliente	E	40000	L
Comanda	E	5000	L
PizzeEffettive	E	9400	L
Add	R	4500	L
BevandeEffettive	E	9400	L
Scontrino	E	1	S

Totale: 6000000-8000000 accessi al giorno

Operazione 8
Bisogna vedere tutti gli scontrini, verificare che siano del giorno corrente e fare la somma per vedere l'incasso giornaliero

Operazione 8			
Concetto	Costr	Accesso	Tipo
Scontrini	E	37000	L

Totale:37000 accessi al giorno

Operazione 9
Come nell' operazione 8 bisogna vedere tutti gli scontrini e verificare che siano del mese che si vuole visualizzare l'incasso

Operazione 9			
Concetto	Costr	Accesso	Tipo
Scontrini	E	37000	L

Totale: 37000 accessi al mese

Operazione 10
In totale ci saranno tra i 40 e i 60 ingredienti in tutto nel sistema una media approssimativa è quindi di 50 accessi in lettura per visualizzare tutti gli ingredienti

Operazione 10			
Concetto	Costr	Accesso	Tipo
Ingredienti	E	50	L

Totale: 50 accessi alla settimana

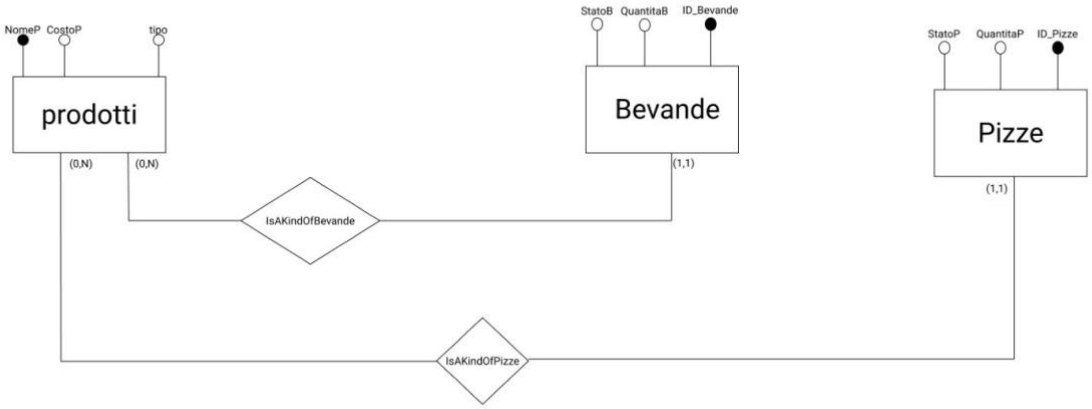
Operazione 11
Considerando che ci saranno circa tra le 10 e le 20 aggiunte una media approssimativa è quindi di 15 ci vorranno 15 accessi in lettura per stampare tutte le aggiunte

Operazione 11			
Concetto	Costr	Accesso	Tipo
Aggiunte	E	15	L

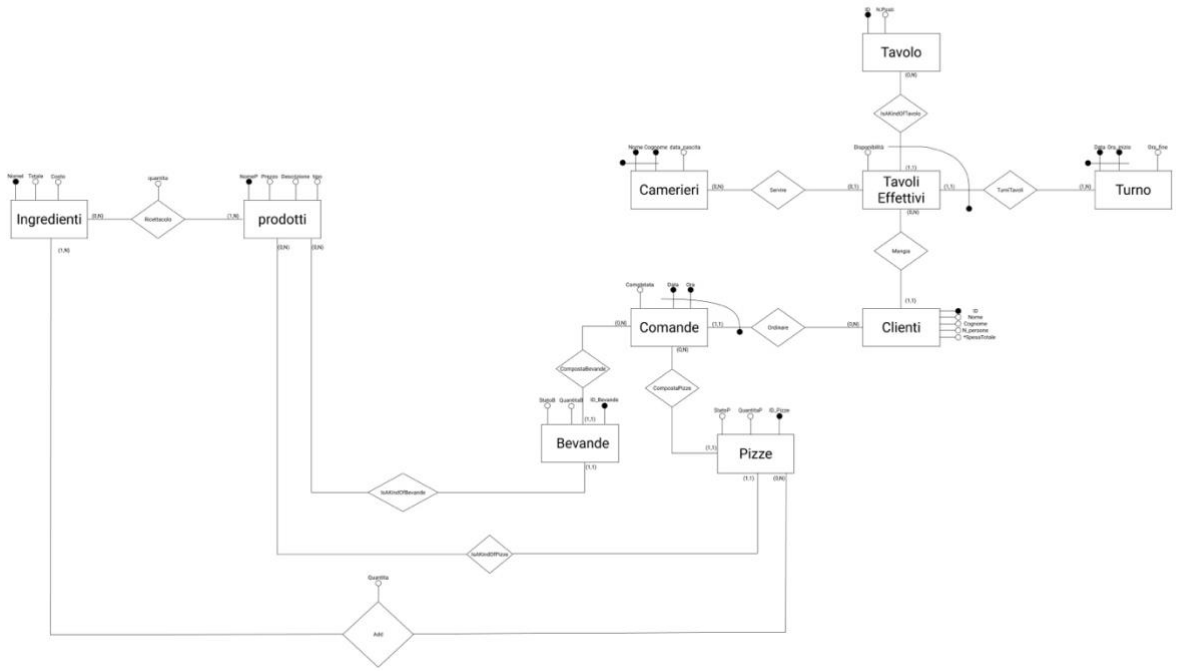
Totale: 1950-2250 accessi al giorno

Ristrutturazione dello schema E-R

Per eliminare la generalizzazione dei Prodotti in pizze e bevande decido di aggiungere su prodotti un attributo che mi indichi se il prodotto è una bevanda o una pizza e quindi la relazione tra pizze effettive e bevande effettive con pizze e bevande sarà diretta direttamente su prodotti. Bisogna però aggiungere due nuove regola aziendali per la quali solo i prodotti di tipo pizza possono essere associati a pizze effettive e solo i prodotti di tipo bevanda possono essere associati a bevande effettive, inoltre si decide di cambiare il nome dell'entità Pizze Effettive in Pizze e di Bevande Effettive in Bevande per tanto questa parte di schema risulterà essere:



Anche per la generalizzazione tra Ingredienti e Aggiunte decido di eliminare la tabella delle aggiunte e di aggiungere agli Ingredienti l'attributo costo che ha valore solo se è una possibile aggiunta. Per tanto la relazione tra Pizze e Aggiunte ora diventa una relazione tra Pizze e Ingredienti ma bisogna mettere un'altra nuova regola aziendale che gli ingredienti che non sono aggiunte non devono essere associati a nessuna Pizza.
Inoltre, per motivi di ridondanza si decide di eliminare l'entità scontrini ed aggiungere ai clienti l'attributo SpesaTotale che indica quanto hanno speso i clienti all'interno della pizzeria.
Visto che non ci sono operazioni che obbligano la visione diretta dei turni del cameriere allora si decide di eliminare la relazione "Lavora" per motivi di ridondanza poiché i turni di un cameriere sono già presenti nell'entità Tavolo Effettivo a lui associato.
Lo schema ristrutturato risulterebbe quindi essere:



Trasformazione di attributi e identificatori

Dato che sia nella relazione ricettacolo che nella relazione add vi è un attributo chiamato quantità trasformo l'attributo quantità nella relazione add in porzione.

Traduzione di entità e associazioni

- Camerieri (Nome, Cognome)
- Tavoli (N_Tavolo, NumeriPersone)
- Turni (Data, Ora_Iniziale, Ora_Finale)
- Ingredienti (NomeI, Totale, Costo)
- Prodotti (NomeP, CostoP, Tipo)
- Ricettacolo (NomeIngrediente, NomeProdotto, Quantità)

con vincoli di integrità referenziale tra l'attributo (NomeIngrediente) e la relazione Ingredienti e tra l'attributo (NomeProdotto) e la relazione Prodotti.

- TavoliEffettivi (Tavolo, Turni_data, Turni_ora, Disponibilità, CamiereNome, CamiereCognome)

con vincoli di integrità referenziale tra la coppia di attributi (Turni_Data,Turni_ora) e la relazione Turni e tra la coppia di attributi (CamiereNome,CamiereCognome) la relazione Camerieri.

- Clienti (ID, Nome, Cognome, N_Persone, TavoloEffettivo, Turno, SpesaTotale)

Con vincolo di integrità referenziale tra la coppia di attributi (TavoloEffettivo,Turno) e la relazione TavoliEffettivi

- Comande (Cliente, Data,Ora, Completata)

Con vincolo di integrità referenziale tra l'attributo (Cliente) e la relazione Clienti

- Bevande (ID_Bevande, Clienti, Data, Ora, NomeProdotto, Stato, quantitaB)

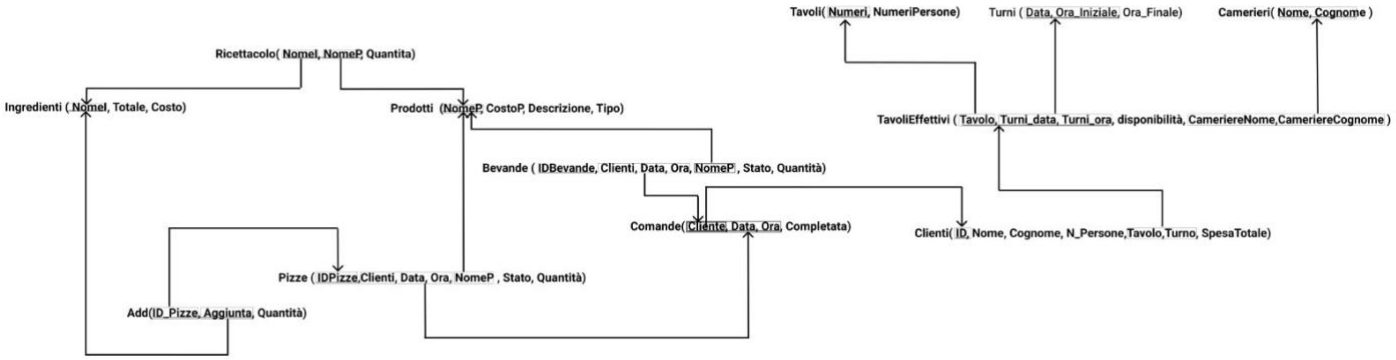
Con vincoli di integrità referenziale tra la terna di attributi (Clienti,Data,Ora) e la relazione Comande e tra l'attributo NomeProdotto e la relazione Prodotti

- Pizze (ID_Pizze, Clienti, Data, Ora, NomeProdotto, Stato, quantitaP)

Con vincoli di integrità referenziale tra la terna di attributi (Clienti,Data,Ora) e la relazione Comande e tra l'attributo NomeProdotto e la relazione Prodott

- Add (ID_Pizze, Aggiunta, porzione)

Con vincoli di integrita referenziale tra l'attributo (ID_Pizze) e la relazione Pizze e tra l'attributo (Aggiunte) e la relazione Ingredienti


Tavoli(Numeri, NumeriPersone)
Turni (Data, Ora_Iniziale, Ora_Finale)
Camerieri(Nome, Cognome)
Ricettacolo(NomeI, NomeP, Quantita)
Ingredienti (.NomeI, Totale, Costo)
Prodotti (NomeP, CostoP, Descrizione, Tipo)
TavoliEffettivi (Tavolo, Turni_data, Turni_ora, disponibilità, CameriereNome,CamerieCognome)
Bevande (IDBevande, Clienti, Data, Ora, NomeP , Stato, Quantità)
Comande(Cliente, Data, Ora, Completata)
Clienti(ID, Nome, Cognome, N_Persone,Tavolo,Turno, SpesaTotale)
Pizze (IDPizze,Clienti, Data, Ora, NomeP , Stato, Quantità)
Add(ID_Pizze, Aggiunta, Quantità)

5.Progettazione fisica

Utenti e privilegi

Ho previsto 4 tipi di utenti quali: Manager, Barista, Pizzaiolo e Cameriere che hanno username e password tramite cui si identificano Gli utenti effettivamente creati sono stati 5, dato che ho creato un utente login che come unico privilegio ha l'Execute sulla Procedure di login, dato che prima di loggare un utente deve avere la possibilità di accedere al database per poter verificare username e password.
I privilegi di tutti gli Utenti sono solo di tipo Execute su determinate Procedure e nessun utente ha la possibilità di modificare o vedere o inserire nessuna tabella se non con le Procedure messe a lui a disposizione.

- LOGIN:
  - EXECUTE login
- MANAGER:
  - EXECUTE AddCamerieri
  - EXECUTE AddCliente
  - EXECUTE addIngredienti
  - EXECUTE addProdotto
  - EXECUTE addTurno
  - EXECUTE AggiungiTavolo
  - EXECUTE AssegnareTavoloCameriere
  - EXECUTE AssegnaTurnoTavolo
  - EXECUTE Cliente_Tavolo
  - EXECUTE Creare_Turno
  - EXECUTE dbListaIngredienti
  - EXECUTE IncassiGiornalieri
  - EXECUTE IncassiMensili
  - EXECUTE IncrementaIngrediente
  - EXECUTE StampaScontrino
  - EXECUTE ListCamerieri
  - EXECUTE ListTurni
  - EXECUTE TavoliLiberi
  - EXECUTE UpdateIngredienti
- BARISTA:
  - EXECUTE BaristaOrdini
  - EXECUTE BevandePronte
- PIZZAIOLO:
  - EXECUTE OrdiniDaPrepararePizzaiolo
  - EXECUTE pizzaPreparata
- CAMERIERE:
  - EXECUTE menu
  - EXECUTE Ordine
  - EXECUTE ordiniDaPortare
  - EXECUTE TavoliAssegnatiCameriere
  - EXECUTE UpdateComandeConsegnate
  - EXECUTE VerificaCameriere
  - EXECUTE Cliente_Tavolo

Strutture di memorizzazione

Tabella camerieri		
Colonna	**Tipo di dato**	**Attributi**[2]
Nome	VARCHAR (15)	PK, NN
Cognome	VARCHAR (15)	PK, NN

Tabella tavoli		
Colonna	**Tipo di dato**	**Attributi**[3]
N_Tavolo	INT	PK, NN
N_posti	INT	NN

Tabella turni		
Colonna	**Tipo di dato**	**Attributi**[4]
Data	DATETME	PK, NN
Ora_fine	TIME	NN

Tabella tavoloeffettivo		
Colonna	**Tipo di dato**	**Attributi**[5]
Tavolo	INT	PK, NN
TurnoData	DATETIME	PK, NN
Disponibilità	tinyint	NN
CamerieреNome	VARCHAR (15)	DEFAULT=NULL
CamerieреCognome	VARCHAR (15)	DEFAULT=NULL

Tabella clienti		
Colonna	**Tipo di dato**	**Attributi**[6]
ID	INT	PK, NN, AI
Nome	VARCHAR (15)	NN
Cognome	VARCHAR (15)	NN
N_persone	INT	NN
FTavlo	INT	NN
Turno	DATETIME	NN
Spesa	FLOAT	DEFAULT=0

Tabella comande		
Colonna	**Tipo di dato**	**Attributi**[7]
Cliente	INT	PK, NN
Data	DATETIME	PK, NN
Completata	VARCHAR (10)	NN

Tabella ingredienti		
Colonna	**Tipo di dato**	**Attributi**[8]
Nome	VARCHAR (15)	PK, NN
Quantita	INT	NN DEFAULT 0
Costo	FLOAT	NN DEFAULT 0

Tabella ricettacolo		
Colonna	**Tipo di dato**	**Attributi**[9]
NomeIngrediente	Varchar(15)	PK, NN
NomeProdotto	Varchar(15)	PK, NN
Quantita	INT	NN

Tabella prodotti		
Colonna	**Tipo di dato**	**Attributi**[10]
Nome	Varchar(15)	PK, NN
Costo	Float	NN
Tipo	INT	NN

Tabella bevande		
Colonna	**Tipo di dato**	**Attributi**[11]
IDBevande	INT	PK, NN, AI
Clienti	INT	NN
DataComanda	DATETIME	NN
NomeProdotto	VARCHAR (15)	NN
Stato	VARCHAR (15)	NN DEFAULT 'in preparazione'
Quantita	INT	NN DEFAULT 1

Tabella pizze		
Colonna	**Tipo di dato**	**Attributi**[12]
IDPizze	INT	PK, NN, AI
Cliente	INT	NN
DataComanda	DATETIME	NN
NomeP	VARCHAR (15)	NN
Stato	VARCHAR (15)	NN DEFAULT 'in preparazione'
Quantita	INT	NN DEFAULT 1

Tabella add		
Colonna	**Tipo di dato**	**Attributi**[13]
IDpizze	INT	PK, NN
Aggiunta	VARCHAR (15)	PK, NN

porzione	INT	NN DEFAULT 1

Indici

Tabella add	
Indice PRIMARY	**Tipo[14]:**
Aggiunta	PR
IDpizze	PR
Indice Aggiunta_idx	**Tipo:**
Aggiunta	IDX
Indice fkpizze_idx	**Tipo:**
IDpizze	IDX

Tabella bevande	
Indice PRIMARY	**Tipo[15]:**
IDBevande	PR
Indice FKComande_idx	**Tipo:**
Clienti	IDX
DataComanda	IDX
Indice FKprodotti_idx	**Tipo:**
NomeProdotto	IDX
Indice_Stato_bevande	**Tipo**
Stato	IDX

Tabella camerieri	
Indice PRIMARY	**Tipo[16]:**
Nome	PR
Cognome	PR

Tabella clienti	
Indice PRIMARY	**Tipo[17]:**
ID	PR
Indice fk_Tavolo_idx	**Tipo:**
FTavolo	IDX
Turno	IDX

Tabella comande	
Indice PRIMARY	**Tipo[18]:**
Cliente	PR
Data	PR
Indice fk_Stato_comanda	**Tipo:**
Completata	IDX

Tabella ingredienti	
Indice PRIMARY	**Tipo[19]:**
Nome	PR

Tabella pizze	
Indice PRIMARY	**Tipo[20]:**
IDPizze	PR
Indice FKComanda_idx	**Tipo:**
Cliente	IDX
DataComanda	IDX
Indice FKProdotto_idx	**Tipo:**
NomeP	IDX
Indice_Stato_bevande	**Tipo**
Stato	IDX

Tabella prodotti	
Indice PRIMARY	**Tipo[21]:**
Nome	PR

Tabella ricettacolo	
Indice PRIMARY	**Tipo[22]:**
NomeIngrediente	PR
NomeProdotto	PR
Indice Prodotti_idx	**Tipo:**
NomeProdotto	IDX
Indice Ingredienti_idx	**Tipo:**
NomeIngrediente	IDX

Tabella tavolo	
Indice PRIMARY	**Tipo[23]:**
N_Tavolo	PR
Indice indice_N_Posti_tavolo_idx	**Tipo:**
N_posti	IDX

Tabella tavoloeffettivo	
Indice PRIMARY	**Tipo[24]:**
Tavolo	PR
TurnoData	PR
Indice Turno_idx	**Tipo:**
TurnoData	IDX
Indice Cameriere_idx	**Tipo:**
CamerieреNome	IDX
CamerieреCognome	IDX
Indice_Disponibilita_Tavolo	**Tipo**
Disponibilità	IDX

Tabella turni	
Indice PRIMARY	**Tipo[25]:**
Data	PR

Trigger

Il trigger VerificaAggiunta permette di controllare che non venga collegato a nessuna pizza un ingrediente che non fosse un'aggiunta poiché si considerano aggiunte tutti gli ingredienti che hanno costo maggiore di 0.

```
 CREATE DEFINER=`kobero`@`localhost` TRIGGER `verificaAggiunta` BEFORE INSERT ON `add` FOR EACH ROW BEGIN
declare msg varchar(128);
    declare s float;
    select Costo from ingredienti where new.Aggiunta=Nome into s;
    if s=0 then
set msg = concat('MyTriggerError: Trying to insert a negative value in trigger_test: ',new.Aggiunta);
        signal sqlstate '45000' set message_text = msg;
    end if;
END
```

Il trigger ControlloAggiunta permette di verificare che per ogni Ingrediente di tipo aggiunta che richiede la quantità richiesta sia minore della Quantità a disposizione della pizzeria se non è così allora si manda un'eccezione altrimenti si decrementa l'ingrediente corrispondente all'aggiunta

```
CREATE DEFINER=`kobero`@`localhost` TRIGGER `ControlloAggiunte` BEFORE INSERT ON `add` FOR EACH ROW BEGIN
declare msg VARCHAR(128);
declare app tinyint;
    declare q int;
    declare i int default 1;
    select Quantita into q from pizze where IDPizza=new.IDpizza;
    select count(I.nome) into app
    from `pizzeria`.`ingredienti` as I
    where new.porzione*q > I.Quantita and I.nome=new.Aggiunta and I.Costo>0;
if app>0 then
set msg = concat('MyTriggerError: Non ci sono abbastanza ingredienti per quest aggiunta: ',new.Aggiunta);
        signal sqlstate '45000' set message_text = msg;
    else
update ingredienti as i
set Quantita=Quantita-new.porzione*q
        where i.Nome=new.Aggiunta;
    end if;
END
```

Il trigger ControlloBevanda permette di verificare che ogni bevanda sia effettivamente collegata ad un prodotto di tipo bevanda a cui è associato sull'attributo tipo il valore 0 se non è cosi si impedisce l'inserimento della bevanda

```
CREATE DEFINER=`kobero`@`localhost` TRIGGER `ControlloBevanda` BEFORE INSERT ON `bevande` FOR EACH ROW BEGIN
declare msg VARCHAR(128);
declare app tinyint;
```

```
Select `prodotti`.tipo from `prodotti` where new.NomeProdotto=`prodotti`.Nome into app;
   if app= 1 then
set msg = concat('MyTriggerError ControlloBevanda: Trying to insert a negative value in trigger_test: ',app);
      signal sqlstate '45000' set message_text = msg;
   end if;
End
```

Il trigger Bevande controlla che per ogni bevanda che si aggiunge nella tabella è possibile decrementare la Quantità degli ingredienti che servono per cucinare quel prodotto in caso il sistema non abbia abbastanza ingredienti allora manda un segnale e si annulla l'inserimento altrimenti si decrementano tutti i prodotti che si usano per preparare quella bevanda

```
CREATE DEFINER=`kobero`@`localhost` TRIGGER `bevande` BEFORE INSERT ON `bevande` FOR EACH ROW BEGIN
declare msg VARCHAR(128);
declare app tinyint;
   declare nome varchar(15);
   declare i int default 1;
   select count(I.nome) into app
   from `pizzeria`.`prodotti` as P join `pizzeria`.`ricettacolo` as R on P.Nome=R.NomeProdotto join `pizzeria`.`ingredienti` as I on R.NomeIngrediente = I.Nome
   where R.Quantita*new.Quantita > I.Quantita and P.Nome=new.NomeProdotto;
if app>0 then
set msg = concat('MyTriggerError: Trying to insert a negative value in trigger_test: ',new.NomeProdotto);
      signal sqlstate '45000' set message_text = msg;
   else
update ingredienti as i
set Quantita=Quantita-new.Quantita*(select Quantita from ricettacolo as R where R.NomeIngrediente=i.Nome and Nomeprodotto=new.NomeProdotto)
      where i.Nome in (select NomeIngrediente from ricettacolo as R where R.NomeIngrediente=i.Nome and Nomeprodotto=new.NomeProdotto);
   end if;
END
```

Il trigger bevande_AFTER_UPDATE controlla che ogni volta che si effettua un update sulla tabella bevande tutte le pizze e tutte le bevande della comanda relativa alla bevanda che si è andata ad aggiornare siano nello stato di 'pronto'. In caso affermativo allora si va ad aggiornare l'attributo Completa della comanda con il valore di 'completata'. In caso negativo non si fa nulla

```
CREATE DEFINER=`kobero`@`localhost` TRIGGER `bevande_AFTER_UPDATE` AFTER UPDATE ON `bevande` FOR EACH ROW BEGIN
declare i int;
   declare j int;

select count(*) into i
   from comande as c join pizze as p on p.Cliente=c.Cliente and p.DataComanda=c.Data
where c.Cliente=new.Clienti and c.Data=new.DataComanda and p.Stato='in preparazione';

   select count(*) into j
```

```
   from comande as c join bevande as p on p.Clienti=c.Cliente and p.DataComanda=c.Data
where c.Cliente=new.Clienti and c.Data=new.DataComanda and p.Stato='in preparazione';

   if i+j=0 then
update comande set Completata='completa' where Cliente=new.Clienti and
Data=new.DataComanda;
end if;
END
```

Il trigger clienti_BEFORE_INSERT controlla che ogni nuovo cliente inserito non possa essere messo su un tavolo dove non vi siano camerieri assegnati

```
CREATE DEFINER=`kobero`@`localhost` TRIGGER `clienti_BEFORE_INSERT` BEFORE
INSERT ON `clienti` FOR EACH ROW BEGIN
declare i int;
   declare msg varchar(128);
select count(CamerieреNome) from tavoloeffettivo where Tavolo=new.FTavolo and
TurnoData=new.Turno into i;
   if i=0 then
set msg = concat('MyTriggerError: il tavolo é sprovvisto di cameriere assegnat prima il cameriere al
tavolo: ',new.FTavolo);
      signal sqlstate '45000' set message_text = msg;
end if;
END
```

Il trigger ControlloPizza permette di verificare che ogni pizza sia effettivamente collegata ad un prodotto di tipo pizza a cui è associato sull'attributo tipo il valore 1 se non è cosi si impedisce l'inserimento della pizza

```
CREATE DEFINER=`kobero`@`localhost` TRIGGER `ControlloPizza` BEFORE INSERT ON
`pizze` FOR EACH ROW BEGIN
declare msg VARCHAR(128);
declare app tinyint;
Select `prodotti`.tipo  into app from `prodotti` where new.NomeP=`prodotti`.Nome;
   if app= 0 then
set msg = concat('MyTriggerError: Trying to insert a negative value in trigger_test: ', new.NomeP);
      signal sqlstate '45000' set message_text = msg;
   end if;
END
```

Il trigger pizze permette di controllare che per ogni pizza che si va ad inserire ci siano abbastanza ingredienti nel sistema per realizzarla se non fosse così si va a bloccare l'inserimento

```
CREATE DEFINER=`kobero`@`localhost` TRIGGER `pizze` BEFORE INSERT ON `pizze` FOR
EACH ROW BEGIN
declare msg VARCHAR(128);
declare app tinyint;
   declare nome varchar(15);
   declare i int default 1;
   select count(I.nome) into app
   from `pizzeria`.`prodotti` as P join `pizzeria`.`ricettacolo` as R on P.Nome=R.NomeProdotto join
`pizzeria`.`ingredienti` as I on R.NomeIngrediente = I.Nome
```

```
   where R.Quantita*new.Quantita > I.Quantita and P.Nome=new.NomeP;
if app>0 then
set msg = concat('MyTriggerError: Trying to insert a negative value in trigger_test: ',new.NomeP);
      signal sqlstate '45000' set message_text = msg;
   else
update ingredienti as i
set Quantita=Quantita-new.Quantita*(select Quantita from ricettacolo as R where R.NomeIngrediente=i.Nome and R.Nomeprodotto=new.NomeP)
        where i.Nome in (select NomeIngrediente from ricettacolo as R where R.NomeIngrediente=i.Nome and R.Nomeprodotto=new.NomeP);
   end if;
END
```

Il trigger pizze_AFTER_UPDATE controlla che ogni volta che si effettua un update sulla tabella pizze tutte le pizze e tutte le bevande della comanda relativa alla pizza che si è andata ad aggiornare siano nello stato di ‘pronto’. In caso affermativo allora si va ad aggiornare l’attributo Completa della comanda con il valore di ‘completata’. In caso negativo non si fa nulla

```
CREATE DEFINER=`kobero`@`localhost` TRIGGER `pizze_AFTER_UPDATE` AFTER UPDATE ON `pizze` FOR EACH ROW BEGIN
declare i int;
   declare j int;

select count(*) into i
   from comande as c join pizze as p on p.Cliente=c.Cliente and p.DataComanda=c.Data
where c.Cliente=new.Cliente and c.Data=new.DataComanda and p.Stato='in preparazione';

   select count(*) into j
   from comande as c join bevande as p on p.Clienti=c.Cliente and p.DataComanda=c.Data
where c.Cliente=new.Cliente and c.Data=new.DataComanda and p.Stato='in preparazione';

   if i+j=0 then
update comande set Completata='completa' where Cliente=new.Cliente and Data=new.DataComanda;
end if;
END
```

Eventi

L'evento Cancella_comanda che viene eseguito una volta al mese elimina tutte le comande dal sistema che hanno l’attributo Data più vecchio di due mesi, serve per impedire che il volume dei dati cresca all‘ infinito. Anche i due eventi Cancella_Turni e Cancella_Cliente sono pensati per lo stesso scopo di limitare la crescita dei dati e sono pensati per eseguire il primo una volta l’anno e il secondo una volta ogni due anni in modo da eliminare i turni e i clienti più vecchi di uno e due anni.

```
create event Cancella_comanda
on schedule every 2 month
do
delete from comande where  datediff(current_date(),date(Data))>60;

create event Cancella_clienti
```

```
on schedule every 2 year
do
delete from clienti where  datediff(current_date(),date(Turno))>730;

create event Cancella_Turni
on schedule every 1 year
do
   delete from turni where  datediff(current_date(),date(Data))>365;
```

Viste

La vista permette di vedere tutte le pizze e le rispettive aggiunte e tutte le bevande, con i relativi costi e il subtotale del costo complessivo di ogni pizza e bevanda

```
CREATE
   ALGORITHM = UNDEFINED
   DEFINER = `kobero`@`localhost`
   SQL SECURITY DEFINER
VIEW `Scontrini` AS
   SELECT
      `p`.`Cliente` AS `Cliente`,
      `p`.`IDPizza` AS `IDPizza`,
      `p`.`NomeP` AS `NomeP`,
      `p`.`Quantita` AS `Quantita`,
      `pr`.`Costo` AS `prezzo`,
      `a`.`Aggiunta` AS `Aggiunta`,
      `a`.`porzione` AS `porzione`,
      `i`.`Costo` AS `Costo`,
      ((`p`.`Quantita` * (SELECT
            SUM((`a1`.`porzione` * `i1`.`Costo`))
         FROM
            ((`pizze` `p1`
            JOIN `add` `a1` ON ((`a1`.`IDpizza` = `p1`.`IDPizza`)))
            JOIN `ingredienti` `i1` ON ((`a1`.`Aggiunta` = `i1`.`Nome`)))
         WHERE
               (`p`.`IDPizza` = `p1`.`IDPizza`))) + (`p`.`Quantita` * `pr`.`Costo`)) AS
`CostoTotaleProdotto`
   FROM
      (((`pizze` `p`
      JOIN `add` `a` ON ((`a`.`IDpizza` = `p`.`IDPizza`)))
      JOIN `prodotti` `pr` ON ((`pr`.`Nome` = `p`.`NomeP`)))
      JOIN `ingredienti` `i` ON ((`a`.`Aggiunta` = `i`.`Nome`)))
   UNION SELECT
      `b`.`Clienti` AS `Clienti`,
      `b`.`IDBevande` AS `IDBevande`,
      `b`.`NomeProdotto` AS `NomeProdotto`,
      `b`.`Quantita` AS `Quantita`,
      `pr`.`Costo` AS `Costo`,
      NULL AS `NULL`,
      NULL AS `NULL`,
      NULL AS `NULL`,
```

```
    (`b`.`Quantita` * `pr`.`Costo`) AS `CostoTotatale`
  FROM
    (`bevande` `b`
    JOIN `prodotti` `pr` ON ((`pr`.`Nome` = `b`.`NomeProdotto`)))
  UNION SELECT
    `pizze`.`Cliente` AS `Cliente`,
    `pizze`.`IDPizza` AS `IDPizza`,
    `pizze`.`NomeP` AS `NomeP`,
    `pizze`.`Quantita` AS `Quantita`,
    `prodotti`.`Costo` AS `Costo`,
    NULL AS `NULL`,
    NULL AS `NULL`,
    NULL AS `NULL`,
    (`prodotti`.`Costo` * `pizze`.`Quantita`) AS `prodotti.Costo*pizze.Quantita`
  FROM
    (`pizze`
    JOIN `prodotti` ON ((`pizze`.`NomeP` = `prodotti`.`Nome`)))
  WHERE
    `pizze`.`IDPizza` IN (SELECT
        `add`.`IDpizza`
      FROM
        `add`)
      IS FALSE
  ORDER BY `Cliente` , `IDPizza`
```

Vista che mostra ogni pizza con la sua relativa aggiunta che deve essere preparata dal pizzaiolo

```
CREATE
  ALGORITHM = UNDEFINED
  DEFINER = `kobero`@`localhost`
  SQL SECURITY DEFINER
VIEW `ViewPizzaioloOrdini` AS
  SELECT
    `p`.`DataComanda` AS `DataOrdine`,
    `p`.`IDPizza` AS `IDPizza`,
    `p`.`NomeP` AS `NomeP`,
    `p`.`Stato` AS `Stato`,
    `p`.`Quantita` AS `Quantita`,
    `a`.`Aggiunta` AS `Aggiunta`,
    `a`.`porzione` AS `porzione`
  FROM
    (`pizze` `p`
    LEFT JOIN `add` `a` ON ((`a`.`IDpizza` = `p`.`IDPizza`)))
  WHERE
    (`p`.`Stato` = 'in preparazione')
  ORDER BY `p`.`DataComanda` , `p`.`IDPizza`
```

Stored Procedures e transazioni

La Procedure AddCameriere permette di inserire un nuovo cameriere passando come parametri due stringhe che rappresentano il nome e il cognome del nuovo cameriere

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `AddCamerieri`(IN N Varchar(15),IN C varchar(15))
BEGIN
INSERT INTO `pizzeria`.`camerieri`(Nome,Cognome) value (N,C);
END
```

La procedure AddCliente permette di inserire un nuovo cliente all'interno del sistema e segna che il tavolo in cui si inserisce il nuovo cliente sia occupato, i parametri che prende sono il nome del nuovo cliente, il cognome, il numero di commensali, il tavolo e il turno in cui viene per mangiare

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `AddCliente`(IN nome VARCHAR(15),IN cognome VARCHAR(15),IN persone INTEGER, IN tavolo INTEGER,IN d DATETIME)
BEGIN
set transaction isolation level read committed;
start transaction;
INSERT into clienti (Nome,Cognome,N_Persone,FTavolo,Turno) values (nome,cognome,persone,tavolo,d);
update tavoloeffettivo set Disponibilità=0 where `tavoloeffettivo`.Tavolo=tavolo and `tavoloeffettivo`.TurnoData=d;

   commit;
END
```

La procedure addIngredienti permette di inserire all'interno del sistema un nuovo Ingrediente passando come parametri il nome del nuovo ingrediente la quantità e il costo

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `addIngredienti`(IN nome VARCHAR(15),IN quantita INT,IN c INT)
BEGIN
insert into ingredienti(Nome,Quantita,Costo) values (nome,quantita,c);

END
```

La procedure addProdotto permette di inserire nel sistema un nuovo prodotto passando come parametri il nome del prodotto, il prezzo, il tipo che può essere 1 o 0 a seconda che sia una pizza o una bevanda e una stringa che rappresenta gli ingredienti e le quantità che si usano per quel determinato prodotto, la stringa deve essere composta nel seguente modo per essere accettata correttamente dalla procedure:

*(NomeIngrediente?Quantita#)@

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `addProdotto`(in nome VarChar(15),in prezzo float,in tipo int,in L varchar(2048))
begin
	declare ListaIngredienti varchar(2048);
declare inizio int default 1;
declare end1 int default 1;
	declare end2 int default 1;
	declare app varchar(15);
	declare appQ int;
	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 UNCOMMITTED;
        start transaction;
set ListaIngredienti = L;
insert into `pizzeria`.`prodotti` (Nome,Costo,Tipo) values(nome,prezzo,tipo);
                set end2=LOCATE('@',ListaIngredienti);
                while inizio<end2 do
    set end1=Locate('?',ListaIngredienti,inizio);
                        set app=substring(ListaIngredienti,inizio,end1-inizio);
            set inizio=end1+1;
                        set end1=Locate('#',ListaIngredienti,inizio);
                        set appQ= convert( substring(ListaIngredienti,inizio,end1-inizio), unsigned int );
                          insert into `pizzeria`.`ricettacolo`(NomeIngrediente,NomeProdotto,Quantita)
        values(app,nome,appQ);
set inizio=end1+1;
        end while;
         commit;
End
```

La procedure AddTurno permette di inserire un nuovo turno nel sistema data una data e un'ora con una durata fissa di 8 ore

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `AddTurno`(IN DataInizio DATETIME)
BEGIN
INSERT into turni(Data,Ora_fine) values (DataInizio,TIME(ADDTIME(DataInizio,"08:00:00")));
END
```

La procedure AggiungiTavolo permette di inserire un nuovo tavolo

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `AggiungiTavolo`(IN N_Posti INTEGER)
BEGIN
        Declare Numero INTEGER;
        set transaction isolation level SERIALIZABLE;
        start transaction;
                Select max(N_Tavolo) from tavoli into Numero;
        IF Numero is not null then
                Insert tavoli (N_Tavolo,N_posti) values (Numero+1,N_posti);
        ELSE
                 Insert tavoli (N_Tavolo,N_posti) values (1,N_posti);
        END IF;
COMMIT;
END
```

La procedure AssegnaTavoloCameriere permette di assegnare ad un tavolo in un determinato turno un cameriere prendendo come parametri il nome del cameriere, il cognome, il numero del tavolo e il turno che si sta considerando

```
CREATE DEFINER=`kobero`@`localhost` PROCEDURE `AssegnareTavoloCameriere`(in n varchar(15),in c varchar(15),in t int, in turno DATETIME)
BEGIN
        update tavoloeffettivo set CamerieреNome=n, CamerieреCognome=c where Tavolo=t and TurnoData=turno;
END
```

La procedure AssegnaTurnoTavolo permette di assegnare ad un tavolo un turno e quindi di inserire un nuovo elemento nella tabella tavoloeffettivo, la procedure prende come parametri il Numero del tavolo e una data di un turno

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `AssegnaTurnoTavolo`(IN tavolo INTEGER, IN data DATETIME )
BEGIN
        INSERT INTO tavoloeffettivo(Tavolo,TurnoData,Disponibilità) values (tavolo,data,1);
END
```

La procedure BaristaOrdini permette di vedere tutte le bevande che devono essere preparate dal barista in ordine di arrivo

```
CREATE DEFINER=`kobero`@`localhost` PROCEDURE `BaristaOrdini`()
BEGIN
select IDBevande,NomeProdotto,Quantita from bevande where Stato="in preparazione" order by DataComanda;
END
```

La procedure BevandePronte permette di dire quale bevanda è pronta per essere consegnata dal cameriere

```
CREATE DEFINER=`kobero`@`localhost` PROCEDURE `BevandePronte`(in id int)
BEGIN
update bevande set Stato="Pronto" where IDBevande=id;
END
```

La procedure Cliente_tavolo prende come parametro un tavolo e ritorna ID del cliente che sta mangiando su quel tavolo nel turno in cui si chiede

```
CREATE DEFINER=`kobero`@`localhost` PROCEDURE `Cliente_Tavolo`(in t int)
BEGIN
select ID from clienti where FTavolo=t and Spesa=0 and current_timestamp() between Turno and addtime(Turno,'08:00:00');
END
```

La procedure Crea turno permette di creare turni per tutto un mese che va indicato passando in input il numero del mese e si può anche passare l'anno se si vuole creare i turni per un mese che non è nell'anno corrente i turni che si creano con questa procedure sono turni di 8 ore dove ogni giorno viene diviso in 3 turni che vanno dalle 00:00 alle 8:00 dalle 8:00 alle 16:00 e dalle 16:00 alle 00:00

```
CREATE DEFINER=`kobero`@`localhost` PROCEDURE `Creare_Turno`(in m int,in y int)
BEGIN
   declare  var DATETIME;
   declare i int;
   declare yea int;
   if y = 0 then
set yea = year(current_timestamp());
else set yea = y;
   end if;
   SELECT addtime(adddate(makedate(yea,1), interval m-1 month ),'00:00:00') into var;
   while m=month(var)   do
      call AddTurno(var);
      set var=addtime(var,'08:00:00');
```

```
    end while;
END
```

La procedure dbListaIngredienti permette di visualizzare il nome e la quantità associata ad esso degli ingredienti presenti nel sistema

```
CREATE DEFINER=`kobero`@`localhost` PROCEDURE `dbListaIngredienti`()
BEGIN
select Nome,Quantita from ingredienti;
END
```

La procedure Incassi giornalieri prende il parametro di giorno in input e restituisce la somma di tutte le spese dei clienti che hanno mangiato nel turno del giorno di cui si vuole vedere il guadagno e restituisce il valore nella variabile s

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `IncassiGiornalieri`(IN giorno INT,OUT s INT)
BEGIN
SELECT SUM(Spesa) FROM `clienti` where giorno=DAY(`clienti`.Turno) and month(current_date())=month(`clienti`.Turno) and year(current_date())=year(`clienti`.Turno) into s;
END
```

La procedure IncassiMensili prende in input il mese e fa la somma di tutte le spese dei clienti che hanno mangiato in un turno nel mese selezionato e restituisce il valore nella variabile s

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `IncassiMensili`(IN m int,OUT s DOUBLE)
BEGIN
    SELECT SUM(`clienti`.Spesa) into s FROM `clienti` where m=MONTH(`clienti`.Turno) and year(current_date())=year(`clienti`.Turno);
END
```

La procedure incrementaIngredienti permette di incrementare il quantitativo di un dato ingrediente passando come parametri il nome dell'ingrediente e la quantità di cui si incrementa

```
CREATE DEFINER=`kobero`@`localhost` PROCEDURE `IncrementaIngrediente`(in n varchar(15),in incr INT)
BEGIN
update ingredienti set Quantita=Quantita+incr where Nome=n;
END
```

La procedure ListaAggiunte permette di vedere tutte le possibili aggiunte di una pizza e il loro costo

```
CREATE DEFINER=`kobero`@`localhost` PROCEDURE `ListaAggiunte`()
BEGIN
select Nome,Costo from ingredienti where Costo>0 and Quantita>0;
END
```

La procedure ListCameriere restituisce il nome e il cognome di tutti i camerieri memorizzati nel sistema

```
CREATE DEFINER=`kobero`@`localhost` PROCEDURE `ListCamerieri`()
BEGIN
select * from camerieri;
END
```

La procedure ListTurni restituisce tutti i turni presenti del sistema futuri (fino ad un massimo di due settimane di distanza) e quello corrente

```
CREATE DEFINER=`kobero`@`localhost` PROCEDURE `ListTurni`()
BEGIN
Select turni.Data,adddate(turni.Data,interval 8 hour) from turni where turni.Data between ADDDATE(current_time(), INTERVAL -8 hour) and ADDDATE(current_time(), INTERVAL 1 week);
END
```

La procedure login prende username e password e restituisce il ruolo dell'utente

```
CREATE DEFINER=`kobero`@`localhost` PROCEDURE `login`(in utente varchar(45), in passw varchar(45),out r varchar(45))
BEGIN
select Ruolo from Utenti where Username=utente and Pass=passw into r;
END
```

La procedure menu restituisce il menu della pizzeria con tutti i prodotti e i relativi costi

```
CREATE DEFINER=`kobero`@`localhost` PROCEDURE `menu`()
BEGIN
select Nome,Costo,Tipo from prodotti;
END
```

La procedure Ordine permette di aggiungere una nuova comanda e inserire nuove bevande e nuove Pizze con le loro aggiunte all'interno del sistema, i parametri che prende sono l'id del cliente e due stringhe che sono una la lista delle pizze con le relative aggiunte dei clienti e l'altra la lista delle bevande con le relative quantità le due stringhe devono essere formattate nel seguente modo:
Per la stringa relativa a ListaPizze deve essere:

NomePizza?Quantita#nomeAggiunta?Quantita#*@

Per la stringa relativa a ListaBevade deve essere:
NomeBevanda?Quantita#@

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `Ordine`(in cliente int,in ListaPizze varchar(2048),in ListaBevande varchar(2048))
BEGIN
declare Tempo DATETIME default current_timestamp();
declare inizio int default 1;
declare fine1 int default 1;
declare fine2 int default 1;
declare fine3 int default 1;
declare idP int;
declare nome varchar(15);
declare nomeAggiunta varchar(15);
declare quantita varchar(15);
declare quantitaAggiunta varchar(15);
declare msg varchar(128);
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;
if (select Spesa from clienti where ID=cliente) >0 then
set msg = concat('MyTriggerError: Il cliente selezionato non può effettuare altri ordini: ',cliente);
signal sqlstate '45000' set message_text = msg;
      end if;
insert `pizzeria`.`comande` values (cliente,Tempo,'no');
      set fine2=locate('@',ListaBevande);
      while inizio<fine2 do
set fine1=Locate('?',ListaBevande,inizio);
set nome=substring(ListaBevande,inizio,fine1-inizio);
set inizio=fine1+1;
set fine1=Locate('#',ListaBevande,inizio);
set quantita= convert( substring(ListaBevande,inizio,fine1-inizio), unsigned int );
set inizio=fine1+1;
            insert into `pizzeria`.`bevande`(Clienti,DataComanda,NomeProdotto,Stato,Quantita)
values(cliente,Tempo,nome,'in preparazione',quantita);
      end while;
      set inizio=1;
      set fine2=locate('@',ListaPizze,inizio);
      while inizio<fine2 do
set fine1=Locate('?',ListaPizze,inizio);
set nome=substring(ListaPizze,inizio,fine1-inizio);
set inizio=fine1+1;
         set fine1=Locate('#',ListaPizze,inizio);
set quantita= convert(substring(ListaPizze,inizio,fine1-inizio), unsigned int );
set inizio=fine1+1;
               insert into `pizzeria`.`pizze` (Cliente,DataComanda,NomeP,Stato,Quantita)
values(cliente,Tempo,nome,'in preparazione',quantita);
         set fine3=locate('*',ListaPizze,inizio);
         set idP=last_insert_id();
         while inizio<fine3 do
set fine1=Locate('?',ListaPizze,inizio);
set nomeAggiunta=substring(ListaPizze,inizio,fine1-inizio);
set inizio=fine1+1;
set fine1=Locate('#',ListaPizze,inizio);
set quantitaAggiunta= convert( substring(ListaPizze,inizio,fine1-inizio), unsigned int );
set inizio=fine1+1;
insert                    into                    `pizzeria`.`add`(IDpizza,Aggiunta,porzione)
values(IDp,nomeAggiunta,quantitaAggiunta);
end while;
         set inizio=fine3+1;
      end while;
commit;
END
```

La procedure OrdineDaPortare passato come parametri il nome del cameriere e il cognome restituisce il numero del tavolo, l'ID del cliente, la Data e l'ora della Comanda relativa a tutte le comande che sono da consegnare

```
CREATE DEFINER=`kobero`@`localhost` PROCEDURE `ordiniDaPortare`(in n varchar(15),in
cognome varchar(15))
BEGIN
```

```
select Tavolo,Cliente,Data
   from comande as c join clienti as p on c.Cliente=p.ID
join tavoloeffettivo as t on t.Tavolo=p.FTavolo and t.TurnoData=p.Turno
where t.CamerieреNome=n and
   t.CamerieреCognome=cognome and
   c.Completata="completa"
   order by Data,Cliente;
END
```

La procedure pizzaPreparata permette di dire quale pizza è pronta per essere consegnata dal cameriere

```
CREATE DEFINER=`kobero`@`localhost` PROCEDURE `pizzaPreparata`(in id int)
BEGIN
update pizze set Stato="pronto" where IDPizza=id;
END
```

La procedure OrdineDaPreparare restituisce tutte le pizze con le relative aggiunte che il pizzaiolo deve preparare in ordine di arrivo

```
CREATE DEFINER=`kobero`@`localhost` PROCEDURE `OrdiniDaPrepararePizzaiolo`()
BEGIN
select IDPizza,NomeP,Quantita,Aggiunta,porzione from ViewPizzaioloOrdini;
END
```

La procedure StampaScontrino prende in input l'id di un cliente e restituisce la somma da esso spesa e tutte le pizze e le bevande con i relativi prezzi che lui ha ordinato

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `StampaScontrino`(IN c INT,out CostoTotale INT)
BEGIN
   declare t int;
   declare d datetime;
   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 serializable;
   start transaction;
select IDPizza,NomeP,Quantita,prezzo,Aggiunta,porzione,Costo,CostoTotaleProdotto from Scontrini where Cliente=c;
select sum(P.CostoTotaleProdotto) as costo
from (select distinct(IDPizza),CostoTotaleProdotto from Scontrini where Cliente=c) as P into CostoTotale;
update clienti set Spesa=CostoTotale where ID=c;
select FTavolo from clienti where ID=c into t;
select Turno from clienti where ID=c into d;
update tavoloeffettivo  set Disponibilità=1 where Tavolo=t and TurnoData=d;
   commit;
END
```

La procedure TavoliAssegnatiCameriere prende come parametri il nome e il cognome del cameriere e restituisce al momento i tavoli a lui assegnati nel turno corrente

```
CREATE DEFINER=`kobero`@`localhost` PROCEDURE `TavoliAssegnatiCameriere`(in n varchar(15),in c varchar(15))
BEGIN
declare var DATETIME;

select Tavolo
   from tavoloeffettivo
   where
CamiereNome=n and
CamiereCognome=c and
current_timestamp() between TurnoData and addtime(TurnoData,"08:00:00");
END
```

La procedure TavoliLiberi restituisce tutti i tavoli disponibili per il turno corrente della dimensione uguale o maggiore del valore passato tra i parametri

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `TavoliLiberi`(IN posti INTEGER)
BEGIN
SELECT `tavoloeffettivo`.Tavolo, `tavoloeffettivo`.TurnoData FROM `tavoli` JOIN `tavoloeffettivo` ON `tavoli`.N_Tavolo=`tavoloeffettivo`.Tavolo where `tavoli`.N_posti>=posti and timediff(current_timestamp(),`tavoloeffettivo`.TurnoData)<'08:00:00' and timediff(current_timestamp(),`tavoloeffettivo`.TurnoData) > '00:00:00' and Disponibilità=1;
END
```

La procedure UpdateComandeConsegnate è usata per indicare se una comanda è stata consegnata o meno impostando l'attributo Completata al valore "chiusa"

```
CREATE DEFINER=`kobero`@`localhost` PROCEDURE `UpdateComandeConsegnate`(in id INT,in d DATETIME)
BEGIN
update comande set Completata="chiusa" where Cliente=id and Data=d;
END
```

La procedure UpdateIngredienti prende come parametri il nome dell'ingrediente e aggiorna la quantità del ingrediente scelto del valore passato come secondo parametro

```
CREATE DEFINER=`kobero`@`localhost` PROCEDURE `UpdateIngrdienti`(in n varchar(15),in incr INT)
BEGIN
update ingredienti set Quantita=Quantita+incr where Nome=n;
END
```

La procedure verificaCameriere controlla se dato il nome e il cognome esiste un cameriere con tale nome o cognome

```
CREATE DEFINER=`kobero`@`localhost` PROCEDURE `VerificaCameriere`(in n varchar(15),in c varchar(15))
BEGIN
declare i int;
        declare msg varchar(128);
select count(Nome) from camerieri where Nome=n and Cognome=c into i;
        if i = 0 then
set msg = "il Cameriere non esiste";
                signal sqlstate '45000' set message_text = msg;
        end if;
```

END

Appendice: Implementazione

Codice SQL per instanziare il database

```
CREATE DATABASE  IF NOT EXISTS `pizzeria` /*!40100 DEFAULT CHARACTER SET utf8
*/ /*!80016 DEFAULT ENCRYPTION='N' */;
USE `pizzeria`;
-- MySQL dump 10.13  Distrib 8.0.28, for Linux (x86_64)
--
-- Host: 127.0.0.1    Database: pizzeria
-- ------------------------------------------------------
-- Server version        8.0.28-0ubuntu0.21.10.3

/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
/*!40101 SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
/*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;
/*!50503 SET NAMES utf8 */;
/*!40103 SET @OLD_TIME_ZONE=@@TIME_ZONE */;
/*!40103 SET TIME_ZONE='+00:00' */;
/*!40014 SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0 */;
/*!40014          SET          @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS,
FOREIGN_KEY_CHECKS=0 */;
/*!40101                              SET                              @OLD_SQL_MODE=@@SQL_MODE,
SQL_MODE='NO_AUTO_VALUE_ON_ZERO' */;
/*!40111 SET @OLD_SQL_NOTES=@@SQL_NOTES, SQL_NOTES=0 */;

--
-- Table structure for table `Utenti`
--

DROP TABLE IF EXISTS `Utenti`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!50503 SET character_set_client = utf8mb4 */;
CREATE TABLE `Utenti` (
  `Username` varchar(45) NOT NULL,
  `Pass` varchar(45) NOT NULL,
  `Ruolo` int NOT NULL,
  PRIMARY KEY (`Username`,`Pass`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb3;
/*!40101 SET character_set_client = @saved_cs_client */;

--
-- Dumping data for table `Utenti`
--

LOCK TABLES `Utenti` WRITE;
/*!40000 ALTER TABLE `Utenti` DISABLE KEYS */;
INSERT                              INTO                              `Utenti`                              VALUES
('barista','1234',1),('cameriere','1234',4),('manager','1234',2),('pizzaiolo','1234',3);
/*!40000 ALTER TABLE `Utenti` ENABLE KEYS */;
UNLOCK TABLES;
```

```
--
-- Table structure for table `add`
--

DROP TABLE IF EXISTS `add`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!50503 SET character_set_client = utf8mb4 */;
CREATE TABLE `add` (
  `Aggiunta` varchar(15) NOT NULL,
  `porzione` int NOT NULL DEFAULT '1',
  `IDpizza` int NOT NULL,
  PRIMARY KEY (`Aggiunta`,`IDpizza`),
  KEY `Aggiunta_idx` (`Aggiunta`),
  KEY `fkpizza_idx` (`IDpizza`),
  CONSTRAINT `fkAggiunta` FOREIGN KEY (`Aggiunta`) REFERENCES `ingredienti` (`Nome`) ON DELETE CASCADE ON UPDATE CASCADE,
  CONSTRAINT `fkpizza` FOREIGN KEY (`IDpizza`) REFERENCES `pizze` (`IDPizza`) ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb3;
/*!40101 SET character_set_client = @saved_cs_client */;

--
-- Dumping data for table `add`
--

LOCK TABLES `add` WRITE;
/*!40000 ALTER TABLE `add` DISABLE KEYS */;
INSERT INTO `add` VALUES ('funghi',1,44),('funghi',1,46),('funghi',1,48),('funghi',1,52),('funghi',3,54),('funghi',1,56),('funghi',1,58),('funghi',2,61),('melanzana',1,56),('melanzana',1,63),('mozzarella',1,60),('salsiccia',1,53);
/*!40000 ALTER TABLE `add` ENABLE KEYS */;
UNLOCK TABLES;

--
-- Table structure for table `bevande`
--

DROP TABLE IF EXISTS `bevande`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!50503 SET character_set_client = utf8mb4 */;
CREATE TABLE `bevande` (
  `Clienti` int NOT NULL,
  `DataComanda` datetime NOT NULL,
  `NomeProdotto` varchar(15) NOT NULL,
  `Stato` varchar(45) NOT NULL DEFAULT 'in preparazione',
  `Quantita` int NOT NULL DEFAULT '1',
  `IDBevande` int NOT NULL AUTO_INCREMENT,
  PRIMARY KEY (`IDBevande`),
  KEY `FKComande_idx` (`Clienti`,`DataComanda`),
  KEY `FKProdotti_idx` (`NomeProdotto`),
```

```
  CONSTRAINT `FKComande` FOREIGN KEY (`Clienti`, `DataComanda`) REFERENCES `comande` (`Cliente`, `Data`) ON DELETE CASCADE ON UPDATE CASCADE,
  CONSTRAINT `FKProdotti` FOREIGN KEY (`NomeProdotto`) REFERENCES `prodotti` (`Nome`) ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB AUTO_INCREMENT=9 DEFAULT CHARSET=utf8mb3;
/*!40101 SET character_set_client = @saved_cs_client */;

--
-- Dumping data for table `bevande`
--

LOCK TABLES `bevande` WRITE;
/*!40000 ALTER TABLE `bevande` DISABLE KEYS */;
INSERT INTO `bevande` VALUES (25,'2022-02-03 16:07:59','pepsicola','Pronto',1,4),(25,'2022-02-03 16:07:59','peroni','Pronto',1,5),(26,'2022-02-03 20:21:29','pepsicola','Pronto',1,6),(25,'2022-02-03 23:48:24','pepsicola','Pronto',1,7),(27,'2022-02-06 14:38:14','pepsicola','Pronto',1,8);
/*!40000 ALTER TABLE `bevande` ENABLE KEYS */;
UNLOCK TABLES;

--
-- Table structure for table `camerieri`
--

DROP TABLE IF EXISTS `camerieri`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!50503 SET character_set_client = utf8mb4 */;
CREATE TABLE `camerieri` (
  `Nome` varchar(15) NOT NULL,
  `Cognome` varchar(15) NOT NULL,
  PRIMARY KEY (`Nome`,`Cognome`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb3;
/*!40101 SET character_set_client = @saved_cs_client */;

--
-- Dumping data for table `camerieri`
--

LOCK TABLES `camerieri` WRITE;
/*!40000 ALTER TABLE `camerieri` DISABLE KEYS */;
INSERT INTO `camerieri` VALUES ('cameriere','cameriere'),('giuseppe','federico'),('lorenzo','d'),('m','f'),('matteo','federico'),('sara','da canal'),('steven','cameriere');
/*!40000 ALTER TABLE `camerieri` ENABLE KEYS */;
UNLOCK TABLES;

--
-- Table structure for table `clienti`
--

DROP TABLE IF EXISTS `clienti`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
```

```
/*!50503 SET character_set_client = utf8mb4 */;
CREATE TABLE `clienti` (
  `ID` int NOT NULL AUTO_INCREMENT,
  `Nome` varchar(15) NOT NULL,
  `Cognome` varchar(15) NOT NULL,
  `N_persone` int NOT NULL,
  `FTavolo` int DEFAULT NULL,
  `Turno` datetime DEFAULT NULL,
  `Spesa` int DEFAULT '0',
  PRIMARY KEY (`ID`),
  KEY `fk_Tavolo_idx` (`FTavolo`,`Turno`),
    CONSTRAINT `fk_Tavolo` FOREIGN KEY (`FTavolo`, `Turno`) REFERENCES `tavoloeffettivo` (`Tavolo`, `TurnoData`) ON DELETE SET NULL ON UPDATE SET NULL
) ENGINE=InnoDB AUTO_INCREMENT=30 DEFAULT CHARSET=utf8mb3;
/*!40101 SET character_set_client = @saved_cs_client */;

--
-- Dumping data for table `clienti`
--

LOCK TABLES `clienti` WRITE;
/*!40000 ALTER TABLE `clienti` DISABLE KEYS */;
INSERT INTO `clienti` VALUES (17,'matteo','pazzizzo',3,NULL,NULL,0),(18,'luigi','talamo',3,NULL,NULL,0),(19,'nando','ferraro',5,1,'2022-02-01 08:00:00',0),(20,'peppe','peppe',4,5,'2022-02-02 08:00:00',0),(21,'matte','federico',6,1,'2022-02-02 16:00:00',50),(22,'giggi','testo',5,3,'2022-02-03 00:00:00',0),(23,'d','f',3,1,'2022-02-03 00:00:00',0),(24,'marzai','ara',4,1,'2022-02-03 08:00:00',0),(25,'matteo','federico',3,1,'2022-02-03 16:00:00',0),(26,'lorenzo','federico',3,2,'2022-02-03 16:00:00',0),(27,'f','s',3,1,'2022-02-06 08:00:00',8),(28,'f','d',3,3,'2022-02-06 08:00:00',0),(29,'f','d',3,1,'2022-02-06 08:00:00',0);
/*!40000 ALTER TABLE `clienti` ENABLE KEYS */;
UNLOCK TABLES;

--
-- Table structure for table `comande`
--

DROP TABLE IF EXISTS `comande`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!50503 SET character_set_client = utf8mb4 */;
CREATE TABLE `comande` (
  `Cliente` int NOT NULL,
  `Data` datetime NOT NULL,
  `Completata` varchar(10) NOT NULL,
  PRIMARY KEY (`Cliente`,`Data`),
  CONSTRAINT `Clienti` FOREIGN KEY (`Cliente`) REFERENCES `clienti` (`ID`) ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb3;
/*!40101 SET character_set_client = @saved_cs_client */;

--
```

```
-- Dumping data for table `comande`
--

LOCK TABLES `comande` WRITE;
/*!40000 ALTER TABLE `comande` DISABLE KEYS */;
INSERT INTO `comande` VALUES (21,'2022-02-02 22:57:04','chiusa'),(21,'2022-02-02 22:58:42','chiusa'),(21,'2022-02-02 22:58:52','chiusa'),(21,'2022-02-02 22:59:41','chiusa'),(23,'2022-02-03 00:05:45','chiusa'),(24,'2022-02-03 12:21:48','chiusa'),(24,'2022-02-03 12:50:12','chiusa'),(25,'2022-02-03 16:07:59','chiusa'),(25,'2022-02-03 23:48:24','chiusa'),(26,'2022-02-03 20:21:29','chiusa'),(27,'2022-02-06 14:38:14','chiusa');
/*!40000 ALTER TABLE `comande` ENABLE KEYS */;
UNLOCK TABLES;

--
-- Table structure for table `ingredienti`
--

DROP TABLE IF EXISTS `ingredienti`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!50503 SET character_set_client = utf8mb4 */;
CREATE TABLE `ingredienti` (
  `Nome` varchar(15) NOT NULL,
  `Quantita` int NOT NULL DEFAULT '0',
  `Costo` float DEFAULT '0',
  PRIMARY KEY (`Nome`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb3;
/*!40101 SET character_set_client = @saved_cs_client */;

--
-- Dumping data for table `ingredienti`
--

LOCK TABLES `ingredienti` WRITE;
/*!40000 ALTER TABLE `ingredienti` DISABLE KEYS */;
INSERT INTO `ingredienti` VALUES ('acciughe',900,1),('acqua',262,0),('alici',899,1),('basilico',59,0),('carne',52,0),('farina',172,0),('funghi',50,1),('melanzana',127,1),('moretti',900,0),('mozzarella',894,1),('olio',899,0),('olive',900,0),('peperoni',900,1),('pepsicola',896,0),('peroni',899,0),('salame',900,0),('salsiccia',896,2),('sugo',178,0),('zucchine',90,1);
/*!40000 ALTER TABLE `ingredienti` ENABLE KEYS */;
UNLOCK TABLES;

--
-- Table structure for table `pizze`
--

DROP TABLE IF EXISTS `pizze`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!50503 SET character_set_client = utf8mb4 */;
CREATE TABLE `pizze` (
  `Cliente` int NOT NULL,
```

```
  `DataComanda` datetime NOT NULL,
  `NomeP` varchar(15) NOT NULL,
  `Stato` varchar(45) NOT NULL DEFAULT 'in preparazione',
  `Quantita` int NOT NULL DEFAULT '1',
  `IDPizza` int NOT NULL AUTO_INCREMENT,
  PRIMARY KEY (`IDPizza`),
  KEY `FKProdotto_idx` (`NomeP`),
  KEY `FkComanda_idx` (`Cliente`,`DataComanda`),
   CONSTRAINT `FkComanda` FOREIGN KEY (`Cliente`, `DataComanda`) REFERENCES `comande` (`Cliente`, `Data`) ON DELETE CASCADE ON UPDATE CASCADE,
  CONSTRAINT `FKProdotto` FOREIGN KEY (`NomeP`) REFERENCES `prodotti` (`Nome`) ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB AUTO_INCREMENT=64 DEFAULT CHARSET=utf8mb3;
/*!40101 SET character_set_client = @saved_cs_client */;

--
-- Dumping data for table `pizze`
--

LOCK TABLES `pizze` WRITE;
/*!40000 ALTER TABLE `pizze` DISABLE KEYS */;
INSERT INTO `pizze` VALUES (21,'2022-02-02 22:57:04','margherita','pronto',1,44),(21,'2022-02-02 22:57:04','boscaiola','pronto',1,45),(21,'2022-02-02 22:58:42','margherita','pronto',1,46),(21,'2022-02-02 22:58:42','boscaiola','pronto',1,47),(21,'2022-02-02 22:58:52','margherita','pronto',1,48),(21,'2022-02-02 22:58:52','boscaiola','pronto',1,49),(21,'2022-02-02 22:59:41','margherita','pronto',1,52),(21,'2022-02-02 22:59:41','boscaiola','pronto',1,53),(23,'2022-02-03 00:05:45','boscaiola','pronto',1,54),(24,'2022-02-03 12:21:48','margherita','pronto',2,56),(24,'2022-02-03 12:50:12','margherita','pronto',2,57),(24,'2022-02-03 12:50:12','boscaiola','pronto',2,58),(25,'2022-02-03 16:07:59','boscaiola','pronto',1,60),(26,'2022-02-03 20:21:29','margherita','pronto',1,61),(25,'2022-02-03 23:48:24','boscaiola','pronto',1,62),(27,'2022-02-06 14:38:14','marinara','pronto',1,63);
/*!40000 ALTER TABLE `pizze` ENABLE KEYS */;
UNLOCK TABLES;

--
-- Table structure for table `prodotti`
--

DROP TABLE IF EXISTS `prodotti`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!50503 SET character_set_client = utf8mb4 */;
CREATE TABLE `prodotti` (
  `Nome` varchar(15) NOT NULL,
  `Costo` float NOT NULL,
  `Tipo` int NOT NULL,
  PRIMARY KEY (`Nome`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb3;
/*!40101 SET character_set_client = @saved_cs_client */;

--
```

```
-- Dumping data for table `prodotti`
--

LOCK TABLES `prodotti` WRITE;
/*!40000 ALTER TABLE `prodotti` DISABLE KEYS */;
INSERT INTO `prodotti` VALUES
('boscaiola',6,1),('margherita',5,1),('marinara',5,1),('pepsicola',2,0),('peroni',3,0);
/*!40000 ALTER TABLE `prodotti` ENABLE KEYS */;
UNLOCK TABLES;

--
-- Table structure for table `ricettacolo`
--

DROP TABLE IF EXISTS `ricettacolo`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!50503 SET character_set_client = utf8mb4 */;
CREATE TABLE `ricettacolo` (
  `NomeIngrediente` varchar(15) NOT NULL,
  `NomeProdotto` varchar(15) NOT NULL,
  `Quantita` int NOT NULL,
  PRIMARY KEY (`NomeIngrediente`,`NomeProdotto`),
  KEY `Prodotti_idx` (`NomeProdotto`),
  CONSTRAINT `Ingrediente` FOREIGN KEY (`NomeIngrediente`) REFERENCES `ingredienti`
(`Nome`) ON DELETE CASCADE ON UPDATE CASCADE,
  CONSTRAINT `Prodotti` FOREIGN KEY (`NomeProdotto`) REFERENCES `prodotti` (`Nome`)
ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb3;
/*!40101 SET character_set_client = @saved_cs_client */;

--
-- Dumping data for table `ricettacolo`
--

LOCK TABLES `ricettacolo` WRITE;
/*!40000 ALTER TABLE `ricettacolo` DISABLE KEYS */;
INSERT INTO `ricettacolo` VALUES
('acqua','boscaiola',1),('acqua','margherita',1),('alici','marinara',1),('basilico','margherita',1),('farina','b
oscaiola',1),('farina','margherita',5),('farina','marinara',2),('funghi','boscaiola',4),('mozzarella','boscai
ola',2),('mozzarella','margherita',1),('olio','marinara',1),('pepsicola','pepsicola',1),('peroni','peroni',1),(
'salsiccia','boscaiola',2),('sugo','margherita',1),('sugo','marinara',1);
/*!40000 ALTER TABLE `ricettacolo` ENABLE KEYS */;
UNLOCK TABLES;

--
-- Table structure for table `tavoli`
--

DROP TABLE IF EXISTS `tavoli`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!50503 SET character_set_client = utf8mb4 */;
```

```
CREATE TABLE `tavoli` (
  `N_Tavolo` int NOT NULL,
  `N_posti` int NOT NULL,
  PRIMARY KEY (`N_Tavolo`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb3;
/*!40101 SET character_set_client = @saved_cs_client */;

--
-- Dumping data for table `tavoli`
--

LOCK TABLES `tavoli` WRITE;
/*!40000 ALTER TABLE `tavoli` DISABLE KEYS */;
INSERT INTO `tavoli` VALUES (1,9),(2,10),(3,5),(4,1),(5,15),(6,32),(7,1),(8,2),(9,2),(10,3),(11,6),(12,5),(13,12);
/*!40000 ALTER TABLE `tavoli` ENABLE KEYS */;
UNLOCK TABLES;

--
-- Table structure for table `tavoloeffettivo`
--

DROP TABLE IF EXISTS `tavoloeffettivo`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!50503 SET character_set_client = utf8mb4 */;
CREATE TABLE `tavoloeffettivo` (
  `Tavolo` int NOT NULL,
  `Disponibilità` tinyint NOT NULL,
  `TurnoData` datetime NOT NULL,
  `CameriereNome` varchar(15) DEFAULT NULL,
  `CameriereCognome` varchar(15) DEFAULT NULL,
  PRIMARY KEY (`Tavolo`,`TurnoData`),
  KEY `Turno_idx` (`TurnoData`),
  KEY `Cameriere_idx` (`CamerieреNome`,`CamerieреCognome`),
    CONSTRAINT `Cameriere` FOREIGN KEY (`CamerieреNome`, `CamerieреCognome`) REFERENCES `camerieri` (`Nome`, `Cognome`) ON DELETE CASCADE ON UPDATE CASCADE,
  CONSTRAINT `FKTavolo` FOREIGN KEY (`Tavolo`) REFERENCES `tavoli` (`N_Tavolo`) ON DELETE CASCADE ON UPDATE CASCADE,
  CONSTRAINT `FKTurno` FOREIGN KEY (`TurnoData`) REFERENCES `turni` (`Data`) ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb3;
/*!40101 SET character_set_client = @saved_cs_client */;

--
-- Dumping data for table `tavoloeffettivo`
--

LOCK TABLES `tavoloeffettivo` WRITE;
/*!40000 ALTER TABLE `tavoloeffettivo` DISABLE KEYS */;
```

```
INSERT INTO `tavoloeffettivo` VALUES (1,0,'2022-02-01 08:00:00','sara ','da canal'),(1,0,'2022-02-02 16:00:00','sara','da canal'),(1,0,'2022-02-03 00:00:00','sara','da canal'),(1,0,'2022-02-03 08:00:00','sara','da canal'),(1,0,'2022-02-03 16:00:00','sara','da canal'),(1,0,'2022-02-06 08:00:00','cameriere','cameriere'),(1,1,'2022-02-06 16:00:00',NULL,NULL),(2,1,'2022-02-02 16:00:00',NULL,NULL),(2,1,'2022-02-03 08:00:00',NULL,NULL),(2,0,'2022-02-03 16:00:00','sara','da canal'),(2,1,'2022-02-06 08:00:00',NULL,NULL),(3,1,'2022-02-01 08:00:00','sara ','da canal'),(3,1,'2022-02-02 08:00:00',NULL,NULL),(3,0,'2022-02-03 00:00:00','sara','da canal'),(3,1,'2022-02-03 16:00:00','matteo','federico'),(3,0,'2022-02-06 08:00:00','lorenzo','d'),(4,1,'2022-02-04 16:00:00',NULL,NULL),(5,1,'2022-02-01 08:00:00',NULL,NULL),(5,0,'2022-02-02 08:00:00','sara','da canal'),(5,1,'2022-02-02 16:00:00',NULL,NULL),(6,1,'2022-02-01 08:00:00','sara','da canal'),(6,1,'2022-02-02 08:00:00',NULL,NULL);
/*!40000 ALTER TABLE `tavoloeffettivo` ENABLE KEYS */;
UNLOCK TABLES;

--
-- Table structure for table `turni`
--

DROP TABLE IF EXISTS `turni`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!50503 SET character_set_client = utf8mb4 */;
CREATE TABLE `turni` (
  `Data` datetime NOT NULL,
  `Ora_fine` time NOT NULL,
  PRIMARY KEY (`Data`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb3;
/*!40101 SET character_set_client = @saved_cs_client */;

--
-- Dumping data for table `turni`
--

LOCK TABLES `turni` WRITE;
/*!40000 ALTER TABLE `turni` DISABLE KEYS */;
INSERT INTO `turni` VALUES ('2022-02-01 00:00:00','08:00:00'),('2022-02-01 08:00:00','16:00:00'),('2022-02-01 16:00:00','00:00:00'),('2022-02-02 00:00:00','08:00:00'),('2022-02-02 08:00:00','16:00:00'),('2022-02-02 16:00:00','00:00:00'),('2022-02-03 00:00:00','08:00:00'),('2022-02-03 08:00:00','16:00:00'),('2022-02-03 16:00:00','00:00:00'),('2022-02-04 00:00:00','08:00:00'),('2022-02-04 08:00:00','16:00:00'),('2022-02-04 16:00:00','00:00:00'),('2022-02-05 00:00:00','08:00:00'),('2022-02-05 08:00:00','16:00:00'),('2022-02-05 16:00:00','00:00:00'),('2022-02-06 00:00:00','08:00:00'),('2022-02-06 08:00:00','16:00:00'),('2022-02-06 16:00:00','00:00:00'),('2022-02-07 00:00:00','08:00:00'),('2022-02-07 08:00:00','16:00:00'),('2022-02-07 16:00:00','00:00:00'),('2022-02-08 00:00:00','08:00:00'),('2022-02-08 08:00:00','16:00:00'),('2022-02-08 16:00:00','00:00:00'),('2022-02-09 00:00:00','08:00:00'),('2022-02-09 08:00:00','16:00:00'),('2022-02-09 16:00:00','00:00:00'),('2022-02-10 00:00:00','08:00:00'),('2022-02-10 08:00:00','16:00:00'),('2022-02-10 16:00:00','00:00:00'),('2022-02-11 00:00:00','08:00:00'),('2022-02-11 08:00:00','16:00:00'),('2022-02-11 16:00:00','00:00:00'),('2022-02-12 00:00:00','08:00:00'),('2022-02-12 08:00:00','16:00:00'),('2022-02-12 16:00:00','00:00:00'),('2022-02-13 00:00:00','08:00:00'),('2022-02-13 08:00:00','16:00:00'),('2022-02-13 16:00:00','00:00:00'),('2022-
```

02-14 00:00:00','08:00:00'),('2022-02-14 08:00:00','16:00:00'),('2022-02-14 16:00:00','00:00:00'),('2022-02-15 00:00:00','08:00:00'),('2022-02-15 08:00:00','16:00:00'),('2022-02-15 16:00:00','00:00:00'),('2022-02-16 00:00:00','08:00:00'),('2022-02-16 08:00:00','16:00:00'),('2022-02-16 16:00:00','00:00:00'),('2022-02-17 00:00:00','08:00:00'),('2022-02-17 08:00:00','16:00:00'),('2022-02-17 16:00:00','00:00:00'),('2022-02-18 00:00:00','08:00:00'),('2022-02-18 08:00:00','16:00:00'),('2022-02-18 16:00:00','00:00:00'),('2022-02-19 00:00:00','08:00:00'),('2022-02-19 08:00:00','16:00:00'),('2022-02-19 16:00:00','00:00:00'),('2022-02-20 00:00:00','08:00:00'),('2022-02-20 08:00:00','16:00:00'),('2022-02-20 16:00:00','00:00:00'),('2022-02-21 00:00:00','08:00:00'),('2022-02-21 08:00:00','16:00:00'),('2022-02-21 16:00:00','00:00:00'),('2022-02-22 00:00:00','08:00:00'),('2022-02-22 08:00:00','16:00:00'),('2022-02-22 16:00:00','00:00:00'),('2022-02-23 00:00:00','08:00:00'),('2022-02-23 08:00:00','16:00:00'),('2022-02-23 16:00:00','00:00:00'),('2022-02-24 00:00:00','08:00:00'),('2022-02-24 08:00:00','16:00:00'),('2022-02-24 16:00:00','00:00:00'),('2022-02-25 00:00:00','08:00:00'),('2022-02-25 08:00:00','16:00:00'),('2022-02-25 16:00:00','00:00:00'),('2022-02-26 00:00:00','08:00:00'),('2022-02-26 08:00:00','16:00:00'),('2022-02-26 16:00:00','00:00:00'),('2022-02-27 00:00:00','08:00:00'),('2022-02-27 08:00:00','16:00:00'),('2022-02-27 16:00:00','00:00:00'),('2022-02-28 00:00:00','08:00:00'),('2022-02-28 08:00:00','16:00:00'),('2022-02-28 16:00:00','00:00:00'),('2022-03-01 00:00:00','08:00:00'),('2022-03-01 08:00:00','16:00:00'),('2022-03-01 16:00:00','00:00:00'),('2022-03-02 00:00:00','08:00:00'),('2022-03-02 08:00:00','16:00:00'),('2022-03-02 16:00:00','00:00:00'),('2022-03-03 00:00:00','08:00:00'),('2022-03-03 08:00:00','16:00:00'),('2022-03-03 16:00:00','00:00:00'),('2022-03-04 00:00:00','08:00:00'),('2022-03-04 08:00:00','16:00:00'),('2022-03-04 16:00:00','00:00:00'),('2022-03-05 00:00:00','08:00:00'),('2022-03-05 08:00:00','16:00:00'),('2022-03-05 16:00:00','00:00:00'),('2022-03-06 00:00:00','08:00:00'),('2022-03-06 08:00:00','16:00:00'),('2022-03-06 16:00:00','00:00:00'),('2022-03-07 00:00:00','08:00:00'),('2022-03-07 08:00:00','16:00:00'),('2022-03-07 16:00:00','00:00:00'),('2022-03-08 00:00:00','08:00:00'),('2022-03-08 08:00:00','16:00:00'),('2022-03-08 16:00:00','00:00:00'),('2022-03-09 00:00:00','08:00:00'),('2022-03-09 08:00:00','16:00:00'),('2022-03-09 16:00:00','00:00:00'),('2022-03-10 00:00:00','08:00:00'),('2022-03-10 08:00:00','16:00:00'),('2022-03-10 16:00:00','00:00:00'),('2022-03-11 00:00:00','08:00:00'),('2022-03-11 08:00:00','16:00:00'),('2022-03-11 16:00:00','00:00:00'),('2022-03-12 00:00:00','08:00:00'),('2022-03-12 08:00:00','16:00:00'),('2022-03-12 16:00:00','00:00:00'),('2022-03-13 00:00:00','08:00:00'),('2022-03-13 08:00:00','16:00:00'),('2022-03-13 16:00:00','00:00:00'),('2022-03-14 00:00:00','08:00:00'),('2022-03-14 08:00:00','16:00:00'),('2022-03-14 16:00:00','00:00:00'),('2022-03-15 00:00:00','08:00:00'),('2022-03-15 08:00:00','16:00:00'),('2022-03-15 16:00:00','00:00:00'),('2022-03-16 00:00:00','08:00:00'),('2022-03-16 08:00:00','16:00:00'),('2022-03-16 16:00:00','00:00:00'),('2022-03-17 00:00:00','08:00:00'),('2022-03-17 08:00:00','16:00:00'),('2022-03-17 16:00:00','00:00:00'),('2022-03-18 00:00:00','08:00:00'),('2022-03-18 08:00:00','16:00:00'),('2022-03-18 16:00:00','00:00:00'),('2022-03-19 00:00:00','08:00:00'),('2022-03-19 08:00:00','16:00:00'),('2022-03-19 16:00:00','00:00:00'),('2022-03-20 00:00:00','08:00:00'),('2022-03-20 08:00:00','16:00:00'),('2022-03-20 16:00:00','00:00:00'),('2022-03-21 00:00:00','08:00:00'),('2022-03-21 08:00:00','16:00:00'),('2022-03-21 16:00:00','00:00:00'),('2022-03-22 00:00:00','08:00:00'),('2022-03-22 08:00:00','16:00:00'),('2022-03-22 16:00:00','00:00:00'),('2022-03-23 00:00:00','08:00:00'),('2022-03-23 08:00:00','16:00:00'),('2022-03-23 16:00:00','00:00:00'),('2022-03-24 00:00:00','08:00:00'),('2022-03-24 08:00:00','16:00:00'),('2022-03-24 16:00:00','00:00:00'),('2022-03-25 00:00:00','08:00:00'),('2022-03-25 08:00:00','16:00:00'),('2022-03-25 16:00:00','00:00:00'),('2022-03-26 00:00:00','08:00:00'),('2022-03-26 08:00:00','16:00:00'),('2022-03-26 16:00:00','00:00:00'),('2022-03-27 00:00:00','08:00:00'),('2022-03-27 08:00:00','16:00:00'),('2022-03-27 16:00:00','00:00:00'),('2022-03-28 00:00:00','08:00:00'),('2022-03-28

```
08:00:00','16:00:00'),('2022-03-28 16:00:00','00:00:00'),('2022-03-29 00:00:00','08:00:00'),('2022-03-29 08:00:00','16:00:00'),('2022-03-29 16:00:00','00:00:00'),('2022-03-30 00:00:00','08:00:00'),('2022-03-30 08:00:00','16:00:00'),('2022-03-30 16:00:00','00:00:00'),('2022-03-31 00:00:00','08:00:00'),('2022-03-31 08:00:00','16:00:00'),('2022-03-31 16:00:00','00:00:00');
/*!40000 ALTER TABLE `turni` ENABLE KEYS */;
UNLOCK TABLES;
/*!40103 SET TIME_ZONE=@OLD_TIME_ZONE */;

/*!40101 SET SQL_MODE=@OLD_SQL_MODE */;
/*!40014 SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS */;
/*!40014 SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS */;
/*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;
/*!40101 SET CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS */;
/*!40101 SET COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION */;
/*!40111 SET SQL_NOTES=@OLD_SQL_NOTES */;

-- Dump completed on 2022-02-06 22:59:48
```

Codice del Front-End

Codice file defines.h

```
#pragma once
#include <stdbool.h>
#include <mysql.h>
struct configuration {
	char *host;
	char *db_username;
	char *db_password;
	unsigned int port;
	char *database;
	char username[128];
	char password[128];
};
typedef bool my_bool;
typedef enum {
	BARISTA=1,
	MANAGER=2,
	PIZZAIOLO=3,
	CAMERIERE=4,
	FAILED_LOGIN=5
}ruolo;
#define USERLENGTH 45
#define PASSLENGTH 45
typedef struct credential{
	char username[USERLENGTH];
	char password[PASSLENGTH];
}credenziali;
typedef struct Ordini{
	int cliente;
	int tavolo;
	MYSQL_TIME t;
```

```
        struct Ordini * next;
}ordini;
typedef struct ingrediente{
        char nome[16];
        int quantita;
        int costo;
        struct ingrediente * next;
}ingrediente;
typedef struct prodotto{
        int id;
        char nome[16];
        int prezzo;
        int tipo;
        int quantita;
        int CostoTotaleProdotto;
        ingrediente * listaIngredienti;
        struct prodotto* next;
}prodotto;
#define NOMELENGTH 15
#define COGNOMELENGTH 15
typedef struct cameriere{
        char nome[NOMELENGTH];
        char cognome[COGNOMELENGTH];
        struct cameriere * next;
}cameriere;
typedef struct Turno{
        MYSQL_TIME tempo;
        MYSQL_TIME oraFine;
        struct Turno * next;
}turno;
typedef struct Cliente{
        char nome[NOMELENGTH];
        char cognome[COGNOMELENGTH];
        int N_Persone;
        int Tavolo;
        MYSQL_TIME Turno;
}cliente;
typedef struct Tavolo{
        int N_Tavolo;
        MYSQL_TIME turno;
        struct Tavolo *next;
}Tavolo;
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);
```

codice file inout.c

```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <termios.h>
#include <sys/ioctl.h>
#include <pthread.h>
#include <signal.h>
#include <stdbool.h>
#include "defines.h"
// Per la gestione dei segnali
static volatile sig_atomic_t signo;
typedef struct sigaction sigaction_t;
static void handler(int s);
char *getInput(unsigned int lung, char *stringa, bool hide)
{
        char c;
        unsigned int i;


        // Dichiara le variabili necessarie ad un possibile mascheramento dell'input
        sigaction_t sa, savealrm, saveint, savehup, savequit, saveterm;
        sigaction_t savetstp, savettin, savettou;
        struct termios term, oterm;
        if(hide) {
        // Svuota il buffer
        (void) fflush(stdout);


        // Cattura i segnali che altrimenti potrebbero far terminare il programma, lasciando l'utente senza output sulla shell
        sigemptyset(&sa.sa_mask);
        sa.sa_flags = SA_INTERRUPT; // Per non resettare le system call
        sa.sa_handler = handler;
        (void) sigaction(SIGALRM, &sa, &savealrm);
        (void) sigaction(SIGINT, &sa, &saveint);
        (void) sigaction(SIGHUP, &sa, &savehup);
        (void) sigaction(SIGQUIT, &sa, &savequit);
        (void) sigaction(SIGTERM, &sa, &saveterm);
        (void) sigaction(SIGTSTP, &sa, &savetstp);
        (void) sigaction(SIGTTIN, &sa, &savettin);
        (void) sigaction(SIGTTOU, &sa, &savettou);
```

```
// Disattiva l'output su schermo
if (tcgetattr(fileno(stdin), &oterm) == 0) {
        (void) memcpy(&term, &oterm, sizeof(struct termios));
        term.c_lflag &= ~(ECHO|ECHONL);
        (void) tcsetattr(fileno(stdin), TCSAFLUSH, &term);
} else {
        (void) memset(&term, 0, sizeof(struct termios));
        (void) memset(&oterm, 0, sizeof(struct termios));
        }
}
// Acquisisce da tastiera al pi 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 pi 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  stata data
```

```
if(c == '\0') { // getInput() non pu 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 possibilit
        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  un carattere valido
        for(i = 0; i < num; i++) {
                if(c == choices[i])
        return c;
        }
        }
}
```

Codice main.c

```
#include <stdio.h>
```

```c
#include "../util/defines.h"
#include "../controller/controllerLogin.h"
#include "../controller/controllerManager.h"
#include "../controller/controllerCameriere.h"
#include "../controller/controllerPizzaiolo.h"
#include "../controller/controllerBarista.h"
int main(void)
{
	do{
		ruolo c = login();
		switch(c) {
			case CAMERIERE:
				run_as_cameriere();
			break;
			case MANAGER:
				run_Manager();
			break;
			case BARISTA:
				run_barista();
			break;
			case PIZZAIOLO:
				run_pizzaiolo();
			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();
		}
	}while(1);
}
```

Codice file baristaView.h

```c
 #pragma once
#include "../util/defines.h"
extern void ViewOrdini(prodotto * testa);
extern int ViewBevandePreparata(void);
extern int ViewBarista(void);
```

Codice file BaristaView.c

```c
#include "../util/defines.h"
#include "baristaView.h"
#include <stdio.h>
void ViewOrdini(prodotto * testa){
	system("clear");
	printf("--------------Ordini Barista------------\n");
	for (prodotto * i = testa; i!=NULL; i=i->next)
```

```
        {
                printf("%d] %s %d\n",i->prezzo,i->nome,i->tipo);
        }
        return;
}
int ViewBevandePreparata(){
        int i;
        system("clear");
        printf("inserire il numero della Bevanda pronta per la consegna: ");
        char app[4];
        do{
                getInput(4,app,false);
                i=atoi(app);
        }while(i<=0);
        return i;
}
int ViewBarista(){
        system("clear");
        printf("################################################\n");
        printf("## ##\n");
        printf("## Barista ##\n");
        printf("## ##\n");
        printf("################################################\n");
        printf("1]Visualizza ordini da preparare\n");
        printf("2]Ordine Pronto\n");
        printf("3]Cambia Utente\n");
        printf("4]Chiudi Applicazione\n");
        int c;
        char app[3];
        do{
                getInput(3,app,false);
                c=atoi(app);
        }while(c<1 ||c>4);
        return c;
}
```

Codice file camerieреView.h

```
#pragma once
#include "../util/defines.h"

extern int ViewOrdiniDaConsegnare(ordini * list,int z);
extern int ViewSelezionareTavolo();
extern void ViewTavoliAssegnati(int * a,int d);
extern int ViewCameriere();
extern cameriere* viewDatiCameriere();
extern char * OrdineCamerieреBevande(prodotto * lista);
extern char* OrdineCamerierePizze(ingrediente * aggiunte,prodotto * lista);
```

Codice file camerieреView.c

```c
#include "../util/defines.h"
#include "cameriereView.h"
#include <stdio.h>
#include <string.h>
int ViewSelezionareTavolo(){
	int i;
	char a[4];
	system("clear");
	printf("Inserire Tavolo: ");
	do{
		getInput(4,a,false);
		i=atoi(a);
	}while(i<1);
	return i;
}
void ViewTavoliAssegnati(int * a,int d){
	int i=1;
	system("clear");
	printf("Tavoli Assegnati\n");
	while(i<=d)
	{
		printf("%d]Tavolo numero:%d\n",i,a[i-1]);
		i++;
	}
	getchar();
}
cameriere * viewDatiCameriere(){
	cameriere * c=malloc(sizeof(cameriere));
	printf("Nome Cameriere: ");
	getInput(16,c->nome,false);
	printf("Cognome Cameriere:");
	getInput(16,c->cognome,false);
	return c;
}
char * viewAggiunta(ingrediente * aggiunte){
	int dim=0;
	char *s=malloc(256);
	memset(s,0,256);
	ingrediente *p;
	if(!yesOrNo("Si Vuole Inserire Aggiunte alla pizza?",'y','n',true,false)){
		strcpy(s,"*");
		return s;
	}
	do{
		int i=1;
		for(p=aggiunte;p!=NULL;p=p->next)
		{
			printf("%d]%s %d\n",i,p->nome,p->costo);
			i++;
		}
```

```
                int number;
                char n[4];
                printf("Digitare numero Aggiunta: ");
                do{
                        getInput(4,n,false);
                        number=atoi(n);
                }while(number<1 || number>=i);
                p=aggiunte;
                i=1;
                while(i<number){
                        p=p->next;
                        i=i+1;
                }
                strncpy(s+dim,p->nome,strlen(p->nome));
                dim=dim+strlen(p->nome);
                strncpy(s+dim,"?",2);
                dim=dim+1;
                printf("Quantita dell'agginta? ");
                do{
                        getInput(4,n,false);
                }while(atoi(n)<=0);
                strncpy(s+dim,n,strlen(n));
                dim=dim+strlen(n);
                strncpy(s+dim,"#",2);
                dim=dim+1;
        }while(yesOrNo("Vuoi Inserire Altre Aggiunte per questo prodoto?",'y','n',true,false) &&
        dim<256);
        strncpy(s+dim,"*",2);
        return s;
}
char* OrdineCamerierePizze(ingrediente * aggiunte,prodotto * lista){
        char * s=malloc(2048);
        memset(s,0,2048);
        int dim=0;
        do{
                int i=1;
                prodotto *p;
                char *s2;
                for(p=lista;p!=NULL;p=p->next)
                {
                        if(p->tipo==1){
                                printf("%d]%s prezzo: %d\n",i,p->nome,p->prezzo);
                                i++;
                        }
                }
                int number;
                char n[4];
                printf("Digitare numero prodotto: ");
                do{
                        getInput(4,n,false);
                        number=atoi(n);
```

```
		}while(number<1 || number>=i);
		p=lista;
		i=0;
		while(i<number){
			if(p->tipo==1){
				i=i+1;
				if(i!=number) p=p->next;
			}
			else p=p->next;
		}
		strncpy(s+dim,p->nome,strlen(p->nome));
		dim=dim+strlen(p->nome);
		strncpy(s+dim,"?",2);
		dim=dim+1;
		s2=viewAggiunta(aggiunte);
		printf("quante pizze cosi? ");
		do{
			getInput(4,n,false);
		}while(atoi(n)<=0);
		strncpy(s+dim,n,strlen(n)+1);
		dim=dim+strlen(n);
		strncpy(s+dim,"#",2);
		dim=dim+1;
		strncpy(s+dim,s2,strlen(s2));
		dim=dim+strlen(s2);
		free(s2);
	}while(yesOrNo("Vuoi Inserire Altri Prodotti?",'y','n',true,false) && dim<2048);
	strncpy(s+dim,"@",2);
	return s;
}
char * OrdineCameriereBevande(prodotto * lista){
	char * s=malloc(2048);
	memset(s,0,2048);
	int dim=0;
	system("clear");
	do{
		int i=1;
		prodotto *p;
		for(p=lista;p!=NULL;p=p->next)
		{
			if(p->tipo==0){
				printf("%d]%s prezzo: %d\n",i,p->nome,p->prezzo);
				i++;
			}
		}
		int number;
		char n[4];
		printf("Digitare numero prodotto: ");
		do{
			getInput(4,n,false);
			number=atoi(n);
```

```
                }while(number<1 || number>=i);
                p=lista;
                i=0;
                while(i<number){
                        if(p->tipo==0){
                                i=i+1;
                                if(i!=number) p=p->next;
                        }
                        else p=p->next;
                }
                printf("%s\n",p->nome);
                strncpy(s+dim,p->nome,strlen(p->nome));
                dim=dim+strlen(p->nome);
                strncpy(s+dim,"?",2);
                dim=dim+1;
                printf("quantita? ");
                do{
                        getInput(4,n,false);
                }while(atoi(n)<=0);
                strncpy(s+dim,n,strlen(n)+1);
                dim=dim+strlen(n);
                strncpy(s+dim,"#",2);
                dim=dim+1;
        }while(yesOrNo("Vuoi Inserire Altri Prodotti?",'y','n',true,false) && dim<2048);
        strncpy(s+dim,"@",2);
        return s;
}
int ViewOrdiniDaConsegnare(ordini * list,int z)
{
        int j=1;
        system("clear");
        printf("-----Lista Ordini Da COnsegnare-----\n");
        for (ordini* i = list; i!=NULL; i=i->next)
        {
                printf("%d]Tavolo %d ordine delle ore %2d:%2d\n",j,i->tavolo,(i->t).hour,(i-
                >t).minute);
                j++;
        }
        int c=0;
        if(z!=0){
                char app[4];
                printf("Inserire numero comanda consegnata: ");
                do{
                        getInput(4,app,false);
                        c=atoi(app);
                }while(c<=0 || c>=j);
        }
        return c;
}
int ViewCameriere(){
        system("clear");
```

```
	printf("################################################\n");
	printf("## ##\n");
	printf("## Cameriere ##\n");
	printf("## ##\n");
	printf("################################################\n");
	printf("1]Visualizza Tavoli Assegnati nel turno corrente\n");
	printf("2]Prendi ordine Tavolo\n");
	printf("3]Visualizza ordini pronti\n");
	printf("4]Consegnare Ordine\n");
	printf("5]Cambiare Utente\n");
	printf("6]Chiudere Programma\n");
	int c;
	char app[3];
	do{
		getInput(3,app,false);
		c=atoi(app);
	}while(c<1 ||c>6);
	return c;
}
```

Codice file ControllerBarista.h

```
#include "../util/defines.h"
#pragma once
extern void run_barista();
```

Codice file ControllerBarista.c

```
#include "../util/defines.h"
#include "../model/db.h"
#include "../View/baristaView.h"
#include "controllerBarista.h"
#include <stdio.h>

void bevandeDaPreparare(){
	prodotto * Testa=NULL;
	dbBevandeDaCucinare(&Testa);
	if(Testa==NULL){
		printf("Nessuna Bevanda da Preparare\n");
		printf("premere invio per continuare...\n");
		getchar();
		return;
		}
	ViewOrdini(Testa);
	freeProdotti(Testa);
	printf("premere invio per continuare...\n");
	getchar();
	return;
}
void OrdineProntoBevande(){
	int i=ViewBevandePreparata();
	dbBevandaPronta(i);
```

```
	printf("premere invio per continuare...\n");
	getchar();
	return;
}
void run_barista(){
	do{
		int i=ViewBarista();
		switch (i)
		{
			case 1:
				bevandeDaPreparare();
			break;
			case 2:
				OrdineProntoBevande();
			break;
			case 3:
				return;
			break;
			case 4:
				exit(1);
			break;
			default:
				fprintf(stderr, "Invalid condition at %s:%d\n", __FILE__, __LINE__);
				abort();
			break;
		}
	}while(1);
}
```

Codice file controllerCameriere.h

```
#pragma once
#include "../util/defines.h"

extern void run_as_cameriere();
```

Codice file controllerCameriere.c

```
#include "../util/defines.h"
#include "../model/db.h"
#include "../View/cameriereView.h"
#include "controllerCameriere.h"
#include <string.h>
#include <stdio.h>


cameriere io;
void TavoliAssegnati(){
	int dim;
	int * i= dbViewTavoli(io,&dim);
	if(i==NULL){
		printf("Tavoli non Assegnati\n");
```

```
                printf("premere invio per continuare...\n");
                getchar();
                return;
        }
        ViewTavoliAssegnati(i,dim);
        printf("premere invio per continuare...\n");
        getchar();
}
void prendereOrdine(){
        int t=ViewSelezionareTavolo();
        int c=dbClienteTavolo(t);
        printf("codice cliente:%d\n",c);
        if(c==0){
                printf("Errore nessun Nuovo cliente seduto a questo tavolo\n");
                printf("Premere invio per continuare...");
                getchar();
                return;
        }
        ingrediente * ListaAggiunte=NULL;
        dbAggiunte(&ListaAggiunte);
        if(ListaAggiunte==NULL) return;
        prodotto * menu=NULL;
        dbListaMenu(&menu);
        if(menu==NULL) return;
        char *pizze=OrdineCamerierePizze(ListaAggiunte,menu);
        char *bevande=OrdineCameriereBevande(menu);
        dbAddNewOrdine(c,pizze,bevande);
        printf("premere invio per continuare...\n");
        getchar();
        free(bevande);
        free(pizze);
}
void OrdiniDaConsegnare(){
        ordini * list=NULL;
        dbOrdiniDaConsegnare(&list,io);
        if(list==NULL){
                printf("Non Ci sono Comande Completate\n");
                printf("premere invio per continuare...\n");
                getchar();
                return;
        }
        ViewOrdiniDaConsegnare(list,0);
        printf("premere invio per continuare...\n");
        getchar();
}
void Ordine_consegnato(){
        ordini * list=NULL;
        dbOrdiniDaConsegnare(&list,io);
        if(list==NULL){
                printf("Non Ci sono Comande Completate\n");
                printf("premere invio per continuare...\n");
```

```
                getchar();
                return;
        }
        int c=ViewOrdiniDaConsegnare(list,1);
        int i=1;
        ordini * p=list;
        while(i<c){
                i++;
                p=p->next;
        }
        dbUpdateComanda(*p);
        printf("premere invio per continuare...\n");
        getchar();
        return ;
}
void run_as_cameriere(){
        do{
                cameriere * c=viewDatiCameriere();
                strcpy(io.nome,c->nome);
                strcpy(io.cognome,c->cognome);
                free(c);
        }while(!dbVerificaCameriere(io));
        do{
                int i=ViewCameriere();
                switch (i)
                {
                        case 1:
                                TavoliAssegnati();
                        break;
                        case 2:
                                prendereOrdine();
                        break;
                        case 3:
                                OrdiniDaConsegnare();
                        break;
                        case 4:
                                Ordine_consegnato();
                        break;
                        case 5:
                                return;
                        break;
                        case 6:
                                exit(1);
                        break;
                        default:
                                fprintf(stderr, "Invalid condition at %s:%d\n", __FILE__, __LINE__);
                                abort();
                        break;
                }
        }while(1);
}
```

Codice file controllerLogin.h

```
#pragma once
#include "defines.h"
extern ruolo login();
```

Codice file controllerLogin.c

```
#include <stdio.h>
#include "defines.h"
#include "loginView.h"
#include "db.h"
ruolo login()
{
	ruolo r;
	credenziali cred;
	loginView(&cred);
	r=dblogin(cred);
	return r;
}
```

Codice file controllerManager.h

```
#pragma once
#include "../util/defines.h"
extern void run_Manager(void);
```

Codice file controllerManager.c

```
#include "../util/defines.h"
#include "../View/managerView.h"
#include "../model/db.h"
#include "controllerManager.h"
#include <stdio.h>

void freeCameriere(cameriere * Testa)
{
	int c=0;
	cameriere * p=Testa;
	while (p!=NULL){
		p=p->next;
		c++;
	}
	for (int i = 0; i < c; i++){
		p=Testa;
		for (int j =0; j < c-i-1; j++)
		{
			p=p->next;
		}
		free(p->next);
```

```
                p->next=NULL;
                }
        }
        void freeTurno(turno * Testa)
        {
                int c=0;
                turno * p=Testa;
                while (p!=NULL){
                        p=p->next;
                        c++;
                }
                for (int i = 0; i < c; i++){
                        p=Testa;
                        for (int j =0; j < c-i-1; j++)
                        {
                                p=p->next;
                        }
                        free(p->next);
                        p->next=NULL;
                }
        }
void Creare_Turni(){
        int * a=ViewCreareTurni();
        if(dbCreaTurno(a[0],a[1])!=0){
                printf("error\n");
                printf("premere invio per continuare\n");
        }
        free(a);
}
void InserireCliente(){
        cliente new;
        Tavolo* testaLista=NULL;
        viewADDCliente(&new);
        dbListaTavoli(&testaLista,new.N_Persone);
        if(testaLista!=NULL){
                Tavolo tavoloScelto=ViewtavoliLiberi(testaLista);
                new.Tavolo=tavoloScelto.N_Tavolo;
                new.Turno=tavoloScelto.turno;
                dbAddCliente(new);
        }
        else printf("Non Ci sono Tavoli Liberi\n");
        printf("premere invio per continuare...\n");
        getchar();
}
void AggiungereTavolo(){
        int n=ViewTavolo();
        if(dbAddTavolo(n)!=0)
        printf("error\n");
        printf("premere invio per continuare\n");
        getchar();
        return;
```

```
}
void AssegnareCamerieraTavolo(){
	cameriere * listaCameriere=NULL;
	dbListCamerieri(&listaCameriere);
	turno * listaTurno=NULL;
	dbListTurni(&listaTurno);
	cameriere cameriere=viewCameriere(listaCameriere);
	turno * scelto=viewTurni(listaTurno);
	int t;
	printf("Quale Tavolo? ");
	char app[4];
	do{
		getInput(4,app,false);
		t=atoi(app);
	}while(t<=0);
	dbAssegnaTavoloTurno(cameriere,*scelto,t);
	printf("premere invio per continuare\n");
	getchar();
	freeCameriere(listaCameriere);
	freeTurno(listaTurno);
}
void AggiungiCameriere(){
	cameriere c;
	ViewAddCameriere(&c);
	dbAddCameriere(c);
	printf("premere invio per continuare\n");
	getchar();
}
void TurnoTavolo(){
	int tavolo=viewTurnoTavoli();
	turno * lista=NULL;
	dbListTurni(&lista);
	turno* scelto=viewTurni(lista);
	dbAssegnaTavoli(tavolo,*scelto);
	printf("premere invio per continuare\n");
	getchar();
	freeTurno(lista);
};
void InserireNuovoElementoMenu(){
	prodotto p=viewAddNuovoElemento();
	dbAddElementoMenu(p);
	printf("premere invio per continuare\n");
	getchar();
};
void AggiungereIngredienti(){
	if(viewIngredienti()==1){
		ingrediente i=viewAddIngrediente();
		dbAddIngrediente(i);
	}
	else{
		ingrediente *listaIngredienti=NULL;
```

```
                viewUpdateIngrediente(&listaIngredienti);
                if(listaIngredienti==NULL)
                {
                        printf("Error\n");
                        printf("premere invio per continuare\n");
                        getchar();
                        return;
                }
                while(listaIngredienti!=NULL)
                {
                        ingrediente * p=listaIngredienti;
                        dbUpdateIngredienti(*listaIngredienti);
                        listaIngredienti=listaIngredienti->next;
                        free(p);
                }
        }
        printf("premere invio per continuare...");
        getchar();
}
void statoAggiunte(){
        ingrediente * lista=NULL;
        dbIngredienti(&lista);
        if(lista!=NULL) ViewStatoIngredienti(lista);
        printf("premere invio per continuare\n");
        getchar();
}
void StampaScontrino(){
        int t = ViewTavoloCliente();
        int c =dbClienteTavolo(t);
        prodotto * Lista=NULL;
        int SpesaToT;
        dbStampaScontrino(&Lista,&SpesaToT,c);
        if(Lista==NULL)
        {
                printf("non ci sono ordini da parte di questo Cliente\n");
                getchar();
                return;
        }
        ViewScontrino(Lista,SpesaToT);
}
void IncassiGiorno(){
        int i=ViewGiorno();
        int s=dbIncassoGiorno(i);
        printf("l'incasso del giorno selezionato é di: %d\n",s);
        getchar();
}
void IncassiMese(){
        int i=ViewMese();
        int s=dbIncassoMese(i);
        printf("l'incasso del mese selezionato é di: %d\n",s);
        getchar();
```

```
}
void run_Manager(){
	do{
		int r=ViewManager();
		switch(r){
			case 1:
				InserireCliente();
			break;
			case 2:
				StampaScontrino();
			break;
			case 3:
				IncassiGiorno();
			break;
			case 4:
				IncassiMese();
			break;
			case 5:
				AggiungereIngredienti();
			break;
			case 6:
				AggiungiCameriere();
			break;
			case 7:
				InserireNuovoElementoMenu();
			break;
			case 8:
				statoAggiunte();
			break;
			case 9:
				AggiungereTavolo();
			break;
			case 10:
				AssegnareCamerieraTavolo();
			break;
			case 11:
				Creare_Turni();
			break;
			case 12:
				TurnoTavolo();
			break;
			case 13:
				return;
			break;
			case 14:
				exit(1);
			break;
			default:
				fprintf(stderr, "Invalid condition at %s:%d\n", __FILE__, __LINE__);
				abort();
			break;
```

```
		}
	}while(1);
};
```

Codice file controllerPizzaiolo.h

```
#pragma once
#include "../util/defines.h"
extern void run_pizzaiolo();
```

Codice file controllerPizzaiolo.c

```
#include "../util/defines.h"
#include "../model/db.h"
#include "../View/pizzaioloView.h"
#include "controllerPizzaiolo.h"
#include <stdio.h>
void InPreparazione(){
	prodotto * testa=NULL;
	dbOrdiniDaCucinare(&testa);
	if(testa==NULL)
	{
		printf("Nessuna Pizza in Programma\n");
		printf("premere invio per continuare...\n");
		getchar();
		return;
	}
	stampaListaPizze(testa);
	freeProdotti(testa);
	printf("premere invio per continuare...\n");
	getchar();
	return;
}
void OrdinePronto(){
	int i=ViewPizzaFinita();
	dbPizzaFinita(i);
	printf("premere invio per continuare...\n");
	getchar();
	return ;
}
void run_pizzaiolo(){
	do{
		int option=ViewPizzaiolo();
		switch (option)
		{
			case 1:
				InPreparazione();
			break;
			case 2:
				OrdinePronto();
			break;
```

```
                        case 3:
                                return;
                        break;
                        case 4:
                                exit(1);
                        break;
                        default:
                                fprintf(stderr, "Invalid condition at %s:%d\n", __FILE__, __LINE__);
                                abort();
                        break;
                }
        }while(1);
}
```

Codice file db.h

```
#pragma once
#include "../util/defines.h"
extern void freeIngredienti(ingrediente * lista);
extern void freeProdotti(prodotto * testa);
extern int dbClienteTavolo(int t);
extern int* dbViewTavoli(cameriere c,int * d);
extern void dbAddCliente(cliente n);
extern void dbAssegnaTavoloTurno(cameriere c,turno t, int tavolo);
extern void dbAssegnaTavoli(int i,turno scelto);
extern void dbAddElementoMenu(prodotto p);
extern void dbUpdateIngredienti(ingrediente i);
extern void dbAddIngrediente(ingrediente i);
extern void dbListTurni(turno **ListaTurni);
extern void dbListCamerieri(cameriere ** c);
extern int dbCreaTurno(int anno,int mese);
extern ruolo dblogin(credenziali cred);
extern void dbAddCameriere(cameriere c);
extern int dbAddTavolo(int n);
extern void dbListaTavoli(Tavolo ** Testa,int n);
extern void dbIngredienti(ingrediente ** Testa);
extern void dbAggiunte(ingrediente ** Testa);
extern void dbListaMenu(prodotto ** Testa);
extern void dbAddNewOrdine(int c,char * pizze,char * bevande);
extern void dbOrdiniDaCucinare(prodotto ** Testa);
extern void dbPizzaFinita(int i);
extern void dbBevandeDaCucinare(prodotto ** Testa);
extern void dbBevandaPronta(int i);
extern void dbOrdiniDaConsegnare(ordini ** list,cameriere c);
extern void dbUpdateComanda(ordini p);
extern void dbStampaScontrino(prodotto **Lista,int *SpesaToT,int c);
extern int dbIncassoGiorno(int i);
extern int dbIncassoMese(int i);
extern int dbVerificaCameriere(cameriere c);
```

Codice file db.c

```
#include <stdio.h>
#include <string.h>
#include "../util/defines.h"
#include “db.h”
void freeIngredienti(ingrediente * lista){
   if(lista==NULL) return;
   ingrediente * i=lista;
   while(i->next!=NULL)
   {
      ingrediente * app=i;
      while ((app->next)->next!=NULL) app=app->next;
      free(app->next);
      app->next=NULL;
   }
   free(i);
   return;
}
void freeProdotti(prodotto * testa){
   if(testa==NULL) return;
   prodotto * p=testa;
   while(p->next!=NULL)
   {
      prodotto * app=p;
      while ((app->next)->next!=NULL) app=app->next;
      freeIngredienti(app->next->listaIngredienti);
      free(app->next);
      app->next=NULL;
   }
   freeIngredienti(p->listaIngredienti);
   free(p);
   return;
}
struct configuration conf;
MYSQL *conn;
char* parsIngredienti(ingrediente *listaIngredienti){

        int dim=0;
        ingrediente *i=listaIngredienti;
        char a[4];
        while(i!=NULL){
                sprintf(a,"%d",i->quantita);
                dim=dim+strlen(i->nome)+2+ strlen(a);
                i=i->next;
        }
        dim=dim+1;
        i=listaIngredienti;
        char * s=malloc(sizeof(char)*dim);
        char *l=s;
        int j=0;
```

```
        while(i!=NULL){
                strcpy(s+j,i->nome);
                j=j+strlen(i->nome);
                strcpy(s+j,"?");
                j++;
                sprintf(a,"%d",i->quantita);
                strcpy(s+j,a);
                j=j+strlen(a);
                strcpy(s+j,"#");
                j++;
                i=i->next;
        }
        strcpy(s+j,"@");
        return  l;
}
void dbAssegnaTavoloTurno(cameriere c,turno t, int tavolo)
{
if(!parse_config("users/Manager.json", &conf)) {
            fprintf(stderr, "Unable to load login configuration\n");
            exit(EXIT_FAILURE);
      }
conn = mysql_init (NULL);
   if (conn == NULL) {
            fprintf (stderr, "mysql_init() failed (probably out of memory)\n");
            exit(EXIT_FAILURE);
      }

    if (mysql_real_connect(conn, conf.host, conf.db_username, conf.db_password, conf.database,
conf.port, NULL, CLIENT_MULTI_STATEMENTS | CLIENT_MULTI_RESULTS) == NULL) {
            fprintf (stderr, "mysql_real_connect() failed\n");
            mysql_close (conn);
            exit(EXIT_FAILURE);
      }

MYSQL_STMT *AssegnaTurnoTavolo;
MYSQL_BIND param[4]; // Used both for input and output

if(!setup_prepared_stmt(&AssegnaTurnoTavolo, "call AssegnareTavoloCameriere(?,?,?,?)", conn))
{
        print_stmt_error(AssegnaTurnoTavolo, "Unable to initialize login statement\n");
        goto err2;
}
// Prepare parameters
memset(param, 0, sizeof(param));

param[0].buffer_type = MYSQL_TYPE_VAR_STRING; // IN
param[0].buffer = c.nome;
param[0].buffer_length = strlen(c.nome);

param[1].buffer_type = MYSQL_TYPE_VAR_STRING; // IN
param[1].buffer = c.cognome;
```

```
param[1].buffer_length = strlen(c.cognome);

param[2].buffer_type = MYSQL_TYPE_LONG; // IN
param[2].buffer = &tavolo;
param[2].buffer_length = sizeof(tavolo);

param[3].buffer_type = MYSQL_TYPE_DATETIME; // IN
param[3].buffer = (char*)&(t.tempo);

if (mysql_stmt_bind_param(AssegnaTurnoTavolo, param) != 0) { // Note _param
        print_stmt_error(AssegnaTurnoTavolo, "Could not bind parameters for login");
        goto err;
}

// Run procedure
if (mysql_stmt_execute(AssegnaTurnoTavolo) != 0) {
        print_stmt_error(AssegnaTurnoTavolo, "Could not execute login procedure\n");
        goto err;
}
mysql_stmt_close(AssegnaTurnoTavolo);
mysql_close(conn);
return ;

err:
mysql_stmt_close(AssegnaTurnoTavolo);
mysql_close(conn);
err2:
return ;
}
void dbAssegnaTavoli(int i,turno scelto){
if(!parse_config("users/Manager.json", &conf)) {
            fprintf(stderr, "Unable to load login configuration\n");
            exit(EXIT_FAILURE);
      }
conn = mysql_init (NULL);
   if (conn == NULL) {
            fprintf (stderr, "mysql_init() failed (probably out of memory)\n");
            exit(EXIT_FAILURE);
      }

    if (mysql_real_connect(conn, conf.host, conf.db_username, conf.db_password, conf.database,
conf.port, NULL, CLIENT_MULTI_STATEMENTS | CLIENT_MULTI_RESULTS) == NULL) {
            fprintf (stderr, "mysql_real_connect() failed\n");
            mysql_close (conn);
            exit(EXIT_FAILURE);
      }

MYSQL_STMT *AssegnaTurnoTavolo;
MYSQL_BIND param[2]; // Used both for input and output

if(!setup_prepared_stmt(&AssegnaTurnoTavolo, "call AssegnaTurnoTavolo(?,?)", conn)) {
```

```
        print_stmt_error(AssegnaTurnoTavolo, "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 = &i;
param[0].buffer_length = sizeof(i);

param[1].buffer_type = MYSQL_TYPE_DATETIME; // IN
param[1].buffer = (char*)&(scelto.tempo);

if (mysql_stmt_bind_param(AssegnaTurnoTavolo, param) != 0) { // Note _param
        print_stmt_error(AssegnaTurnoTavolo, "Could not bind parameters for login");
        goto err;
}

// Run procedure
if (mysql_stmt_execute(AssegnaTurnoTavolo) != 0) {
        print_stmt_error(AssegnaTurnoTavolo, "Could not execute login procedure");
        goto err;
}
mysql_stmt_close(AssegnaTurnoTavolo);
mysql_close(conn);
return ;

err:
mysql_stmt_close(AssegnaTurnoTavolo);
mysql_close(conn);
err2:
return ;
};
void dbAddElementoMenu(prodotto p){
        if(!parse_config("users/Manager.json", &conf)) {
                fprintf(stderr, "Unable to load login configuration\n");
                exit(EXIT_FAILURE);
            }
        conn = mysql_init (NULL);
           if (conn == NULL) {
                fprintf (stderr, "mysql_init() failed (probably out of memory)\n");
                exit(EXIT_FAILURE);
            }

            if (mysql_real_connect(conn, conf.host, conf.db_username, conf.db_password,
        conf.database, conf.port, NULL, CLIENT_MULTI_STATEMENTS |
        CLIENT_MULTI_RESULTS) == NULL) {
                fprintf (stderr, "mysql_real_connect() failed\n");
                mysql_close (conn);
                exit(EXIT_FAILURE);
              }
```

```
	MYSQL_STMT *addProdotto;
	MYSQL_BIND param[4]; // Used both for input and output

	if(!setup_prepared_stmt(&addProdotto, "call addProdotto(?,?,?,?)", conn)) {
		print_stmt_error(addProdotto, "Unable to initialize login statement\n");
		goto err2;
	}
	// Prepare parameters
	memset(param, 0, sizeof(param));
	param[0].buffer_type = MYSQL_TYPE_VAR_STRING; // IN
	param[0].buffer = p.nome;
	param[0].buffer_length = strlen(p.nome);

	param[1].buffer_type = MYSQL_TYPE_LONG; // IN
	param[1].buffer = &p.prezzo;
	param[1].buffer_length = sizeof(p.prezzo);

	param[2].buffer_type = MYSQL_TYPE_LONG; // IN
	param[2].buffer = &p.tipo;
	param[2].buffer_length = sizeof(p.tipo);

	char* s=parsIngredienti(p.listaIngredienti);
	param[3].buffer_type = MYSQL_TYPE_VAR_STRING; // IN
	param[3].buffer = s;
	param[3].buffer_length = strlen(s);

	if (mysql_stmt_bind_param(addProdotto, param) != 0) { // Note _param
		print_stmt_error(addProdotto, "Could not bind parameters for login");
		goto err;
	}

	// Run procedure
	if (mysql_stmt_execute(addProdotto) != 0) {
		print_stmt_error(addProdotto, "Could not execute login procedure");
		goto err;
	}
	free(s);
	mysql_stmt_close(addProdotto);
	mysql_close(conn);
	return ;

	err:
	free(s);
	mysql_stmt_close(addProdotto);
	err2:
	mysql_close(conn);
	return ;
}
void dbUpdateIngredienti(ingrediente i){
	if(!parse_config("users/Manager.json", &conf)) {
			fprintf(stderr, "Unable to load login configuration\n");
```

```
            exit(EXIT_FAILURE);
        }
conn = mysql_init (NULL);
   if (conn == NULL) {
            fprintf (stderr, "mysql_init() failed (probably out of memory)\n");
            exit(EXIT_FAILURE);
        }

      if (mysql_real_connect(conn, conf.host, conf.db_username, conf.db_password,
conf.database, conf.port, NULL, CLIENT_MULTI_STATEMENTS |
CLIENT_MULTI_RESULTS) == NULL) {
            fprintf (stderr, "mysql_real_connect() failed\n");
            mysql_close (conn);
            exit(EXIT_FAILURE);
        }

MYSQL_STMT *IncrementaIngrediente;
MYSQL_BIND param[2]; // Used both for input and output

if(!setup_prepared_stmt(&IncrementaIngrediente, "call IncrementaIngrediente(?,?)", conn))
{
        print_stmt_error(IncrementaIngrediente, "Unable to initialize login statement\n");
        goto err2;
}
// Prepare parameters
memset(param, 0, sizeof(param));
param[0].buffer_type = MYSQL_TYPE_VAR_STRING; // IN
param[0].buffer = i.nome;
param[0].buffer_length = strlen(i.nome);

param[1].buffer_type = MYSQL_TYPE_LONG; // IN
param[1].buffer = &i.quantita;
param[1].buffer_length = sizeof(i.quantita);


if (mysql_stmt_bind_param(IncrementaIngrediente, param) != 0) { // Note _param
        print_stmt_error(IncrementaIngrediente, "Could not bind parameters for login");
        goto err;
}

// Run procedure
if (mysql_stmt_execute(IncrementaIngrediente) != 0) {
        print_stmt_error(IncrementaIngrediente, "Could not execute login procedure");
        goto err;
}
mysql_stmt_close(IncrementaIngrediente);
mysql_close(conn);
return ;

err:
mysql_stmt_close(IncrementaIngrediente);
```

```
	err2:
	mysql_close(conn);
	return ;
}
void dbAddIngrediente(ingrediente i){

	if(!parse_config("users/Manager.json", &conf)) {
			fprintf(stderr, "Unable to load login configuration\n");
			exit(EXIT_FAILURE);
		}
	conn = mysql_init (NULL);
	  if (conn == NULL) {
			fprintf (stderr, "mysql_init() failed (probably out of memory)\n");
			exit(EXIT_FAILURE);
		}

		if (mysql_real_connect(conn, conf.host, conf.db_username, conf.db_password,
	conf.database, conf.port, NULL, CLIENT_MULTI_STATEMENTS |
	CLIENT_MULTI_RESULTS) == NULL) {
			fprintf (stderr, "mysql_real_connect() failed\n");
			mysql_close (conn);
			exit(EXIT_FAILURE);
		}

	MYSQL_STMT *addIngredienti;
	MYSQL_BIND param[3]; // Used both for input and output

	if(!setup_prepared_stmt(&addIngredienti, "call addIngredienti(?,?,?)", conn)) {
		print_stmt_error(addIngredienti, "Unable to initialize login statement\n");
		goto err2;
	}
	// Prepare parameters
	memset(param, 0, sizeof(param));
	param[0].buffer_type = MYSQL_TYPE_VAR_STRING; // IN
	param[0].buffer = i.nome;
	param[0].buffer_length = strlen(i.nome);

	param[1].buffer_type = MYSQL_TYPE_LONG; // IN
	param[1].buffer = &i.quantita;
	param[1].buffer_length = sizeof(i.quantita);

	param[2].buffer_type = MYSQL_TYPE_LONG; // IN
	param[2].buffer = &i.costo;
	param[2].buffer_length = sizeof(i.costo);


	if (mysql_stmt_bind_param(addIngredienti, param) != 0) { // Note _param
		print_stmt_error(addIngredienti, "Could not bind parameters for login");
		goto err;
	}
```

```
        // Run procedure
        if (mysql_stmt_execute(addIngredienti) != 0) {
                print_stmt_error(addIngredienti, "Could not execute login procedure");
                goto err;
        }
        mysql_stmt_close(addIngredienti);
        mysql_close(conn);
        return ;

        err:
        mysql_stmt_close(addIngredienti);
        err2:
        mysql_close(conn);
        return ;
}
void dbAddCameriere(cameriere c){
if(!parse_config("users/Manager.json", &conf)) {
            fprintf(stderr, "Unable to load login configuration\n");
            exit(EXIT_FAILURE);
      }
conn = mysql_init (NULL);
      if (conn == NULL) {
            fprintf (stderr, "mysql_init() failed (probably out of memory)\n");
            exit(EXIT_FAILURE);
      }
  if (mysql_real_connect(conn, conf.host, conf.db_username, conf.db_password, conf.database,
conf.port, NULL, CLIENT_MULTI_STATEMENTS | CLIENT_MULTI_RESULTS) == NULL) {
            fprintf (stderr, "mysql_real_connect() failed\n");
            mysql_close (conn);
            exit(EXIT_FAILURE);
      }

MYSQL_STMT *Addcamerieri;
MYSQL_BIND param[2]; // Used both for input and output

if(!setup_prepared_stmt(&Addcamerieri, "call AddCamerieri(?,?)", conn)) {
print_stmt_error(Addcamerieri, "Unable to initialize login statement\n");
goto err2;
}
memset(param, 0, sizeof(param));
param[0].buffer_type = MYSQL_TYPE_VAR_STRING; // IN
param[0].buffer = c.nome;
param[0].buffer_length = strlen(c.nome);

param[1].buffer_type = MYSQL_TYPE_VAR_STRING; // IN
param[1].buffer = c.cognome;
param[1].buffer_length = strlen(c.cognome);

if (mysql_stmt_bind_param(Addcamerieri, param) != 0) { // Note _param
print_stmt_error(Addcamerieri, "Could not bind parameters for login");
goto err;
```

```
}

// Run procedure
if (mysql_stmt_execute(Addcamerieri) != 0) {
print_stmt_error(Addcamerieri, "Could not view Tavoli Liberi");
goto err;
}

mysql_stmt_close(Addcamerieri);
mysql_close(conn);
return;

   err:
mysql_stmt_close(Addcamerieri);
   err2:
mysql_close(conn);
return;

}
void dbListTurni(turno **ListaTurni){

        if(!parse_config("users/Manager.json", &conf)) {
                fprintf(stderr, "Unable to load login configuration\n");
                exit(EXIT_FAILURE);
        }
        conn = mysql_init (NULL);
              if (conn == NULL) {
                fprintf (stderr, "mysql_init() failed (probably out of memory)\n");
                exit(EXIT_FAILURE);
              }
             if (mysql_real_connect(conn, conf.host, conf.db_username, conf.db_password,
        conf.database, conf.port, NULL, CLIENT_MULTI_STATEMENTS |
        CLIENT_MULTI_RESULTS) == NULL) {
                fprintf (stderr, "mysql_real_connect() failed\n");
                mysql_close (conn);
                exit(EXIT_FAILURE);
              }

        MYSQL_STMT *ListTurni;
        if(!setup_prepared_stmt(&ListTurni, "call ListTurni()", conn)) {
                print_stmt_error(ListTurni, "Unable to initialize ListCamerieri statement\n");
                goto err2;
        }


        if (mysql_stmt_bind_param(ListTurni, NULL) != 0) { // Note _param
                print_stmt_error(ListTurni, "Could not bind parameters for ListCamerieri");
                goto err;
        }

        // Run procedure
```

```
if (mysql_stmt_execute(ListTurni) != 0) {
	print_stmt_error(ListTurni, "Could not view ListCamerieri");
	goto err;
}

MYSQL_BIND param[2];
MYSQL_TIME	ts;
MYSQL_TIME	ts2;

memset(param, 0, sizeof(param));
param[0].buffer_type= MYSQL_TYPE_TIMESTAMP;
param[0].buffer= (char *)&ts;

param[1].buffer_type= MYSQL_TYPE_TIMESTAMP;
param[1].buffer= (char *)&ts2;

if(mysql_stmt_bind_result(ListTurni,param)!=0)
{
	printf("empty result\n");
	goto err2;
}

if(mysql_stmt_store_result(ListTurni)!=0)
{
	printf("empty result\n");
	goto err2;
}
while (!mysql_stmt_fetch(ListTurni))
{
	printf(" %04d-%02d-%02d %02d:%02d:%02d \n",
			ts.year, ts.month, ts.day,
			ts.hour, ts.minute, ts.second);

	turno * app= malloc(sizeof(turno));
	if(app==NULL){
		printf("error Malloc\n");
	}
	app->next=NULL;
	if(*ListaTurni!=NULL)
	{
		turno * app2=*ListaTurni;
		while(app2->next!=NULL)
		app2=app2->next;
		app2->next=app;
	}
	else *ListaTurni=app;
	memcpy(&(app->tempo),&ts,sizeof(MYSQL_TIME));
	memcpy(&(app->oraFine),&ts2,sizeof(MYSQL_TIME));
}
mysql_stmt_close(ListTurni);
mysql_close(conn);
```

```
        return;

           err:
        mysql_stmt_close(ListTurni);
           err2:
        mysql_close(conn);
        return ;
}
void dbListCamerieri(cameriere ** c){

        if(!parse_config("users/Manager.json", &conf)) {
                fprintf(stderr, "Unable to load login configuration\n");
                exit(EXIT_FAILURE);
            }
        conn = mysql_init (NULL);
        if (conn == NULL) {
                fprintf (stderr, "mysql_init() failed (probably out of memory)\n");
                exit(EXIT_FAILURE);
            }
           if (mysql_real_connect(conn, conf.host, conf.db_username, conf.db_password,
        conf.database, conf.port, NULL, CLIENT_MULTI_STATEMENTS |
        CLIENT_MULTI_RESULTS) == NULL) {
                fprintf (stderr, "mysql_real_connect() failed\n");
                mysql_close (conn);
                exit(EXIT_FAILURE);
            }

        MYSQL_STMT *ListCamerieri;
        if(!setup_prepared_stmt(&ListCamerieri, "call ListCamerieri()", conn)) {
                print_stmt_error(ListCamerieri, "Unable to initialize ListCamerieri statement\n");
                goto err2;
        }


        if (mysql_stmt_bind_param(ListCamerieri, NULL) != 0) { // Note _param
                print_stmt_error(ListCamerieri, "Could not bind parameters for ListCamerieri");
                goto err;
        }

        // Run procedure
        if (mysql_stmt_execute(ListCamerieri) != 0) {
                print_stmt_error(ListCamerieri, "Could not view ListCamerieri");
                goto err;
        }
        MYSQL_BIND param[2];
        char nome[16];
        char cognome[16];

        memset(param, 0, sizeof(param));
        param[0].buffer_type = MYSQL_TYPE_VAR_STRING; // OUT
        param[0].buffer = nome;
```

```
	param[0].buffer_length = 15;

	param[1].buffer_type = MYSQL_TYPE_VAR_STRING; // OUT
	param[1].buffer = cognome;
	param[1].buffer_length = 15;

	if(mysql_stmt_bind_result(ListCamerieri,param)!=0)
	{
		printf("empty result\n");
	goto err2;
	}

	if(mysql_stmt_store_result(ListCamerieri)!=0)
	{
		printf("empty result\n");
	goto err;
	}

	while (!mysql_stmt_fetch(ListCamerieri))
	{
		cameriere * app= malloc(sizeof(cameriere));
		if(app==NULL){
			printf("error Malloc\n");
			goto err;
		}
		strcpy(app->nome,nome);
		strcpy(app->cognome,cognome);
		app->next=NULL;
		if(*c!=NULL)
		{
			cameriere * app2=*c;
			while(app2->next!=NULL) app2=app2->next;
			app2->next=app;
		}
		else *c=app;


		printf("cioa\n");
	}
	mysql_stmt_close(ListCamerieri);
	mysql_close(conn);
	return;

	   err:
	mysql_stmt_close(ListCamerieri);
	   err2:
	mysql_close(conn);
	return ;
}
int dbCreaTurno(int anno,int mese){
	if(!parse_config("users/Manager.json", &conf)) {
```

```
            fprintf(stderr, "Unable to load login configuration\n");
            exit(EXIT_FAILURE);
}
conn = mysql_init (NULL);
      if (conn == NULL) {
            fprintf (stderr, "mysql_init() failed (probably out of memory)\n");
            exit(EXIT_FAILURE);
      }
    if (mysql_real_connect(conn, conf.host, conf.db_username, conf.db_password,
conf.database, conf.port, NULL, CLIENT_MULTI_STATEMENTS |
CLIENT_MULTI_RESULTS) == NULL) {
            fprintf (stderr, "mysql_real_connect() failed\n");
            mysql_close (conn);
            exit(EXIT_FAILURE);
      }
MYSQL_STMT *Creare_Turno;
if(!setup_prepared_stmt(&Creare_Turno, "call Creare_Turno(?,?)", conn)) {
        print_stmt_error(Creare_Turno, "Unable to initialize Creare_Turno statement\n");
        goto err2;
}
MYSQL_BIND param[2]; // Used both for input and output
// Prepare parameters
memset(param, 0, sizeof(param));
param[0].buffer_type = MYSQL_TYPE_LONG; // IN
param[0].buffer = &mese;
param[0].buffer_length = sizeof(mese);

param[1].buffer_type = MYSQL_TYPE_LONG; // IN
param[1].buffer = &anno;
param[1].buffer_length = sizeof(anno);

if (mysql_stmt_bind_param(Creare_Turno, param) != 0) { // Note _param
        print_stmt_error(Creare_Turno, "Could not bind parameters for Creare_Turno");
        goto err;
}

// Run procedure
if (mysql_stmt_execute(Creare_Turno) != 0) {
        print_stmt_error(Creare_Turno, "Could not view Creare_Turno");
        goto err;
}

mysql_stmt_close(Creare_Turno);
mysql_close(conn);
return 0;

   err:
mysql_stmt_close(Creare_Turno);
   err2:
mysql_close(conn);
return 1;
```

```
}
ruolo dblogin(credenziali cred)
{

	if(!parse_config("users/login.json", &conf)) {
		fprintf(stderr, "Unable to load login configuration\n");
		exit(EXIT_FAILURE);
	}
	conn = mysql_init (NULL);
	if (conn == NULL) {
		fprintf (stderr, "mysql_init() failed (probably out of memory)\n");
		exit(EXIT_FAILURE);
	}

	  if (mysql_real_connect(conn, conf.host, conf.db_username, conf.db_password, conf.database, conf.port, NULL, CLIENT_MULTI_STATEMENTS | CLIENT_MULTI_RESULTS) == NULL) {
		fprintf (stderr, "mysql_real_connect() failed\n");
		mysql_close (conn);
		exit(EXIT_FAILURE);
	}


	MYSQL_STMT *login_procedure;
	MYSQL_BIND param[3]; // Used both for input and output
	int ruolo;

	if(!setup_prepared_stmt(&login_procedure, "call login(?, ?, ?)", conn)) {
		print_stmt_error(login_procedure, "Unable to initialize login statement\n");
		goto err2;
	}

	// Prepare parameters
	memset(param, 0, sizeof(param));

	param[0].buffer_type = MYSQL_TYPE_VAR_STRING; // IN
	param[0].buffer = cred.username;
	param[0].buffer_length = strlen(cred.username);

	param[1].buffer_type = MYSQL_TYPE_VAR_STRING; // IN
	param[1].buffer = cred.password;
	param[1].buffer_length = strlen(cred.password);

	param[2].buffer_type = MYSQL_TYPE_LONG; // OUT
	param[2].buffer = &ruolo;
	param[2].buffer_length = sizeof(ruolo);

	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 = &ruolo;
	param[0].buffer_length = sizeof(ruolo);

	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);
	mysql_close(conn);
	return ruolo;

	   err:
	mysql_stmt_close(login_procedure);
	mysql_close(conn);
	   err2:
	return FAILED_LOGIN;
}
int dbAddTavolo(int n)
{

	if(!parse_config("users/Manager.json", &conf)) {
			fprintf(stderr, "Unable to load login configuration\n");
			exit(EXIT_FAILURE);
		}
	conn = mysql_init (NULL);
	   if (conn == NULL) {
			fprintf (stderr, "mysql_init() failed (probably out of memory)\n");
			exit(EXIT_FAILURE);
		}

		if (mysql_real_connect(conn, conf.host, conf.db_username, conf.db_password,
	conf.database, conf.port, NULL, CLIENT_MULTI_STATEMENTS |
	CLIENT_MULTI_RESULTS) == NULL) {
			fprintf (stderr, "mysql_real_connect() failed\n");
			mysql_close (conn);
```

```
                exit(EXIT_FAILURE);
        }

    MYSQL_STMT *addTavolo;
    MYSQL_BIND param; // Used both for input and output

    if(!setup_prepared_stmt(&addTavolo, "call AggiungiTavolo(?)", conn)) {
            print_stmt_error(addTavolo, "Unable to initialize login statement\n");
            goto err2;
    }

    // Prepare parameters
    memset(&param, 0, sizeof(param));
    param.buffer_type = MYSQL_TYPE_LONG; // OUT
    param.buffer = &n;
    param.buffer_length = sizeof(n);

    if (mysql_stmt_bind_param(addTavolo, &param) != 0) { // Note _param
            print_stmt_error(addTavolo, "Could not bind parameters for login");
            goto err;
    }

    // Run procedure
    if (mysql_stmt_execute(addTavolo) != 0) {
            print_stmt_error(addTavolo, "Could not execute login procedure");
            goto err;
    }
    mysql_stmt_close(addTavolo);
    mysql_close(conn);
    return 0;

    err:
    mysql_stmt_close(addTavolo);
    mysql_close(conn);
    err2:
    return 1;
}
void dbAddCliente(cliente n){
    if(!parse_config("users/Manager.json", &conf)) {
                fprintf(stderr, "Unable to load login configuration\n");
                exit(EXIT_FAILURE);
        }
    conn = mysql_init (NULL);
        if (conn == NULL) {
                fprintf (stderr, "mysql_init() failed (probably out of memory)\n");
                exit(EXIT_FAILURE);
        }
       if (mysql_real_connect(conn, conf.host, conf.db_username, conf.db_password,
    conf.database, conf.port, NULL, CLIENT_MULTI_STATEMENTS |
    CLIENT_MULTI_RESULTS) == NULL) {
                fprintf (stderr, "mysql_real_connect() failed\n");
```

```
            mysql_close (conn);
            exit(EXIT_FAILURE);
        }

    MYSQL_STMT *dbAddC;
    MYSQL_BIND param[5]; // Used both for input and output

    if(!setup_prepared_stmt(&dbAddC, "call AddCliente(?,?,?,?,?)", conn)) {
            print_stmt_error(dbAddC, "Unable to initialize AddCliente\n");
            goto err2;
    }
    printf("%d\n",n.Tavolo);
    printf("%04d-%02d-%02d                                %02d:%02d:%02d
    \n",(n.Turno).year,(n.Turno).month,(n.Turno).day,(n.Turno).hour,          (n.Turno).minute,
    (n.Turno).second);

    memset(param, 0, sizeof(param));
    param[0].buffer_type = MYSQL_TYPE_VAR_STRING; // IN
    param[0].buffer = n.nome;
    param[0].buffer_length = strlen(n.nome);

    param[1].buffer_type = MYSQL_TYPE_VAR_STRING; // IN
    param[1].buffer = n.cognome;
    param[1].buffer_length = strlen(n.cognome);

    param[2].buffer_type = MYSQL_TYPE_LONG; // IN
    param[2].buffer = &n.N_Persone;
    param[2].buffer_length = sizeof(n.N_Persone);

    param[3].buffer_type = MYSQL_TYPE_LONG; // IN
    param[3].buffer = &(n.Tavolo);
    param[3].buffer_length = sizeof(n.Tavolo);

    param[4].buffer_type = MYSQL_TYPE_DATETIME; // IN
    param[4].buffer = (char *)&n.Turno;

    if (mysql_stmt_bind_param(dbAddC, param) != 0) { // Note _param
            print_stmt_error(dbAddC, "Could not bind parameters for login");
            goto err;
    }

    // Run procedure
    if (mysql_stmt_execute(dbAddC) != 0) {
            print_stmt_error(dbAddC, "Could not view Tavoli Liberi");
            goto err;
    }

    mysql_stmt_close(dbAddC);
    mysql_close(conn);
    return;
```

```
    err:
    mysql_stmt_close(dbAddC);
    err2:
    mysql_close(conn);
    return;
}
void dbListaTavoli(Tavolo ** Testa,int n)
{

    if(!parse_config("users/Manager.json", &conf)) {
            fprintf(stderr, "Unable to load login configuration\n");
            exit(EXIT_FAILURE);
        }
    conn = mysql_init (NULL);
        if (conn == NULL) {
            fprintf (stderr, "mysql_init() failed (probably out of memory)\n");
            exit(EXIT_FAILURE);
        }
       if (mysql_real_connect(conn, conf.host, conf.db_username, conf.db_password,
    conf.database, conf.port, NULL, CLIENT_MULTI_STATEMENTS |
    CLIENT_MULTI_RESULTS) == NULL) {
            fprintf (stderr, "mysql_real_connect() failed\n");
            mysql_close (conn);
            exit(EXIT_FAILURE);
        }

    MYSQL_STMT *TavoliLiberi_procedure;
    MYSQL_BIND param; // Used both for input and output

    if(!setup_prepared_stmt(&TavoliLiberi_procedure, "call TavoliLiberi(?)", conn)) {
        print_stmt_error(TavoliLiberi_procedure, "Unable to initialize login statement\n");
        goto err2;
    }
    memset(&param, 0, sizeof(param));
    param.buffer_type = MYSQL_TYPE_LONG; // IN
    param.buffer = &n;
    param.buffer_length = sizeof(n);

    if (mysql_stmt_bind_param(TavoliLiberi_procedure, &param) != 0) { // Note _param
        print_stmt_error(TavoliLiberi_procedure, "Could not bind parameters for login");
        goto err;
    }

    // Run procedure
    if (mysql_stmt_execute(TavoliLiberi_procedure) != 0) {
        print_stmt_error(TavoliLiberi_procedure, "Could not view Tavoli Liberi");
        goto err;
    }

    MYSQL_BIND param1[2];
    MYSQL_TIME    ts;
```

```
	int i;

	memset(param1, 0, sizeof(param1));
	param1[0].buffer_type= MYSQL_TYPE_LONG;
	param1[0].buffer= &i;
	param1[0].buffer_length = sizeof(i);


	param1[1].buffer_type= MYSQL_TYPE_TIMESTAMP;
	param1[1].buffer= (char *)&ts;
	if(mysql_stmt_bind_result(TavoliLiberi_procedure,param1)!=0)
	{
		printf("empty result\n");
		goto err2;
	}
	if(mysql_stmt_store_result(TavoliLiberi_procedure)!=0)
	{
		printf("empty result\n");
		goto err2;
	}

	while (!mysql_stmt_fetch(TavoliLiberi_procedure))
	{
		printf("cioa\n");
		Tavolo * app= malloc(sizeof(Tavolo));
		if(app==NULL){
			printf("error Malloc\n");
		}
		app->next=NULL;
		if(*Testa!=NULL)
		{
			Tavolo * app2=*Testa;
			while(app2->next!=NULL)
			app2=app2->next;
			app2->next=app;
		}
		else *Testa=app;

		app->N_Tavolo=i;
		memcpy(&(app->turno),&ts,sizeof(MYSQL_TIME));
	}

	mysql_stmt_close(TavoliLiberi_procedure);
	return;

	  err:
	mysql_stmt_close(TavoliLiberi_procedure);
	  err2:
	Testa=NULL;
	return;
}
```

```
int* dbViewTavoli(cameriere c,int * d){
	if(!parse_config("users/Cameriere.json", &conf)) {
			fprintf(stderr, "Unable to load login configuration\n");
			exit(EXIT_FAILURE);
		}
	conn = mysql_init (NULL);
		if (conn == NULL) {
			fprintf (stderr, "mysql_init() failed (probably out of memory)\n");
			exit(EXIT_FAILURE);
		}
		if (mysql_real_connect(conn, conf.host, conf.db_username, conf.db_password, conf.database, conf.port, NULL, CLIENT_MULTI_STATEMENTS | CLIENT_MULTI_RESULTS) == NULL) {
			fprintf (stderr, "mysql_real_connect() failed\n");
			mysql_close (conn);
			exit(EXIT_FAILURE);
		}

	MYSQL_STMT *TavoliAssegnatiCameriere;
	MYSQL_BIND param[2]; // Used both for input and output

	if(!setup_prepared_stmt(&TavoliAssegnatiCameriere, "call TavoliAssegnatiCameriere(?,?)", conn)) {
		print_stmt_error(TavoliAssegnatiCameriere, "Unable to initialize login statement\n");
		goto err2;
	}
	memset(param, 0, sizeof(param));
	param[0].buffer_type = MYSQL_TYPE_VAR_STRING; // IN
	param[0].buffer = &c.nome;
	param[0].buffer_length = strlen(c.nome);

	param[1].buffer_type = MYSQL_TYPE_VAR_STRING; // IN
	param[1].buffer = &c.cognome;
	param[1].buffer_length = strlen(c.cognome);
	if (mysql_stmt_bind_param(TavoliAssegnatiCameriere, param) != 0) { // Note _param
		print_stmt_error(TavoliAssegnatiCameriere, "Could not bind parameters for login");
		goto err;
	}

	// Run procedure
	if (mysql_stmt_execute(TavoliAssegnatiCameriere) != 0) {
		print_stmt_error(TavoliAssegnatiCameriere, "Could not view Tavoli Liberi");
		goto err;
	}

	MYSQL_BIND res;
	int i;

	memset(&res, 0, sizeof(res));
	res.buffer_type= MYSQL_TYPE_LONG;
```

```
	res.buffer= &i;
	res.buffer_length = sizeof(i);

	if(mysql_stmt_bind_result(TavoliAssegnatiCameriere,&res)!=0)
	{
		printf("empty result\n");
		goto err2;
	}

	if(mysql_stmt_store_result(TavoliAssegnatiCameriere)!=0)
	{
		printf("empty result\n");
		goto err2;
	}
	int *Vet=NULL;
	int dim=mysql_stmt_num_rows(TavoliAssegnatiCameriere);
	*d=dim;
	if(dim>0)
	{
		Vet=malloc(sizeof(int)*dim);
		if(Vet==NULL) goto err;
		int j=0;
		while (!mysql_stmt_fetch(TavoliAssegnatiCameriere))
		{
			Vet[j]=i;
			j++;
		}
	}
	mysql_stmt_close(TavoliAssegnatiCameriere);
	mysql_close(conn);
	return Vet;

	  err:
	mysql_stmt_close(TavoliAssegnatiCameriere);
	  err2:
	mysql_close(conn);
	return NULL;
}
int dbClienteTavolo(int t){
	if(!parse_config("users/Cameriere.json", &conf)) {
			fprintf(stderr, "Unable to load login configuration\n");
			exit(EXIT_FAILURE);
		}
	conn = mysql_init (NULL);
		if (conn == NULL) {
			fprintf (stderr, "mysql_init() failed (probably out of memory)\n");
			exit(EXIT_FAILURE);
		}
		if (mysql_real_connect(conn, conf.host, conf.db_username, conf.db_password,
	conf.database, conf.port, NULL, CLIENT_MULTI_STATEMENTS |
	CLIENT_MULTI_RESULTS) == NULL) {
```

```
            fprintf (stderr, "mysql_real_connect() failed\n");
            mysql_close (conn);
            exit(EXIT_FAILURE);
        }

MYSQL_STMT *Cliente_Tavolo;
MYSQL_BIND param; // Used both for input and output

if(!setup_prepared_stmt(&Cliente_Tavolo, "call Cliente_Tavolo(?)", conn)) {
        print_stmt_error(Cliente_Tavolo, "Unable to initialize login statement\n");
        goto err2;
}
memset(&param, 0, sizeof(param));
param.buffer_type = MYSQL_TYPE_LONG; // IN
param.buffer = &t;
param.buffer_length = sizeof(t);

if (mysql_stmt_bind_param(Cliente_Tavolo,&param) != 0) { // Note _param
        print_stmt_error(Cliente_Tavolo, "Could not bind parameters for login");
        goto err;
}

// Run procedure
if (mysql_stmt_execute(Cliente_Tavolo) != 0) {
        print_stmt_error(Cliente_Tavolo, "Could not view Tavoli Liberi");
        goto err;
}

MYSQL_BIND res;
int i;

memset(&res, 0, sizeof(res));
res.buffer_type= MYSQL_TYPE_LONG;
res.buffer= &i;
res.buffer_length = sizeof(i);

if(mysql_stmt_bind_result(Cliente_Tavolo,&res)!=0)
{
        printf("empty result\n");
        goto err2;
}
if(mysql_stmt_store_result(Cliente_Tavolo)!=0)
{
        printf("empty result\n");
        goto err2;
}
if(mysql_stmt_fetch(Cliente_Tavolo)){
        return 0;
}
mysql_stmt_close(Cliente_Tavolo);
mysql_close(conn);
```

```
        return i;

           err:
        mysql_stmt_close(Cliente_Tavolo);
           err2:
        mysql_close(conn);
        return 0;
}
void dbIngredienti(ingrediente **Testa){

        if(!parse_config("users/Manager.json", &conf)) {
                fprintf(stderr, "Unable to load login configuration\n");
                exit(EXIT_FAILURE);
            }
        conn = mysql_init (NULL);
            if (conn == NULL) {
                fprintf (stderr, "mysql_init() failed (probably out of memory)\n");
                exit(EXIT_FAILURE);
            }
           if (mysql_real_connect(conn, conf.host, conf.db_username, conf.db_password,
        conf.database, conf.port, NULL, CLIENT_MULTI_STATEMENTS |
        CLIENT_MULTI_RESULTS) == NULL) {
                fprintf (stderr, "mysql_real_connect() failed\n");
                mysql_close (conn);
                exit(EXIT_FAILURE);
            }

        MYSQL_STMT *dbListaIngredienti;
         // Used both for input and output

        if(!setup_prepared_stmt(&dbListaIngredienti, "call dbListaIngredienti()", conn)) {
                print_stmt_error(dbListaIngredienti, "Unable to initialize login statement\n");
                goto err2;
        }

        if (mysql_stmt_bind_param(dbListaIngredienti,NULL) != 0) { // Note _param
                print_stmt_error(dbListaIngredienti, "Could not bind parameters for login");
                goto err;
        }

        // Run procedure
        if (mysql_stmt_execute(dbListaIngredienti) != 0) {
                print_stmt_error(dbListaIngredienti, "Could not view Tavoli Liberi");
                goto err;
        }

        MYSQL_BIND param1[2];
        char nome[16];
        int i;

        memset(param1, 0, sizeof(param1));
```

```
	param1[0].buffer_type = MYSQL_TYPE_VAR_STRING; // OUT
	param1[0].buffer = nome;
	param1[0].buffer_length = 15;

	param1[1].buffer_type= MYSQL_TYPE_LONG;
	param1[1].buffer= &i;
	param1[1].buffer_length = sizeof(i);

	if(mysql_stmt_bind_result(dbListaIngredienti,param1)!=0)
	{
		printf("empty result\n");
		goto err2;
	}

	if(mysql_stmt_store_result(dbListaIngredienti)!=0)
	{
		printf("empty result\n");
		goto err2;
	}

	while (!mysql_stmt_fetch(dbListaIngredienti))
	{
		ingrediente * app= malloc(sizeof(ingrediente));
		if(app==NULL){
			printf("error Malloc\n");
		}
		app->next=NULL;
		if(*Testa!=NULL)
		{
			ingrediente * app2=*Testa;
			while(app2->next!=NULL)
			app2=app2->next;
			app2->next=app;
		}
		else *Testa=app;
		app->quantita=i;
		strcpy(app->nome,nome);
	}

	mysql_stmt_close(dbListaIngredienti);
	mysql_close(conn);
	return;

	err:
	mysql_stmt_close(dbListaIngredienti);
	err2:
	Testa=NULL;
	mysql_close(conn);
	return;
}
```

```
void dbAggiunte(ingrediente **Testa)
{
	if(!parse_config("users/Cameriere.json", &conf)) {
			fprintf(stderr, "Unable to load login configuration\n");
			exit(EXIT_FAILURE);
		}
	conn = mysql_init (NULL);
		if (conn == NULL) {
			fprintf (stderr, "mysql_init() failed (probably out of memory)\n");
			exit(EXIT_FAILURE);
		}
		if (mysql_real_connect(conn, conf.host, conf.db_username, conf.db_password,
	conf.database, conf.port, NULL, CLIENT_MULTI_STATEMENTS |
	CLIENT_MULTI_RESULTS) == NULL) {
			fprintf (stderr, "mysql_real_connect() failed\n");
			mysql_close (conn);
			exit(EXIT_FAILURE);
		}

	MYSQL_STMT *ListaAggiunte;
	 // Used both for input and output

	if(!setup_prepared_stmt(&ListaAggiunte, "call ListaAggiunte()", conn)) {
		print_stmt_error(ListaAggiunte, "Unable to initialize login statement\n");
		goto err2;
	}

	if (mysql_stmt_bind_param(ListaAggiunte,NULL) != 0) { // Note _param
		print_stmt_error(ListaAggiunte, "Could not bind parameters for login");
		goto err;
	}

	// Run procedure
	if (mysql_stmt_execute(ListaAggiunte) != 0) {
		print_stmt_error(ListaAggiunte, "Could not view Tavoli Liberi");
		goto err;
	}

	MYSQL_BIND param1[2];
	char nome[16];
	int i;

	memset(param1, 0, sizeof(param1));

	param1[0].buffer_type = MYSQL_TYPE_VAR_STRING; // OUT
	param1[0].buffer = nome;
	param1[0].buffer_length = 15;

	param1[1].buffer_type= MYSQL_TYPE_LONG;
	param1[1].buffer= &i;
	param1[1].buffer_length = sizeof(i);
```

```
	if(mysql_stmt_bind_result(ListaAggiunte,param1)!=0)
	{
		printf("empty result\n");
		goto err2;
	}

	if(mysql_stmt_store_result(ListaAggiunte)!=0)
	{
		printf("empty result\n");
		goto err2;
	}

	while (!mysql_stmt_fetch(ListaAggiunte))
	{
		ingrediente * app= malloc(sizeof(ingrediente));
		if(app==NULL){
			printf("error Malloc\n");
		}
		app->next=NULL;
		if(*Testa!=NULL)
		{
			ingrediente * app2=*Testa;
			while(app2->next!=NULL)
			app2=app2->next;
			app2->next=app;
		}
		else *Testa=app;

		app->costo=i;
		strcpy(app->nome,nome);
	}

	mysql_stmt_close(ListaAggiunte);
	mysql_close(conn);
	return;

	   err:
	mysql_stmt_close(ListaAggiunte);
	   err2:
	Testa=NULL;
	mysql_close(conn);
	return;
}
void dbListaMenu(prodotto ** Testa){
	if(!parse_config("users/Cameriere.json", &conf)) {
			fprintf(stderr, "Unable to load login configuration\n");
			exit(EXIT_FAILURE);
		}
	conn = mysql_init (NULL);
		if (conn == NULL) {
```

```
            fprintf (stderr, "mysql_init() failed (probably out of memory)\n");
            exit(EXIT_FAILURE);
        }
    if (mysql_real_connect(conn, conf.host, conf.db_username, conf.db_password,
conf.database, conf.port, NULL, CLIENT_MULTI_STATEMENTS |
CLIENT_MULTI_RESULTS) == NULL) {
            fprintf (stderr, "mysql_real_connect() failed\n");
            mysql_close (conn);
            exit(EXIT_FAILURE);
        }

MYSQL_STMT *ListaAggiunte;
 // Used both for input and output

if(!setup_prepared_stmt(&ListaAggiunte, "call menu()", conn)) {
        print_stmt_error(ListaAggiunte, "Unable to initialize login statement\n");
        goto err2;
}

if (mysql_stmt_bind_param(ListaAggiunte,NULL) != 0) { // Note _param
        print_stmt_error(ListaAggiunte, "Could not bind parameters for login");
        goto err;
}

// Run procedure
if (mysql_stmt_execute(ListaAggiunte) != 0) {
        print_stmt_error(ListaAggiunte, "Could not view Tavoli Liberi");
        goto err;
}

MYSQL_BIND param1[3];
char nome[16];
int i;
int tipo;

memset(param1, 0, sizeof(param1));

param1[0].buffer_type = MYSQL_TYPE_VAR_STRING; // OUT
param1[0].buffer = nome;
param1[0].buffer_length = 15;

param1[1].buffer_type= MYSQL_TYPE_LONG;
param1[1].buffer= &i;
param1[1].buffer_length = sizeof(i);

param1[2].buffer_type= MYSQL_TYPE_LONG;
param1[2].buffer= &tipo;
param1[2].buffer_length = sizeof(tipo);

if(mysql_stmt_bind_result(ListaAggiunte,param1)!=0)
{
```

```
		printf("empty result\n");
		goto err2;
	}

	if(mysql_stmt_store_result(ListaAggiunte)!=0)
	{
		printf("empty result\n");
		goto err2;
	}

	while (!mysql_stmt_fetch(ListaAggiunte))
	{
		prodotto * app= malloc(sizeof(prodotto));
		if(app==NULL){
			printf("error Malloc\n");
		}
		app->next=NULL;
		if(*Testa!=NULL)
		{
			prodotto * app2=*Testa;
			while(app2->next!=NULL)
			app2=app2->next;
			app2->next=app;
		}
		else *Testa=app;
		app->tipo=tipo;
		app->prezzo=i;
		strcpy(app->nome,nome);
	}

	mysql_stmt_close(ListaAggiunte);
	mysql_close(conn);
	return;

	  err:
	mysql_stmt_close(ListaAggiunte);
	  err2:
	Testa=NULL;
	mysql_close(conn);
	return;
}
void dbAddNewOrdine(int c,char * pizze,char * bevande){
	if(!parse_config("users/Cameriere.json", &conf)) {
			fprintf(stderr, "Unable to load login configuration\n");
			exit(EXIT_FAILURE);
		}
	conn = mysql_init (NULL);
		if (conn == NULL) {
			fprintf (stderr, "mysql_init() failed (probably out of memory)\n");
			exit(EXIT_FAILURE);
		}
```

```
    if (mysql_real_connect(conn, conf.host, conf.db_username, conf.db_password,
conf.database, conf.port, NULL, CLIENT_MULTI_STATEMENTS |
CLIENT_MULTI_RESULTS) == NULL) {
            fprintf (stderr, "mysql_real_connect() failed\n");
            mysql_close (conn);
            exit(EXIT_FAILURE);
      }

MYSQL_STMT *Ordine;
 // Used both for input and output

if(!setup_prepared_stmt(&Ordine, "call Ordine(?,?,?)", conn)) {
        print_stmt_error(Ordine, "Unable to initialize login statement\n");
        goto err2;
}
MYSQL_BIND param[3];
// Prepare parameters
memset(param, 0, sizeof(param));

param[0].buffer_type = MYSQL_TYPE_LONG; // IN
param[0].buffer = &c;
param[0].buffer_length = sizeof(c);

param[1].buffer_type = MYSQL_TYPE_VAR_STRING; // IN
param[1].buffer = pizze;
param[1].buffer_length = strlen(pizze);

param[2].buffer_type = MYSQL_TYPE_VAR_STRING; // IN
param[2].buffer = bevande;
param[2].buffer_length = strlen(bevande);

if (mysql_stmt_bind_param(Ordine,param) != 0) { // Note _param
        print_stmt_error(Ordine, "Could not bind parameters for login");
        goto err;
}

// Run procedure
if (mysql_stmt_execute(Ordine) != 0) {
        print_stmt_error(Ordine, "Could not view Tavoli Liberi");
        goto err;
}

mysql_stmt_close(Ordine);
mysql_close(conn);
return;

   err:
mysql_stmt_close(Ordine);
   err2:
mysql_close(conn);
return;
```

```
}
void dbOrdiniDaCucinare(prodotto ** Testa){

	if(!parse_config("users/Pizzaiolo.json", &conf)) {
		fprintf(stderr, "Unable to load login configuration\n");
		exit(EXIT_FAILURE);
	}
	conn = mysql_init (NULL);
	if (conn == NULL) {
		fprintf (stderr, "mysql_init() failed (probably out of memory)\n");
		exit(EXIT_FAILURE);
	}
	if (mysql_real_connect(conn, conf.host, conf.db_username, conf.db_password, conf.database, conf.port, NULL, CLIENT_MULTI_STATEMENTS | CLIENT_MULTI_RESULTS) == NULL) {
		fprintf (stderr, "mysql_real_connect() failed\n");
		mysql_close (conn);
		exit(EXIT_FAILURE);
	}

	MYSQL_STMT *OrdiniDaPrepararePizzaiolo;

	if(!setup_prepared_stmt(&OrdiniDaPrepararePizzaiolo, "call OrdiniDaPrepararePizzaiolo()", conn)) {
		print_stmt_error(OrdiniDaPrepararePizzaiolo, "Unable to initialize OrdiniDaPrepararePizzaiolo\n");
		goto err2;
	}


	if (mysql_stmt_bind_param(OrdiniDaPrepararePizzaiolo, NULL) != 0) { // Note _param
		print_stmt_error(OrdiniDaPrepararePizzaiolo, "Could not bind parameters for login");
		goto err;
	}
	// Run procedure
	if (mysql_stmt_execute(OrdiniDaPrepararePizzaiolo) != 0) {
		print_stmt_error(OrdiniDaPrepararePizzaiolo, "Could not view Tavoli Liberi");
		goto err;
	}
	MYSQL_BIND param[5]; // Used both for input and output
	int id;
	char nome[16];
	char aggiunta[16];
	int quantita;
	int porzione;

	memset(param, 0, sizeof(param));
	param[0].buffer_type = MYSQL_TYPE_LONG; // IN
	param[0].buffer = &id;
	param[0].buffer_length = sizeof(id);
```

```
param[1].buffer_type = MYSQL_TYPE_VAR_STRING; // IN
param[1].buffer = nome;
param[1].buffer_length = 16;

param[2].buffer_type = MYSQL_TYPE_LONG; // IN
param[2].buffer = &quantita;
param[2].buffer_length = sizeof(quantita);

bool controlloAggiunta;
param[3].buffer_type = MYSQL_TYPE_VAR_STRING; // IN
param[3].buffer = aggiunta;
param[3].buffer_length = 16;
param[3].is_null=&controlloAggiunta;

param[4].buffer_type = MYSQL_TYPE_LONG; // IN
param[4].buffer = &porzione;
param[4].buffer_length = sizeof(porzione);

if(mysql_stmt_bind_result(OrdiniDaPrepararePizzaiolo,param)!=0)
{
        printf("empty result\n");
        goto err2;
}
if(mysql_stmt_store_result(OrdiniDaPrepararePizzaiolo)!=0)
{
        printf("empty result\n");
        goto err2;
}
int j;
prodotto * app=*Testa;
while (!mysql_stmt_fetch(OrdiniDaPrepararePizzaiolo))
{
        if(app==NULL){
                app=malloc(sizeof(prodotto));
                app->next=NULL;
                strcpy(app->nome,nome);
                app->tipo=quantita;
                app->prezzo=id;
                j=id;
                app->listaIngredienti=NULL;
                if(!controlloAggiunta){
                        app->listaIngredienti=malloc(sizeof(ingrediente));
                        app->listaIngredienti->next=NULL;
                        strcpy(app->listaIngredienti->nome,aggiunta);
                        app->listaIngredienti->quantita=porzione;
                }
                *Testa=app;
                app=*Testa;
        }
        else{
```

```
                        if(j==id){
                                ingrediente *app2=app->listaIngredienti;
                                while (app2->next!=NULL) app2=app2->next;
                                app2->next=malloc(sizeof(ingrediente));
                                app2=app2->next;
                                app2->next=NULL;
                                strcpy(app2->nome,aggiunta);
                                app2->quantita=porzione;
                        }
                        else{
                                app->next=malloc(sizeof(prodotto));
                                app=app->next;
                                app->next=NULL;
                                strcpy(app->nome,nome);
                                app->tipo=quantita;
                                app->prezzo=id;
                                j=id;
                                app->listaIngredienti=NULL;
                                if(!controlloAggiunta){
                                        app->listaIngredienti=malloc(sizeof(ingrediente));
                                        app->listaIngredienti->next=NULL;
                                        strcpy(app->listaIngredienti->nome,aggiunta);
                                        app->listaIngredienti->quantita=porzione;
                                }
                        }
                }
        }

        mysql_stmt_close(OrdiniDaPrepararePizzaiolo);
        mysql_close(conn);
        return;

           err:
        mysql_stmt_close(OrdiniDaPrepararePizzaiolo);
           err2:
        mysql_close(conn);
        return;
}
void dbPizzaFinita(int i){

        if(!parse_config("users/Pizzaiolo.json", &conf)) {
                fprintf(stderr, "Unable to load login configuration\n");
                exit(EXIT_FAILURE);
            }
        conn = mysql_init (NULL);
            if (conn == NULL) {
                fprintf (stderr, "mysql_init() failed (probably out of memory)\n");
                exit(EXIT_FAILURE);
            }
```

```
        if (mysql_real_connect(conn, conf.host, conf.db_username, conf.db_password,
    conf.database, conf.port, NULL, CLIENT_MULTI_STATEMENTS |
    CLIENT_MULTI_RESULTS) == NULL) {
            fprintf (stderr, "mysql_real_connect() failed\n");
            mysql_close (conn);
            exit(EXIT_FAILURE);
        }

    MYSQL_STMT *pizzaPreparata;

    if(!setup_prepared_stmt(&pizzaPreparata, "call pizzaPreparata(?)", conn)) {
        print_stmt_error(pizzaPreparata, "Unable to initialize pizzaPreparata\n");
        goto err2;
    }

    MYSQL_BIND param; // Used both for input and output
    memset(&param, 0, sizeof(param));
    param.buffer_type = MYSQL_TYPE_LONG; // IN
    param.buffer = &i;
    param.buffer_length = sizeof(i);


    if (mysql_stmt_bind_param(pizzaPreparata, &param) != 0) { // Note _param
        print_stmt_error(pizzaPreparata, "Could not bind parameters for login");
        goto err;
    }
    // Run procedure
    if (mysql_stmt_execute(pizzaPreparata) != 0) {
        print_stmt_error(pizzaPreparata, "Could not view Tavoli Liberi");
        goto err;
    }


    mysql_stmt_close(pizzaPreparata);
    mysql_close(conn);
    return;

      err:
    mysql_stmt_close(pizzaPreparata);
      err2:
    mysql_close(conn);
    return;
}
void dbBevandeDaCucinare(prodotto ** Testa){

    if(!parse_config("users/Barista.json", &conf)) {
        fprintf(stderr, "Unable to load login configuration\n");
        exit(EXIT_FAILURE);
    }
    conn = mysql_init (NULL);
      if (conn == NULL) {
```

```
            fprintf (stderr, "mysql_init() failed (probably out of memory)\n");
            exit(EXIT_FAILURE);
        }
    if (mysql_real_connect(conn, conf.host, conf.db_username, conf.db_password,
conf.database, conf.port, NULL, CLIENT_MULTI_STATEMENTS |
CLIENT_MULTI_RESULTS) == NULL) {
            fprintf (stderr, "mysql_real_connect() failed\n");
            mysql_close (conn);
            exit(EXIT_FAILURE);
        }

MYSQL_STMT *BaristaOrdini;

if(!setup_prepared_stmt(&BaristaOrdini, "call BaristaOrdini()", conn)) {
        print_stmt_error(BaristaOrdini, "Unable to initialize BaristaOrdini\n");
        goto err2;
}


if (mysql_stmt_bind_param(BaristaOrdini, NULL) != 0) { // Note _param
        print_stmt_error(BaristaOrdini, "Could not bind parameters for login");
        goto err;
}
// Run procedure
if (mysql_stmt_execute(BaristaOrdini) != 0) {
        print_stmt_error(BaristaOrdini, "Could not view Tavoli Liberi");
        goto err;
}
MYSQL_BIND param[3]; // Used both for input and output
int id;
char nome[16];
int quantita;

memset(param, 0, sizeof(param));
param[0].buffer_type = MYSQL_TYPE_LONG; // IN
param[0].buffer = &id;
param[0].buffer_length = sizeof(id);

param[1].buffer_type = MYSQL_TYPE_VAR_STRING; // IN
param[1].buffer = nome;
param[1].buffer_length = 16;

param[2].buffer_type = MYSQL_TYPE_LONG; // IN
param[2].buffer = &quantita;
param[2].buffer_length = sizeof(quantita);

if(mysql_stmt_bind_result(BaristaOrdini,param)!=0)
{
        printf("empty result\n");
        goto err2;
}
```

```
if(mysql_stmt_store_result(BaristaOrdini)!=0)
{
        printf("empty result\n");
        goto err2;
}
prodotto * app=*Testa;
while (!mysql_stmt_fetch(BaristaOrdini))
{
        if(app==NULL){
                app=malloc(sizeof(prodotto));
                app->next=NULL;
                strcpy(app->nome,nome);
                app->tipo=quantita;
                app->prezzo=id;
                app->listaIngredienti=NULL;
                *Testa=app;
                app=*Testa;
        }
        else{
                app->next=malloc(sizeof(prodotto));
                app=app->next;
                app->next=NULL;
                app->listaIngredienti=NULL;
                strcpy(app->nome,nome);
                app->tipo=quantita;
                app->prezzo=id;
        }
}
mysql_stmt_close(BaristaOrdini);
mysql_close(conn);
return;

   err:
mysql_stmt_close(BaristaOrdini);
   err2:
mysql_close(conn);
return;
}
void dbBevandaPronta(int i){

if(!parse_config("users/Barista.json", &conf)) {
            fprintf(stderr, "Unable to load login configuration\n");
            exit(EXIT_FAILURE);
      }
conn = mysql_init (NULL);
      if (conn == NULL) {
            fprintf (stderr, "mysql_init() failed (probably out of memory)\n");
            exit(EXIT_FAILURE);
      }
```

```
        if (mysql_real_connect(conn, conf.host, conf.db_username, conf.db_password,
    conf.database, conf.port, NULL, CLIENT_MULTI_STATEMENTS |
    CLIENT_MULTI_RESULTS) == NULL) {
                fprintf (stderr, "mysql_real_connect() failed\n");
                mysql_close (conn);
                exit(EXIT_FAILURE);
        }

    MYSQL_STMT *BevandePronte;

    if(!setup_prepared_stmt(&BevandePronte, "call BevandePronte(?)", conn)) {
            print_stmt_error(BevandePronte, "Unable to initialize pizzaPreparata\n");
            goto err2;
    }

    MYSQL_BIND param; // Used both for input and output
    memset(&param, 0, sizeof(param));
    param.buffer_type = MYSQL_TYPE_LONG; // IN
    param.buffer = &i;
    param.buffer_length = sizeof(i);


    if (mysql_stmt_bind_param(BevandePronte, &param) != 0) { // Note _param
            print_stmt_error(BevandePronte, "Could not bind parameters for login");
            goto err;
    }
    // Run procedure
    if (mysql_stmt_execute(BevandePronte) != 0) {
                    print_stmt_error(BevandePronte, "Could not view Tavoli Liberi");
                    goto err;
    }


    mysql_stmt_close(BevandePronte);
    mysql_close(conn);
    return;

       err:
    mysql_stmt_close(BevandePronte);
       err2:
    mysql_close(conn);
    return;
}
void dbOrdiniDaConsegnare(ordini ** list,cameriere c){
    if(!parse_config("users/Cameriere.json", &conf)) {
                fprintf(stderr, "Unable to load login configuration\n");
                exit(EXIT_FAILURE);
        }
    conn = mysql_init (NULL);
          if (conn == NULL) {
                fprintf (stderr, "mysql_init() failed (probably out of memory)\n");
```

```
            exit(EXIT_FAILURE);
        }
    if (mysql_real_connect(conn, conf.host, conf.db_username, conf.db_password, conf.database, conf.port, NULL, CLIENT_MULTI_STATEMENTS | CLIENT_MULTI_RESULTS) == NULL) {
            fprintf (stderr, "mysql_real_connect() failed\n");
            mysql_close (conn);
            exit(EXIT_FAILURE);
        }

MYSQL_STMT *ordiniDaPortare;
MYSQL_BIND param[2]; // Used both for input and output

if(!setup_prepared_stmt(&ordiniDaPortare, "call ordiniDaPortare(?,?)", conn)) {
        print_stmt_error(ordiniDaPortare, "Unable to initialize login statement\n");
        goto err2;
}
memset(param, 0, sizeof(param));
param[0].buffer_type = MYSQL_TYPE_VAR_STRING; // IN
param[0].buffer = &c.nome;
param[0].buffer_length = strlen(c.nome);

param[1].buffer_type = MYSQL_TYPE_VAR_STRING; // IN
param[1].buffer = &c.cognome;
param[1].buffer_length = strlen(c.cognome);
if (mysql_stmt_bind_param(ordiniDaPortare, param) != 0) { // Note _param
        print_stmt_error(ordiniDaPortare, "Could not bind parameters for login");
        goto err;
}

// Run procedure
if (mysql_stmt_execute(ordiniDaPortare) != 0) {
        print_stmt_error(ordiniDaPortare, "Could not view Tavoli Liberi");
        goto err;
}

MYSQL_BIND res[3];
int cl;
MYSQL_TIME tid;
int tavolo;

memset(res, 0, sizeof(res));
res[0].buffer_type= MYSQL_TYPE_LONG;
res[0].buffer= &tavolo;
res[0].buffer_length = sizeof(tavolo);

res[1].buffer_type= MYSQL_TYPE_LONG;
res[1].buffer= &cl;
res[1].buffer_length = sizeof(cl);

res[2].buffer_type= MYSQL_TYPE_TIMESTAMP;
```

```
	res[2].buffer= (char*)&tid;

	if(mysql_stmt_bind_result(ordiniDaPortare,res)!=0)
	{
		printf("empty result\n");
		goto err2;
	}

	if(mysql_stmt_store_result(ordiniDaPortare)!=0)
	{
		printf("empty result\n");
		goto err2;
	}
	while (!mysql_stmt_fetch(ordiniDaPortare))
	{
		ordini* app=malloc(sizeof(ordini));
		app->next=NULL;
		app->cliente=cl;
		app->tavolo=tavolo;
		memcpy(&(app->t),&tid,sizeof(MYSQL_TIME));
		if(*list==NULL)
		{
			*list=app;
		}
		else{
			ordini* app2=*list;
			while(app2->next!=NULL) app2=app2->next;
			app2->next=app;
		}
	}
	mysql_stmt_close(ordiniDaPortare);
	mysql_close(conn);
	return ;

	  err:
	mysql_stmt_close(ordiniDaPortare);
	  err2:
	mysql_close(conn);
	return;
}
void dbUpdateComanda(ordini p){
	if(!parse_config("users/Cameriere.json", &conf)) {
			fprintf(stderr, "Unable to load login configuration\n");
			exit(EXIT_FAILURE);
	  }
	conn = mysql_init (NULL);
	  if (conn == NULL) {
		fprintf (stderr, "mysql_init() failed (probably out of memory)\n");
		exit(EXIT_FAILURE);
	  }
```

```
	if (mysql_real_connect(conn, conf.host, conf.db_username, conf.db_password, conf.database, conf.port, NULL, CLIENT_MULTI_STATEMENTS | CLIENT_MULTI_RESULTS) == NULL) {
		fprintf (stderr, "mysql_real_connect() failed\n");
		mysql_close (conn);
		exit(EXIT_FAILURE);
	}
	MYSQL_STMT *UpdateComandeConsegnate;
	MYSQL_BIND param[2]; // Used both for input and output
	if(!setup_prepared_stmt(&UpdateComandeConsegnate, "call UpdateComandeConsegnate(?,?)", conn)) {
		print_stmt_error(UpdateComandeConsegnate, "Unable to initialize login statement\n");
		goto err2;
	}
	memset(param, 0, sizeof(param));
	param[0].buffer_type = MYSQL_TYPE_LONG; // IN
	param[0].buffer = &p.cliente;
	param[0].buffer_length = sizeof(p.cliente);

	param[1].buffer_type = MYSQL_TYPE_DATETIME; // IN
	param[1].buffer = (char*)&p.t;

	if (mysql_stmt_bind_param(UpdateComandeConsegnate, param) != 0) { // Note _param
		print_stmt_error(UpdateComandeConsegnate, "Could not bind parameters for login");
		goto err;
	}

	// Run procedure
	if (mysql_stmt_execute(UpdateComandeConsegnate) != 0) {
		print_stmt_error(UpdateComandeConsegnate, "Could not view Tavoli Liberi");
		goto err;
	}

	mysql_stmt_close(UpdateComandeConsegnate);
	mysql_close(conn);
	return ;

	  err:
	mysql_stmt_close(UpdateComandeConsegnate);
	  err2:
	mysql_close(conn);
	return;
}
void dbStampaScontrino(prodotto **Lista,int *SpesaToT,int c){
	if(!parse_config("users/Manager.json", &conf)) {
		fprintf(stderr, "Unable to load login configuration\n");
		exit(EXIT_FAILURE);
	}
	conn = mysql_init (NULL);
```

```
        if (conn == NULL) {
            fprintf (stderr, "mysql_init() failed (probably out of memory)\n");
            exit(EXIT_FAILURE);
        }
      if (mysql_real_connect(conn, conf.host, conf.db_username, conf.db_password,
conf.database, conf.port, NULL, CLIENT_MULTI_STATEMENTS |
CLIENT_MULTI_RESULTS) == NULL) {
            fprintf (stderr, "mysql_real_connect() failed\n");
            mysql_close (conn);
            exit(EXIT_FAILURE);
        }

MYSQL_STMT *StampaScontrino;
MYSQL_BIND param[2]; // Used both for input and output

if(!setup_prepared_stmt(&StampaScontrino, "call StampaScontrino(?,?)", conn)) {
        print_stmt_error(StampaScontrino, "Unable to initialize login statement\n");
        goto err2;
}
memset(param, 0, sizeof(param));
param[0].buffer_type = MYSQL_TYPE_LONG; // IN
param[0].buffer = &c;
param[0].buffer_length = sizeof(c);

param[1].buffer_type = MYSQL_TYPE_LONG; // OUT
param[1].buffer =SpesaToT;
param[1].buffer_length = sizeof(*SpesaToT);

if (mysql_stmt_bind_param(StampaScontrino, param) != 0) { // Note _param
        print_stmt_error(StampaScontrino, "Could not bind parameters for login");
        goto err;
}

// Run procedure
if (mysql_stmt_execute(StampaScontrino) != 0) {
        print_stmt_error(StampaScontrino, "Could not view Tavoli Liberi");
        goto err;
}
MYSQL_BIND res[9];

int ID;
char NomeProdotto[16];
int quantita;
int prezzo;
char aggiunta[16];
int porzione;
int costo;
int CostoTotaleProdotto;

memset(res, 0, sizeof(res));
res[0].buffer_type= MYSQL_TYPE_LONG;
```

```
res[0].buffer= &ID;
res[0].buffer_length = sizeof(ID);

res[1].buffer_type= MYSQL_TYPE_VAR_STRING;
res[1].buffer= NomeProdotto;
res[1].buffer_length = 15;

res[2].buffer_type= MYSQL_TYPE_LONG;
res[2].buffer= &quantita;
res[2].buffer_length = sizeof(quantita);

res[3].buffer_type= MYSQL_TYPE_LONG;
res[3].buffer= &prezzo;
res[3].buffer_length = sizeof(prezzo);

bool controlloAggiunta;
res[4].buffer_type= MYSQL_TYPE_VAR_STRING;
res[4].buffer= aggiunta;
res[4].buffer_length = 15;
res[4].is_null=&controlloAggiunta;

res[5].buffer_type= MYSQL_TYPE_LONG;
res[5].buffer= &porzione;
res[5].buffer_length = sizeof(porzione);

res[6].buffer_type= MYSQL_TYPE_LONG;
res[6].buffer= &costo;
res[6].buffer_length = sizeof(costo);

res[7].buffer_type= MYSQL_TYPE_LONG;
res[7].buffer= &CostoTotaleProdotto;
res[7].buffer_length = sizeof(CostoTotaleProdotto);

if(mysql_stmt_bind_result(StampaScontrino,res)!=0)
{
        printf("empty result\n");
        goto err2;
}
if(mysql_stmt_store_result(StampaScontrino)!=0)
{
        printf("empty result\n");
        goto err2;
}
int j;
prodotto * app=*Lista;
while (!mysql_stmt_fetch(StampaScontrino))
{
        if(app==NULL){
                app=malloc(sizeof(prodotto));
                app->next=NULL;
                strcpy(app->nome,NomeProdotto);
```

```
                app->quantita=quantita;
                app->prezzo=prezzo;
                app->CostoTotaleProdotto=CostoTotaleProdotto;
                j=ID;
                app->listaIngredienti=NULL;
                if(!controlloAggiunta){
                        app->listaIngredienti=malloc(sizeof(ingrediente));
                        app->listaIngredienti->next=NULL;
                        strcpy(app->listaIngredienti->nome,aggiunta);
                        app->listaIngredienti->quantita=porzione;
                        app->listaIngredienti->costo=costo;
                }
                *Lista=app;
                app=*Lista;
        }
        else{
                if(j==ID){
                        ingrediente *app2=app->listaIngredienti;
                        while (app2->next!=NULL) app2=app2->next;
                        app2->next=malloc(sizeof(ingrediente));
                        app2=app2->next;
                        app2->next=NULL;
                        strcpy(app2->nome,aggiunta);
                        app2->quantita=porzione;
                        app2->costo=costo;
                }
                else{
                        app->next=malloc(sizeof(prodotto));
                        app=app->next;
                        app->next=NULL;
                        strcpy(app->nome,NomeProdotto);
                        app->quantita=quantita;
                        app->prezzo=prezzo;
                        app->CostoTotaleProdotto=CostoTotaleProdotto;
                        app->id=ID;
                        j=ID;
                        app->listaIngredienti=NULL;
                        if(!controlloAggiunta){
                                app->listaIngredienti=malloc(sizeof(ingrediente));
                                app->listaIngredienti->next=NULL;
                                app->listaIngredienti->costo=costo;
                                strcpy(app->listaIngredienti->nome,aggiunta);
                                app->listaIngredienti->quantita=porzione;
                        }
                }
        }
}
if(mysql_stmt_next_result(StampaScontrino)>0){
        printf("Error\n");
        goto err;
}
```

```
	MYSQL_BIND res1;
	memset(&res1, 0, sizeof(res1));
	res1.buffer_type= MYSQL_TYPE_LONG;
	res1.buffer= SpesaToT;
	res1.buffer_length = sizeof(*SpesaToT);

	if(mysql_stmt_bind_result(StampaScontrino,&res1)!=0)
	{
		printf("empty result1\n");
		goto err2;
	}
	if(mysql_stmt_store_result(StampaScontrino)!=0)
	{
		printf("empty result2\n");
		goto err2;
	}

	if(mysql_stmt_fetch(StampaScontrino)){
		*SpesaToT=0;
		goto err;
	}
	mysql_stmt_close(StampaScontrino);
	mysql_close(conn);
	return ;
	   err:
	mysql_stmt_close(StampaScontrino);
	   err2:
	mysql_close(conn);
	return ;
}
int dbIncassoGiorno(int i){
	if(!parse_config("users/Manager.json", &conf)) {
			fprintf(stderr, "Unable to load login configuration\n");
			exit(EXIT_FAILURE);
		}
	conn = mysql_init (NULL);
		if (conn == NULL) {
			fprintf (stderr, "mysql_init() failed (probably out of memory)\n");
			exit(EXIT_FAILURE);
		}
		if (mysql_real_connect(conn, conf.host, conf.db_username, conf.db_password,
	conf.database, conf.port, NULL, CLIENT_MULTI_STATEMENTS |
	CLIENT_MULTI_RESULTS) == NULL) {
			fprintf (stderr, "mysql_real_connect() failed\n");
			mysql_close (conn);
			exit(EXIT_FAILURE);
		}

	MYSQL_STMT *IncassiGiornalieri;
	MYSQL_BIND param[2]; // Used both for input and output
```

```
if(!setup_prepared_stmt(&IncassiGiornalieri, "call IncassiGiornalieri(?,?)", conn)) {
        print_stmt_error(IncassiGiornalieri, "Unable to initialize login statement\n");
        goto err2;
}
int j=0;
memset(param, 0, sizeof(param));
param[0].buffer_type = MYSQL_TYPE_LONG; // IN
param[0].buffer = &i;
param[0].buffer_length = sizeof(i);

param[1].buffer_type = MYSQL_TYPE_LONG; // OUT
param[1].buffer =&j;
param[1].buffer_length = sizeof(j);

if (mysql_stmt_bind_param(IncassiGiornalieri, param) != 0) { // Note _param
        print_stmt_error(IncassiGiornalieri, "Could not bind parameters for login");
        goto err;
}

// Run procedure
if (mysql_stmt_execute(IncassiGiornalieri) != 0) {
        print_stmt_error(IncassiGiornalieri, "Could not view Tavoli Liberi");
        goto err;
}

MYSQL_BIND res1;
memset(&res1, 0, sizeof(res1));
res1.buffer_type= MYSQL_TYPE_LONG;
res1.buffer= &j;
res1.buffer_length = sizeof(j);

if(mysql_stmt_bind_result(IncassiGiornalieri,&res1)!=0)
{
        printf("empty result1\n");
        goto err2;
}
if(mysql_stmt_store_result(IncassiGiornalieri)!=0)
{
        printf("empty result2\n");
        goto err2;
}

if(mysql_stmt_fetch(IncassiGiornalieri)){
        goto err;
}
mysql_stmt_close(IncassiGiornalieri);
mysql_close(conn);
return j;
   err:
mysql_stmt_close(IncassiGiornalieri);
   err2:
```

```
	mysql_close(conn);
	return 0;
}
int dbIncassoMese(int i){
	if(!parse_config("users/Manager.json", &conf)) {
			fprintf(stderr, "Unable to load login configuration\n");
			exit(EXIT_FAILURE);
		}
	conn = mysql_init (NULL);
		if (conn == NULL) {
			fprintf (stderr, "mysql_init() failed (probably out of memory)\n");
			exit(EXIT_FAILURE);
		}
		if (mysql_real_connect(conn, conf.host, conf.db_username, conf.db_password,
	conf.database, conf.port, NULL, CLIENT_MULTI_STATEMENTS |
	CLIENT_MULTI_RESULTS) == NULL) {
			fprintf (stderr, "mysql_real_connect() failed\n");
			mysql_close (conn);
			exit(EXIT_FAILURE);
		}

	MYSQL_STMT *IncassiMensili;
	MYSQL_BIND param[2]; // Used both for input and output

	if(!setup_prepared_stmt(&IncassiMensili, "call IncassiMensili(?,?)", conn)) {
		print_stmt_error(IncassiMensili, "Unable to initialize login statement\n");
		goto err2;
	}
	int j=0;
	memset(param, 0, sizeof(param));
	param[0].buffer_type = MYSQL_TYPE_LONG; // IN
	param[0].buffer = &i;
	param[0].buffer_length = sizeof(i);

	param[1].buffer_type = MYSQL_TYPE_LONG; // OUT
	param[1].buffer =&j;
	param[1].buffer_length = sizeof(j);

	if (mysql_stmt_bind_param(IncassiMensili, param) != 0) { // Note _param
		print_stmt_error(IncassiMensili, "Could not bind parameters for login");
		goto err;
	}

	// Run procedure
	if (mysql_stmt_execute(IncassiMensili) != 0) {
		print_stmt_error(IncassiMensili, "Could not view Tavoli Liberi");
		goto err;
	}

	MYSQL_BIND res1;
	memset(&res1, 0, sizeof(res1));
```

```
	res1.buffer_type= MYSQL_TYPE_LONG;
	res1.buffer= &j;
	res1.buffer_length = sizeof(j);

	if(mysql_stmt_bind_result(IncassiMensili,&res1)!=0)
	{
		printf("empty result1\n");
		goto err2;
	}
	if(mysql_stmt_store_result(IncassiMensili)!=0)
	{
		printf("empty result2\n");
		goto err2;
	}

	if(mysql_stmt_fetch(IncassiMensili)){
		printf("empty result2\n");
		goto err;
	}
	mysql_stmt_close(IncassiMensili);
	mysql_close(conn);
	return j;
	  err:
	mysql_stmt_close(IncassiMensili);
	  err2:
	mysql_close(conn);
	return 0;
}
int dbVerificaCameriere(cameriere c){
	if(!parse_config("users/Cameriere.json", &conf)) {
			fprintf(stderr, "Unable to load login configuration\n");
			exit(EXIT_FAILURE);
		}
	conn = mysql_init (NULL);
		if (conn == NULL) {
			fprintf (stderr, "mysql_init() failed (probably out of memory)\n");
			exit(EXIT_FAILURE);
		}

		  if (mysql_real_connect(conn, conf.host, conf.db_username, conf.db_password,
	conf.database, conf.port, NULL, CLIENT_MULTI_STATEMENTS |
	CLIENT_MULTI_RESULTS) == NULL) {
			fprintf (stderr, "mysql_real_connect() failed\n");
			mysql_close (conn);
			exit(EXIT_FAILURE);
		}


	MYSQL_STMT *VerificaCameriere;
	MYSQL_BIND param[2]; // Used both for input and output
```

```
	if(!setup_prepared_stmt(&VerificaCameriere, "call VerificaCameriere(?, ?)", conn)) {
		print_stmt_error(VerificaCameriere, "Unable to initialize login statement\n");
		goto err2;
	}

	// Prepare parameters
	memset(param, 0, sizeof(param));

	param[0].buffer_type = MYSQL_TYPE_VAR_STRING; // IN
	param[0].buffer = c.nome;
	param[0].buffer_length = strlen(c.nome);

	param[1].buffer_type = MYSQL_TYPE_VAR_STRING; // IN
	param[1].buffer = c.cognome;
	param[1].buffer_length = strlen(c.cognome);


	if (mysql_stmt_bind_param(VerificaCameriere, param) != 0) { // Note _param
		print_stmt_error(VerificaCameriere, "Could not bind parameters for login");
		goto err;
	}

	// Run procedure
	if (mysql_stmt_execute(VerificaCameriere) != 0) {
		print_stmt_error(VerificaCameriere, "Could not execute login");
		goto err;
	}

	mysql_stmt_close(VerificaCameriere);
	mysql_close(conn);
	return 1;

	  err:
	mysql_stmt_close(VerificaCameriere);
	mysql_close(conn);
	  err2:
	return 0;
}
```

Codice file loginView.h

```
#include "../util/defines.h"
#include <stdio.h>
void loginView(credenziali * cred);
```

Codice file loginView.c

```
#include "../util/defines.h"
#include "loginView.h"
#include <stdio.h>
```

```
void loginView(credenziali * cred){
	system("clear");
	printf("##########################################\n");
	printf("##                                      ##\n");
	printf("##                                      ##\n");
	printf("##              LOGIN PIZZERIA          ##\n");
	printf("##                                      ##\n");
	printf("##                                      ##\n");
	printf("##########################################\n");
	printf("Username: ");
	getInput(128, cred->username, false);
	printf("Password: ");
	getInput(128, cred->password, true);
}
```

Codice file ManagerView.h

```
#include "../util/defines.h"
extern int ViewTavoloCliente();
extern int* ViewCreareTurni();
extern int viewTurnoTavoli();
extern prodotto viewAddNuovoElemento();
extern void viewUpdateIngrediente(ingrediente **listaIngredienti);
extern ingrediente viewAddIngrediente(void);
extern int viewIngredienti(void);
extern turno *viewTurni(turno* T);
extern cameriere viewCameriere(cameriere* T);
extern int ViewManager(void);
extern void viewADDCliente(cliente * new);
extern Tavolo ViewtavoliLiberi(Tavolo* Testa);
extern int ViewTavolo();
extern void ViewAddCameriere(cameriere * c);
extern void ViewStatoIngredienti(ingrediente *lista);
extern void ViewScontrino(prodotto * Lista,int spesa);
extern int ViewGiorno();
extern int ViewMese();
```

Codice file ManagerView.c

```
#include <stdio.h>
#include "../util/defines.h"
#include "managerView.h"

int ViewTavoloCliente(){
	int i;
	char a[4];
	system("clear");
	printf("Inserire Tavolo: ");
	do{
		getInput(4,a,false);
		i=atoi(a);
	}while(i<1);
```

```
        return i;
}
int * ViewCreareTurni(){
        int *a=malloc(sizeof(int)*2);
        system("clear");
        if(yesOrNo("Si desidera creare i turni di un mese per l'anno corrente?",'y','n',true,false)){
                a[0]=0;
        }else{
                printf("Inserire Anno");
                do{
                        char app[5];
                        getInput(5,app,false);
                        a[0]=atoi(app);
                }while(a[0]<2000 ||a[0]>2099);
        }
        printf("Inserire mese di cui si vogliono creare i turni: ");
        do{
                char app[5];
                getInput(5,app,false);
                a[1]=atoi(app);
        }while(a[1]<1 ||a[1]>12);
        return a;
}
turno *viewTurni(turno* T){
        int conto=0;
        turno * p;
        if(T==NULL){
                printf("Error\n");
                return NULL;
        }
        system("clear");
        printf("scegli il turno\n");
        for (p = T; p!=NULL; p=p->next)
        {
                conto++;
                printf("%d]%04d-%02d-%02d %02d:%02d:%02d ----%02d:%02d:%02d
                \n",conto,(p->tempo).year,(p->tempo).month,(p->tempo).day,(p->tempo).hour, (p-
                >tempo).minute, (p->tempo).second,(p->oraFine).hour, (p->oraFine).minute, (p-
                >oraFine).second);
        }
        char app[3];
        int c;
        do{
                getInput(3,app,false);
                c=atoi(app);
        }while(c<1 ||c>conto);
        p=T;
        for (conto = 1; conto<c; conto=conto+1)
        {p=p->next;}
        return p;
}
```

```
cameriere viewCameriere(cameriere * T){
	int conto=0;
	cameriere * p=NULL;
	if(T==NULL){
		printf("Error\n");
		return *p;
	}
	system("clear");
	printf("scegli il cameriere\n");
	for (p = T; p!=NULL; p=p->next)
	{
		conto++;
		printf("%d] %s %s\n",conto,p->nome,p->cognome);
	}
	char app[3];
	int c;
	do{
		getInput(3,app,false);
		c=atoi(app);
	}while(c<1 ||c>conto);
	p=T;
	for (conto = 1; conto<c; conto=conto+1)
	{
		p=p->next;
	}
	return *p;
}
int ViewManager(void){
	system("clear");
	printf("################################################\n");
	printf("## ##\n");
	printf("## MANAGER ##\n");
	printf("## ##\n");
	printf("################################################\n");
	printf("1]Registrare nuovo cliente\n");
	printf("2]Stampare scontrontrino di un cliente\n");
	printf("3]Stampare l'incasso di un giorno\n");
	printf("4]Stampare l'incasso mensile\n");
	printf("5]Aggiungere lista degli Ingredienti\n");
	printf("6]Assumere nuovo Cameriere\n");
	printf("7]Aggiungere nuovo elemento del menu\n");
	printf("8]Vedere Stato Ingredienti in dispensa\n");
	printf("9]Aggiungere Nuovo Tavolo\n");
	printf("10]Assegna cameriere Tavoli\n");
	printf("11]Creare Turni per il mese corrente\n");
	printf("12]Assegnare Turno Al tavolo\n");
	printf("13]Cambiare Utente\n");
	printf("14]Chiudere programma\n");
	int c;
	char app[3];
	do{
```

```
                getInput(3,app,false);
                c=atoi(app);
        }while(c<1 ||c>14);
        return c;
}
int ViewTavolo(){
        int c;
        system("clear");
        printf("dimensione del nuovo tavolo: ");
        do{
                char app[3];
                getInput(2,app,false);
                c=atoi(app);
        }while(c<1);
        return c;
}
int viewTurnoTavoli(){
        system("clear");
        char app[4];
        int i=0;
        printf("quale Tavolo si vuole usare? ");
        do{
                getInput(4,app,false);
                i=atoi(app);
        }while (i<=0);
        return i;
}
void ViewAddCameriere(cameriere *c)
{
        system("clear");
        printf("Nome: ");
        getInput(16, c->nome, false);
        printf("Cognome: ");
        getInput(16, c->cognome, false);
        return;
}
int viewIngredienti(){
        system("clear");
        printf("Cosa desideri fare?\n");
        printf("1]Inserirre nuovo Elemento\n");
        printf("2]Aumentare la quantita di elementi già presenti\n");
        int c;
        char app[3];
        do{
                getInput(3,app,false);
                c=atoi(app);
        }while(c<1 ||c>2);
        return c;
}
ingrediente viewAddIngrediente(){
        system("clear");
```

```
	ingrediente i;
	char app[4];
	printf("Inserire Nome nuovo Ingrediente: ");
	getInput(16,i.nome,false);
	printf("Inserire Quantita nuovo Ingrediente: ");
	do{
		getInput(4,app,false);
		i.quantita=atoi(app);
	}while (i.quantita<0);
	if(!yesOrNo("é una possibile Aggiunta delle pizze?",'y','n',true,false)){
		i.costo=0;
	}else{
		printf("Inserire Costo nuovo Ingrediente: ");
		do{
			getInput(4,app,false);
			i.costo=atoi(app);
		}while (i.costo<0);
	}
	return i;
}
void viewUpdateIngrediente(ingrediente **listaIngredienti)
{
	do
	{
		system("clear");
		ingrediente *app=malloc(sizeof(ingrediente));
		app->next=NULL;
		if(*listaIngredienti==NULL) *listaIngredienti=app;
		else{
			ingrediente *app2=*listaIngredienti;
			while(app2->next!=NULL) app2=app2->next;
			app2->next=app;
		}
		printf("Inserire Nome Ingrediente: ");
		getInput(16,app->nome,false);
		printf("Inserire Quantita da aggiungere Ingrediente: ");
		do{
			char n[4];
			getInput(4,n,false);
			app->quantita=atoi(n);
		}while (app->quantita<0);
	}while(yesOrNo("si desidera Aggiungere altri ingredienti da aggiornare?",'y','n',true,false));
	return ;
}
prodotto viewAddNuovoElemento(){
	prodotto prod;
	system("clear");
	printf("Nome nuovo prodotto: ");
	getInput(16, prod.nome, false);
	printf("prezzo: ");
	char app[4];
```

```
	do{
		getInput(4, app, false);
		prod.prezzo=atoi(app);
	}while(prod.prezzo<=0);
	prod.tipo=!yesOrNo("é una bevanda?",'y','n',true,false);
	prod.listaIngredienti=NULL;
	printf("Elenco degli ingredienti:\n");
	do
	{
		ingrediente *app=malloc(sizeof(ingrediente));
		app->next=NULL;
		if(prod.listaIngredienti ==NULL) prod.listaIngredienti=app;
		else{
			ingrediente * app2=prod.listaIngredienti;
			while(app2->next!=NULL) app2=app2->next;
			app2->next=app;
		}
		printf("Inserire Nome Ingrediente: ");
		getInput(16,app->nome,false);
		printf("Inserire la quantita che si utilizza: ");
		do{
			char n[4];
			getInput(4,n,false);
			app->quantita=atoi(n);
		}while (app->quantita<0);
	}while(yesOrNo("si utilizzano altri ingredienti?",'y','n',true,false));
	return prod;
}
void viewADDCliente(cliente * new){
	char app[2];
	system("clear");
	printf("Nome: ");
	getInput(16, new->nome, false);
	printf("Cognome: ");
	getInput(16, new->cognome, false);
	printf("Numero di commensali: ");
	getInput(3,app,false);
	new->N_Persone=atoi(app);
}
Tavolo ViewtavoliLiberi(Tavolo* Testa){
	system("clear");
	Tavolo t;
	Tavolo * p;
	printf("Lista dei tavoli Liberi:\n");
	for(p=Testa;p!=NULL;p=p->next){
		printf("Numero:%d \n",p->N_Tavolo);
	}
	int app;
	int c=0;
	printf("sceglierne uno: \n");
	do{
```

```
                scanf("%d",&app);
                t.N_Tavolo=app;
                for(p=Testa;p!=NULL;p=p->next){
                        if(p->N_Tavolo==t.N_Tavolo){
                                c=1;
                                t.turno=p->turno;
                        }
                }
                if(c!=1)printf("valore non valido RE-INSERIRE");
        }while(c!=1);
        return t;
}
void ViewStatoIngredienti(ingrediente *lista){
        ingrediente *p;
        system("clear");
        printf("----------------Stato degli Ingredienti----------------\n");
        for (p= lista; p!=NULL; p=p->next)
        {
                printf("Ingrediente: %s\tquantita rimanente: %d\n",p->nome,p->quantita);
        }
}
void ViewScontrino(prodotto * Lista,int spesa){
        prodotto * i;
        int j=1;
        system("clear");
        printf("----------------------Scontrino----------------------\n");
        for(i = Lista; i != NULL; i=i->next)
        {
                printf("%d]prodotto: %s %d X %d€\tcosto totale prodotto=%d€\n",j,i->nome,i-
                >quantita,i->prezzo,i->CostoTotaleProdotto);
                ingrediente *t;
                for ( t = i->listaIngredienti; t!=NULL; t=t->next)
                {
                        printf("\t\tAggiunta: %s %d X %d€\n",t->nome,t->quantita,t->costo);
                }
                printf("-------------------------------------------------\n");
                j++;
        }
        printf("\t\t\t\t\tCosto totale: %d\n",spesa);
        getchar();
        return;
}
int ViewGiorno(){
        system("clear");
        int c=0;
        char app[4];
        printf("di quale giorno del mese corrente si vuole visualizzare l'incasso totale? ");
        do{
                getInput(4,app,false);
                c=atoi(app);
        }while(c<1 || c>31);
```

```
	return c;
}
int ViewMese(){
	system("clear");
	int c=0;
	char app[4];
	printf("di quale mese dell' anno corrente si vuole visualizzare l'incasso totale? ");
	do{
		getInput(4,app,false);
		c=atoi(app);
	}while(c<1 || c>12);
	return c;
}
```

Codice file pizzaioloView.h

```
#pragma once
#include "../util/defines.h"
extern int ViewPizzaFinita(void);
extern int ViewPizzaiolo(void);
extern void stampaListaPizze(prodotto * testa);
```

Codice file pizzaioloView.C

```
#include "../util/defines.h"
#include "pizzaioloView.h"
#include <stdio.h>

void stampaListaPizze(prodotto * testa){
	system("clear");
	printf("---------------Ordinazioni----------------\n");
	prodotto * p=NULL;
	for(p=testa;p!=NULL;p=p->next)
	{
		printf("%d] %s %d\n",p->prezzo,p->nome,p->tipo);
		if(p->listaIngredienti!=NULL)
		{
			ingrediente * i=p->listaIngredienti;
			for(i=p->listaIngredienti;i!=NULL;i=i->next)
			{
				printf("\t%s %d\n",i->nome,i->quantita);
			}
		}
	}
}
int ViewPizzaFinita(){
	int i;
	system("clear");
	printf("inserire il numero della Pizza pronta per la consegna: ");
	char app[4];
	do{
		getInput(4,app,false);
		i=atoi(app);
```

```
	}while(i<=0);
	return i;
}
int ViewPizzaiolo(){
	system("clear");
	printf("################################################\n");
	printf("## ##\n");
	printf("## Pizzaiolo ##\n");
	printf("## ##\n");
	printf("################################################\n");
	printf("1]Visualizza ordini da preparare\n");
	printf("2]Ordine Pronto\n");
	printf("3]Cambia Utente\n");
	printf("4]Chiudi Applicazione\n");
	int c;
	char app[3];
	do{
		getInput(3,app,false);
		c=atoi(app);
	}while(c<1 ||c>4);
	return c;
}
```

Codice file Utils.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "defines.h"

void print_stmt_error (MYSQL_STMT *stmt, char *message)
{
	fprintf (stderr, "%s\n", message);
	if (stmt != NULL) {
		fprintf (stderr, "Error %u (%s): %s\n",
		mysql_stmt_errno (stmt),
		mysql_stmt_sqlstate(stmt),
		mysql_stmt_error (stmt));
	}
}


void print_error(MYSQL *conn, char *message)
{
	fprintf (stderr, "%s\n", message);
	if (conn != NULL) {
		#if MYSQL_VERSION_ID >= 40101
		fprintf (stderr, "Error %u (%s): %s\n",
		mysql_errno (conn), mysql_sqlstate(conn), mysql_error (conn));
		#else
```

```
                fprintf (stderr, "Error %u: %s\n",
                mysql_errno (conn), mysql_error (conn));
                #endif
        }
        }

bool setup_prepared_stmt(MYSQL_STMT **stmt, char *statement, MYSQL *conn)
{
        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->max_length + 2; j++)
                        putchar('-');
                putchar('+');
        }
        putchar('\n');
}

static void dump_result_set_header(MYSQL_RES *res_set)
{
        MYSQL_FIELD *field;
        unsigned long col_len;
        unsigned int i;

        /* determine column display widths -- requires result set to be */
        /* generated with mysql_store_result(), not mysql_use_result() */

        mysql_field_seek (res_set, 0);

        for (i = 0; i < mysql_num_fields (res_set); i++) {
                field = mysql_fetch_field (res_set);
                col_len = strlen(field->name);

                if (col_len < field->max_length)
                        col_len = field->max_length;
                if (col_len < 4 && !IS_NOT_NULL(field->flags))
                        col_len = 4; /* 4 = length of the word "NULL" */
                field->max_length = col_len; /* reset column info */
        }
        print_dashes(res_set);
        putchar('|');
        mysql_field_seek (res_set, 0);
        for (i = 0; i < mysql_num_fields(res_set); i++) {
                field = mysql_fetch_field(res_set);
                printf(" %-*s |", (int)field->max_length, field->name);
        }
        putchar('\n');
        print_dashes(res_set);
}

void dump_result_set(MYSQL *conn, MYSQL_STMT *stmt, char *title)
{
        int i;
        int status;
        int num_fields;       /* number of columns in result */
        MYSQL_FIELD *fields;  /* for result set metadata */
        MYSQL_BIND *rs_bind;  /* for output buffers */
        MYSQL_RES *rs_metadata;
        MYSQL_TIME *date;
        size_t attr_size;
```

```
/* Prefetch the whole result set. This in conjunction with
 * STMT_ATTR_UPDATE_MAX_LENGTH set in `setup_prepared_stmt`
 * updates the result set metadata which are fetched in this
 * function, to allow to compute the actual max length of
 * the columns.
 */
if (mysql_stmt_store_result(stmt)) {
fprintf(stderr, " mysql_stmt_execute(), 1 failed\n");
fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
exit(0);
}

/* the column count is > 0 if there is a result set */
/* 0 if the result is only the final status packet */
num_fields = mysql_stmt_field_count(stmt);

if (num_fields > 0) {
/* there is a result set to fetch */
printf("%s\n", title);

if((rs_metadata = mysql_stmt_result_metadata(stmt)) == NULL) {
finish_with_stmt_error(conn, stmt, "Unable to retrieve result metadata\n", true);
}

dump_result_set_header(rs_metadata);

fields = mysql_fetch_fields(rs_metadata);

rs_bind = (MYSQL_BIND *)malloc(sizeof (MYSQL_BIND) * num_fields);
if (!rs_bind) {
finish_with_stmt_error(conn, stmt, "Cannot allocate output buffers\n", true);
}
memset(rs_bind, 0, sizeof (MYSQL_BIND) * num_fields);

/* set up and bind result set output buffers */
for (i = 0; i < num_fields; ++i) {

// Properly size the parameter buffer
switch(fields[i].type) {
case MYSQL_TYPE_DATE:
case MYSQL_TYPE_TIMESTAMP:
case MYSQL_TYPE_DATETIME:
case MYSQL_TYPE_TIME:
attr_size = sizeof(MYSQL_TIME);
break;
case MYSQL_TYPE_FLOAT:
attr_size = sizeof(float);
break;
case MYSQL_TYPE_DOUBLE:
attr_size = sizeof(double);
break;
```

```
case MYSQL_TYPE_TINY:
attr_size = sizeof(signed char);
break;
case MYSQL_TYPE_SHORT:
case MYSQL_TYPE_YEAR:
attr_size = sizeof(short int);
break;
case MYSQL_TYPE_LONG:
case MYSQL_TYPE_INT24:
attr_size = sizeof(int);
break;
case MYSQL_TYPE_LONGLONG:
attr_size = sizeof(int);
break;
default:
attr_size = fields[i].max_length;
break;
}

// Setup the binding for the current parameter
rs_bind[i].buffer_type = fields[i].type;
rs_bind[i].buffer = malloc(attr_size + 1);
rs_bind[i].buffer_length = attr_size + 1;

if(rs_bind[i].buffer == NULL) {
finish_with_stmt_error(conn, stmt, "Cannot allocate output buffers\n", true);
}
}

if(mysql_stmt_bind_result(stmt, rs_bind)) {
finish_with_stmt_error(conn, stmt, "Unable to bind output parameters\n", true);
}

/* fetch and display result set rows */
while (true) {
status = mysql_stmt_fetch(stmt);

if (status == 1 || status == MYSQL_NO_DATA)
break;

putchar('|');

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

if (rs_bind[i].is_null_value) {
printf (" %-*s |", (int)fields[i].max_length, "NULL");
continue;
}

switch (rs_bind[i].buffer_type) {
```

```
        case MYSQL_TYPE_VAR_STRING:
        case MYSQL_TYPE_DATETIME:
        printf(" %-*s |", (int)fields[i].max_length, (char*)rs_bind[i].buffer);
        break;

        case MYSQL_TYPE_DATE:
        case MYSQL_TYPE_TIMESTAMP:
        date = (MYSQL_TIME *)rs_bind[i].buffer;
        printf(" %d-%02d-%02d |", date->year, date->month, date->day);
        break;

        case MYSQL_TYPE_STRING:
        printf(" %-*s |", (int)fields[i].max_length, (char *)rs_bind[i].buffer);
        break;

        case MYSQL_TYPE_FLOAT:
        case MYSQL_TYPE_DOUBLE:
        printf(" %.02f |", *(float *)rs_bind[i].buffer);
        break;

        case MYSQL_TYPE_LONG:
        case MYSQL_TYPE_SHORT:
        case MYSQL_TYPE_TINY:
        printf(" %-*d |", (int)fields[i].max_length, *(int *)rs_bind[i].buffer);
        break;

        case MYSQL_TYPE_NEWDECIMAL:
        printf(" %-*.02lf |", (int)fields[i].max_length, *(float*) rs_bind[i].buffer);
        break;

        default:
           printf("ERROR: Unhandled type (%d)\n", rs_bind[i].buffer_type);
           abort();
        }
        }
        putchar('\n');
        print_dashes(rs_metadata);
        }

        mysql_free_result(rs_metadata); /* free metadata */

        /* free output buffers */
        for (i = 0; i < num_fields; i++) {
                free(rs_bind[i].buffer);
        }
        free(rs_bind);
        }
}
```

Codice file Parse.c

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "defines.h"

#define BUFF_SIZE 4096

// The final config struct will point into this
static char config[BUFF_SIZE];

/**
* JSON type identifier. Basic types are:
*       o Object
*       o Array
*       o String
*       o Other primitive: number, boolean (true/false) or null
*/
typedef enum {
JSMN_UNDEFINED = 0,
JSMN_OBJECT = 1,
JSMN_ARRAY = 2,
JSMN_STRING = 3,
JSMN_PRIMITIVE = 4
} jsmntype_t;

enum jsmnerr {
/* Not enough tokens were provided */
JSMN_ERROR_NOMEM = -1,
/* Invalid character inside JSON string */
JSMN_ERROR_INVAL = -2,
/* The string is not a full JSON packet, more bytes expected */
JSMN_ERROR_PART = -3
};

/**
* JSON token description.
* type          type (object, array, string etc.)
* start  start position in JSON data string
* end           end position in JSON data string
*/
typedef struct {
jsmntype_t type;
int start;
int end;
int size;
#ifdef JSMN_PARENT_LINKS
int parent;
#endif
```

```
} jsmntok_t;

/**
* JSON parser. Contains an array of token blocks available. Also stores
* the string being parsed now and current position in that string
*/
typedef struct {
unsigned int pos; /* offset in the JSON string */
unsigned int toknext; /* next token to allocate */
int toksuper; /* superior token node, e.g parent object or array */
} jsmn_parser;

/**
* Allocates a fresh unused token from the token pool.
*/
static jsmntok_t *jsmn_alloc_token(jsmn_parser *parser, jsmntok_t *tokens, size_t num_tokens) {
jsmntok_t *tok;
if (parser->toknext >= num_tokens) {
return NULL;
}
tok = &tokens[parser->toknext++];
tok->start = tok->end = -1;
tok->size = 0;
#ifdef JSMN_PARENT_LINKS
tok->parent = -1;
#endif
return tok;
}

/**
* Fills token type and boundaries.
*/
static void jsmn_fill_token(jsmntok_t *token, jsmntype_t type,
                            int start, int end) {
token->type = type;
token->start = start;
token->end = end;
token->size = 0;
}

/**
* Fills next available token with JSON primitive.
*/
static int jsmn_parse_primitive(jsmn_parser *parser, const char *js,
size_t len, jsmntok_t *tokens, size_t num_tokens) {
jsmntok_t *token;
int start;

start = parser->pos;

for (; parser->pos < len && js[parser->pos] != '\0'; parser->pos++) {
```

```
switch (js[parser->pos]) {
#ifndef JSMN_STRICT
/* In strict mode primitive must be followed by "," or "}" or "]" */
case ':':
#endif
case '\t' : case '\r' : case '\n' : case ' ' :
case ','  : case ']'  : case '}' :
goto found;
}
if (js[parser->pos] < 32 || js[parser->pos] >= 127) {
parser->pos = start;
return JSMN_ERROR_INVAL;
}
}
#ifdef JSMN_STRICT
/* In strict mode primitive must be followed by a comma/object/array */
parser->pos = start;
return JSMN_ERROR_PART;
#endif

found:
if (tokens == NULL) {
parser->pos--;
return 0;
}
token = jsmn_alloc_token(parser, tokens, num_tokens);
if (token == NULL) {
parser->pos = start;
return JSMN_ERROR_NOMEM;
}
jsmn_fill_token(token, JSMN_PRIMITIVE, start, parser->pos);
#ifdef JSMN_PARENT_LINKS
token->parent = parser->toksuper;
#endif
parser->pos--;
return 0;
}

/**
* Fills next token with JSON string.
*/
static int jsmn_parse_string(jsmn_parser *parser, const char *js,
size_t len, jsmntok_t *tokens, size_t num_tokens) {
jsmntok_t *token;

int start = parser->pos;

parser->pos++;

/* Skip starting quote */
for (; parser->pos < len && js[parser->pos] != '\0'; parser->pos++) {
```

```
char c = js[parser->pos];

/* Quote: end of string */
if (c == '\"') {
if (tokens == NULL) {
return 0;
}
token = jsmn_alloc_token(parser, tokens, num_tokens);
if (token == NULL) {
parser->pos = start;
return JSMN_ERROR_NOMEM;
}
jsmn_fill_token(token, JSMN_STRING, start+1, parser->pos);
#ifdef JSMN_PARENT_LINKS
token->parent = parser->toksuper;
#endif
return 0;
}

/* Backslash: Quoted symbol expected */
if (c == '\\' && parser->pos + 1 < len) {
int i;
parser->pos++;
switch (js[parser->pos]) {
/* Allowed escaped symbols */
case '\"': case '/' : case '\\' : case 'b' :
case 'f' : case 'r' : case 'n'  : case 't' :
break;
/* Allows escaped symbol \uXXXX */
case 'u':
parser->pos++;
for(i = 0; i < 4 && parser->pos < len && js[parser->pos] != '\0'; i++) {
/* If it isn't a hex character we have an error */
if(!((js[parser->pos] >= 48 && js[parser->pos] <= 57) || /* 0-9 */
(js[parser->pos] >= 65 && js[parser->pos] <= 70) || /* A-F */
(js[parser->pos] >= 97 && js[parser->pos] <= 102))) { /* a-f */
parser->pos = start;
return JSMN_ERROR_INVAL;
}
parser->pos++;
}
parser->pos--;
break;
/* Unexpected symbol */
default:
parser->pos = start;
return JSMN_ERROR_INVAL;
}
}
}
parser->pos = start;
```

```
return JSMN_ERROR_PART;
}

/**
* Parse JSON string and fill tokens.
*/
static int jsmn_parse(jsmn_parser *parser, const char *js, size_t len, jsmntok_t *tokens, unsigned int
num_tokens)
{
	int r;
	int i;
	jsmntok_t *token;
	int count = parser->toknext;

	for (; parser->pos < len && js[parser->pos] != '\0'; parser->pos++) {
		char c;
		jsmntype_t type;

		c = js[parser->pos];
		switch (c) {
			case '{': case '[':
				count++;
				if (tokens == NULL) {
					break;
				}
				token = jsmn_alloc_token(parser, tokens, num_tokens);
				if (token == NULL)
					return JSMN_ERROR_NOMEM;
				if (parser->toksuper != -1) {
					tokens[parser->toksuper].size++;
					#ifdef JSMN_PARENT_LINKS
					token->parent = parser->toksuper;
					#endif
				}
				token->type = (c == '{' ? JSMN_OBJECT : JSMN_ARRAY);
				token->start = parser->pos;
				parser->toksuper = parser->toknext - 1;
			break;
			case '}': case ']':
				if (tokens == NULL)
					break;
				type = (c == '}' ? JSMN_OBJECT : JSMN_ARRAY);
				#ifdef JSMN_PARENT_LINKS
				if (parser->toknext < 1) {
					return JSMN_ERROR_INVAL;
				}
				token = &tokens[parser->toknext - 1];
				for (;;) {
					if (token->start != -1 && token->end == -1) {
						if (token->type != type) {
							return JSMN_ERROR_INVAL;
```

```
                                }
                                token->end = parser->pos + 1;
                                parser->toksuper = token->parent;
                                break;
                        }
                        if (token->parent == -1) {
                                if(token->type != type || parser->toksuper == -1) {
                                        return JSMN_ERROR_INVAL;
                                }
                                break;
                        }
                        token = &tokens[token->parent];
                }
                #else
                for (i = parser->toknext - 1; i >= 0; i--) {
                        token = &tokens[i];
                        if (token->start != -1 && token->end == -1) {
                                if (token->type != type) {
                                        return JSMN_ERROR_INVAL;
                                }
                                parser->toksuper = -1;
                                token->end = parser->pos + 1;
                                break;
                        }
                }
                /* Error if unmatched closing bracket */
                if (i == -1) return JSMN_ERROR_INVAL;
                for (; i >= 0; i--) {
                        token = &tokens[i];
                        if (token->start != -1 && token->end == -1) {
                                parser->toksuper = i;
                                break;
                        }
                }
                #endif
        break;
case '\"':
        r = jsmn_parse_string(parser, js, len, tokens, num_tokens);
        if (r < 0) return r;
        count++;
        if (parser->toksuper != -1 && tokens != NULL)
                tokens[parser->toksuper].size++;
break;
case '\t' : case '\r' : case '\n' : case ' ':
break;
case ':':
        parser->toksuper = parser->toknext - 1;
break;
case ',':
if (tokens != NULL && parser->toksuper != -1 &&
tokens[parser->toksuper].type != JSMN_ARRAY &&
```

```
tokens[parser->toksuper].type != JSMN_OBJECT) {
        #ifdef JSMN_PARENT_LINKS
        parser->toksuper = tokens[parser->toksuper].parent;
        #else
        for (i = parser->toknext - 1; i >= 0; i--) {
                if (tokens[i].type == JSMN_ARRAY || tokens[i].type ==
                JSMN_OBJECT) {
                        if (tokens[i].start != -1 && tokens[i].end == -1) {
                                parser->toksuper = i;
                                break;
                        }
                }
        }
        #endif
        }
break;
#ifdef JSMN_STRICT
/* In strict mode primitives are: numbers and booleans */
case '-': case '0': case '1' : case '2': case '3' : case '4':
case '5': case '6': case '7' : case '8': case '9':
case 't': case 'f': case 'n' :
/* And they must not be keys of the object */
if (tokens != NULL && parser->toksuper != -1) {
jsmntok_t *t = &tokens[parser->toksuper];
if (t->type == JSMN_OBJECT ||
(t->type == JSMN_STRING && t->size != 0)) {
return JSMN_ERROR_INVAL;
}
}
#else
/* In non-strict mode every unquoted value is a primitive */
default:
#endif
r = jsmn_parse_primitive(parser, js, len, tokens, num_tokens);
if (r < 0) return r;
count++;
if (parser->toksuper != -1 && tokens != NULL)
tokens[parser->toksuper].size++;
break;

#ifdef JSMN_STRICT
/* Unexpected char in strict mode */
default:
return JSMN_ERROR_INVAL;
#endif
}
}

if (tokens != NULL) {
for (i = parser->toknext - 1; i >= 0; i--) {
/* Unmatched opened object or array */
```

```
                if (tokens[i].start != -1 && tokens[i].end == -1) {
                return JSMN_ERROR_PART;
                }
                }
        }

        return count;
}

/**
* Creates a new parser based over a given  buffer with an array of tokens
* available.
*/
static void jsmn_init(jsmn_parser *parser) {
        parser->pos = 0;
        parser->toknext = 0;
        parser->toksuper = -1;
}

static int jsoneq(const char *json, jsmntok_t *tok, const char *s)
{
        if (tok->type == JSMN_STRING
           && (int) strlen(s) == tok->end - tok->start
           && strncmp(json + tok->start, s, tok->end - tok->start) == 0) {
                return 0;
        }
        return -1;
}

static size_t load_file(char *filename)
{
        FILE *f = fopen(filename, "rb");
        if(f == NULL) {
                fprintf(stderr, "Unable to open file %s\n", filename);
                exit(1);
        }

        fseek(f, 0, SEEK_END);
        size_t fsize = ftell(f);
        fseek(f, 0, SEEK_SET);  //same as rewind(f);

        if(fsize >= BUFF_SIZE) {
                fprintf(stderr, "Configuration file too large\n");
                abort();
        }

        fread(config, fsize, 1, f);
        fclose(f);

        config[fsize] = 0;
        return fsize;
```

```
}

int parse_config(char *path, struct configuration *conf)
{
	int i;
	int r;
	jsmn_parser p;
	jsmntok_t t[128]; /* We expect no more than 128 tokens */

	load_file(path);

	jsmn_init(&p);
	r = jsmn_parse(&p, config, strlen(config), t, sizeof(t)/sizeof(t[0]));
	if (r < 0) {
		printf("Failed to parse JSON: %d\n", r);
		return 0;
	}

	/* Assume the top-level element is an object */
	if (r < 1 || t[0].type != JSMN_OBJECT) {
		printf("Object expected\n");
		return 0;
	}

	/* Loop over all keys of the root object */
	for (i = 1; i < r; i++) {
		if (jsoneq(config, &t[i], "host") == 0) {
		/* We may use strndup() to fetch string value */
		conf->host = strndup(config + t[i+1].start, t[i+1].end-t[i+1].start);
		i++;
	} else
		if (jsoneq(config, &t[i], "username") == 0) {
			conf->db_username = strndup(config + t[i+1].start, t[i+1].end-t[i+1].start);
			i++;
		} else
			if (jsoneq(config, &t[i], "password") == 0) {
				conf->db_password = strndup(config + t[i+1].start, t[i+1].end-
			t[i+1].start);
				i++;
			} else
				if (jsoneq(config, &t[i], "port") == 0) {
					conf->port = strtol(config + t[i+1].start, NULL, 10);
					i++;
				} else if (jsoneq(config, &t[i], "database") == 0) {
					conf->database = strndup(config + t[i+1].start, t[i+1].end-
				t[i+1].start);
					i++;
					} else {
						printf("Unexpected key: %.*s\n", t[i].end-t[i].start,
					config + t[i].start);
					}
```

```
                    }
    return 1;
```