Basi di Dati e Conoscenza

Progetto A.A. 2020/2021

Sistema di gestione di una pizzeria

0272911

Stefano Belli

Indice

1. Descrizione del Minimondo

Si vuole progettare il backend di un sistema informativo per la gestione dell'operatività di una pizzeria. In tale pizzeria è di interesse tenere traccia dei tavoli disponibili ed assegnati, dei camerieri associati ai tavoli, dei pizzaioli che preparano le pizze, del barista, del manager. Ciascuno dei lavoratori della pizzeria ha differenti mansioni e può effettuare operazioni differenti all'interno del sistema.

All'ingresso di un cliente, il manager lo riceve e lo registra, segnando nome, cognome e numero di commensali, assegnando un tavolo disponibile in grado di ospitarli tutti.

Un cameriere ha sempre la possibilità di visualizzare quali tavoli a lui assegnati sono occupati e quali sono stati serviti. Al momento di prendere l'ordine, il cameriere registra la comanda. Parte delle ordinazioni sono espletate dal barista, parte dal pizzaiolo.

Barista, 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
2	Assegnati	Occupati	Il tavolo viene occupato dai clienti
3	Associati	Assegnati	Al cameriere viene assegnato il tavolo
10	Comanda	Ordinazione	E' stato scelto il termine ordinazione per riferirsi allo stesso concetto
12	Comanda	Ordinazione	E' stato scelto il termine ordinazione per riferirsi allo stesso concetto
12	Bevanda	Prodotto nel menu bar	Risulta più chiaro che ci si riferisce al menu
12	Pizza	Prodotto nel menu pizzeria	Risulta più chiaro che ci si riferisce al menu
14	Ordini	Ordinazioni	E' stato scelto il termine ordinazione per riferirsi allo stesso concetto
20	Menu	Ordinazione	L'aggiunta dell'ingrediente avviene per ordinazione, cioè è facoltativa
24	Prodotti	Ingredienti	La motivazione che segue fa intendere che i prodotti di interesse in questo caso siano gli ingredienti
24	Prodotto	Ingrediente aggiuntivo	Come la motivazione precedente
25	Ordine	Ordinazione	E' stato scelto il termine ordinazione per riferirsi allo stesso concetto
27	Ordine	Tavolo occupato	Lo scontrino è riferito al tavolo occupato e alla completa consumazione

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 occupati, dei camerieri assegnati 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 l'ordinazione. 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 dell'ordinazione. Quando hanno preparato un prodotto del menu bar o un prodotto del

menu pizzeria, il cameriere può visualizzare cosa è pronto (in relazione alle ordinazioni) 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. Per l'ordinazione 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 ingrediente aggiuntivo che non è disponibile, questo non potrà essere aggiunto all'ordinazione.

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

Glossario dei Termini

Termine	Descrizione	Sinonimi	Collegamenti
Lavoratore	Lavoratore in ambito della pizzeria		Cameriere, Pizzaiolo, Barista, Manager
Cameriere	Il cameriere prende ordinazioni dai clienti accomodati nei tavoli e consegna le ordinazioni pronte	Lavoratore	Tavolo, Lavoratore, Turno
Pizzaiolo	Il pizzaiolo prepara le ordinazioni riguardanti la pizza	Lavoratore	Ordinazione. Lavoratore
Barista	Il barista prepara le ordinazioni riguardanti bevande	Lavoratore	Ordinazione, Lavoratore
Manager	Il manager gestisce il ristorante nel suo complesso	Lavoratore	Tavolo, Lavoratore, Scontrino, Menu, Turno
Ordinazione	Cosa desiderano i clienti di un determinato tavolo	Comanda, Ordine	Cameriere, Barista, Pizzaiolo, Tavolo, Prodotto nel menu pizzeria, Prodotto nel menu bar

Prodotto nel menu pizzeria	Riguarda una pizza che è consumabile e rintracciabile nel menu	Pizza	Menu
Prodotto nel menu bar	Riguarda una bevanda che è consumabile e rintracciabile nel menu	Bevanda	Menu
Turno	Descrizione di un particolare turno completo di camerieri richiesti e tavoli utilizzati per esso.		Camerieri, Tavoli, Manager
Menu	Descrizione del menu scelto esclusivamente dal manager, non-variabile in funzione del turno specifico.		Manager, Prodotto nel menu pizzeria, Prodotto nel menu bar
Ingrediente	Ingrediente utilizzato per il prodotto nel menu pizzeria/bar finale e/o aggiungerlo ad un prodotto finale su eventuale richiesta del cliente	Prodotto	Prodotto nel menu pizzeria, Prodotto nel menu bar
Scontrino	Scontrini non ancora eliminati che permangono nella base di dati.		Manager, Tavolo occupato
Tavolo occupato	Tavolo occupato ad una determinata data e ora dal cliente con un certo numero di commensali.		Tavolo
Cliente	Cliente che occupa il tavolo, si tiene traccia del suo nome e cognome.		Tavolo occupato

Raggruppamento dei requisiti in insiemi omogenei

Frasi di carattere generale
In tale pizzeria è di interesse tenere traccia dei tavoli disponibili ed occupati, dei camerieri assegnati 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. 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.

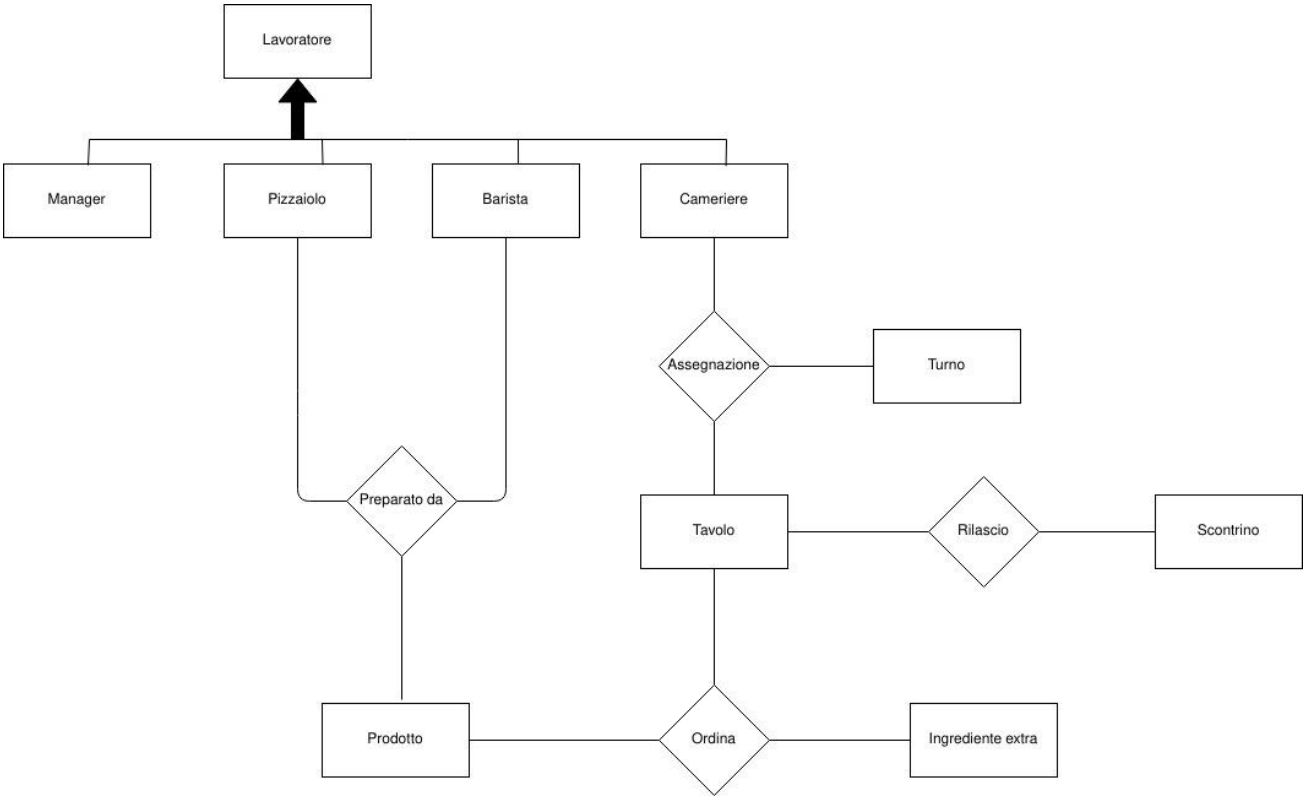
Frasi relative al manager
All'ingresso di un cliente, il manager lo riceve e lo registra, segnando nome, cognome e numero di commensali, assegnando un tavolo disponibile in grado di ospitarli tutti. Il manager può definire quali camerieri lavorano in quali turni e quali tavoli sono utilizzati in quali turni. Il menu è unico per tutti i turni e definito dal manager, con i rispettivi prezzi. Allo stesso modo, il manager ha la possibilità di tenere traccia delle disponibilità dei singoli ingredienti. Il manager ha la possibilità di stampare lo scontrino di un tavolo occupato. Inoltre, per motivi statistici, ha la possibilità di visualizzare le entrate giornaliere e/o mensili.
Frasi relative al cameriere
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 l'ordinazione. Quando hanno preparato un prodotto del menu bar o un prodotto del menu pizzeria, il cameriere può visualizzare cosa è pronto (in relazione alle ordinazioni) e sapere cosa deve consegnare a quale tavolo.
Frasi relative al pizzaiolo e al barista
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 dell'ordinazione. Quando hanno preparato un prodotto del menu bar o un prodotto del menu pizzeria […] (Segnalazione di ordine espletato)
Frasi relative alle ordinazioni
Per l'ordinazione 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. In questo modo, se viene ordinato ad un cameriere da un cliente un ingrediente aggiuntivo che non è disponibile, questo non potrà essere aggiunto all'ordinazione.

3. Progettazione concettuale

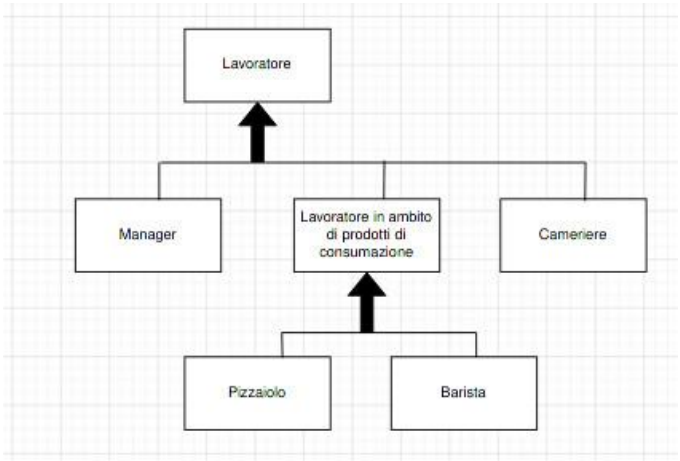
Costruzione dello schema E-R

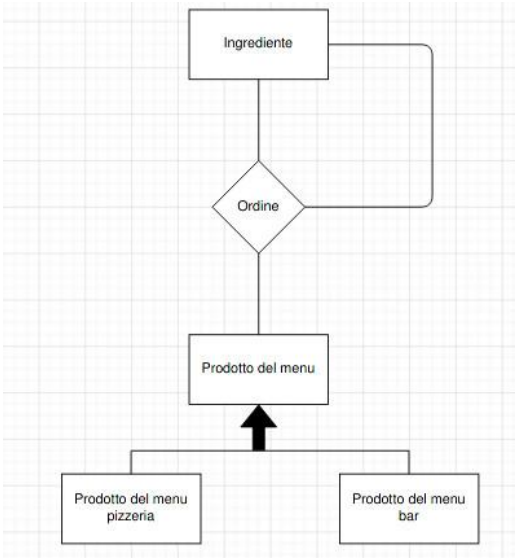
Per la progettazione concettuale è stato seguito un approccio **top-down**:

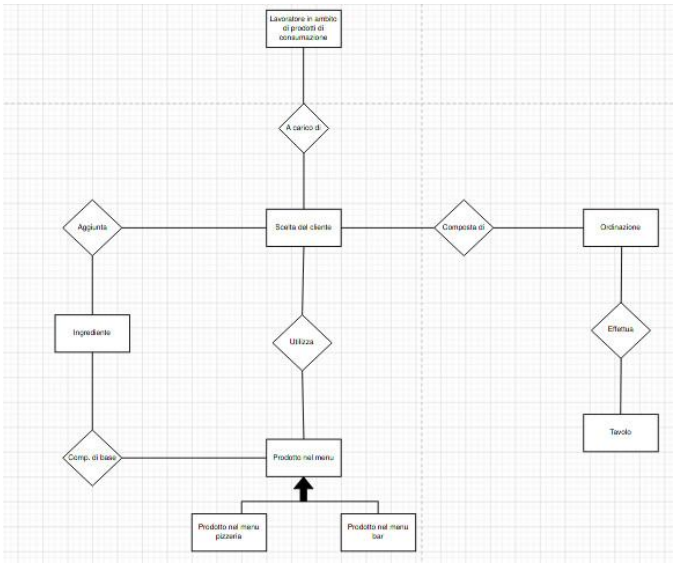
1) Identificazione dei concetti di base



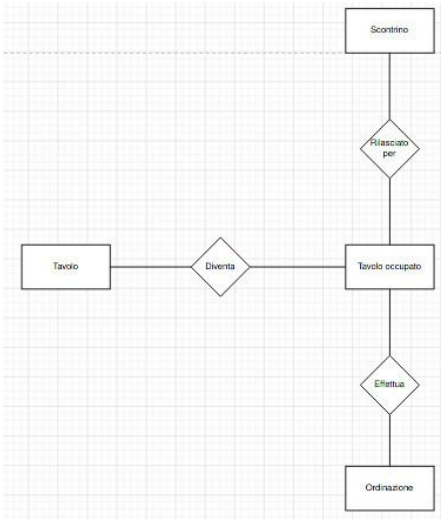
2) Raffinazione dei concetti

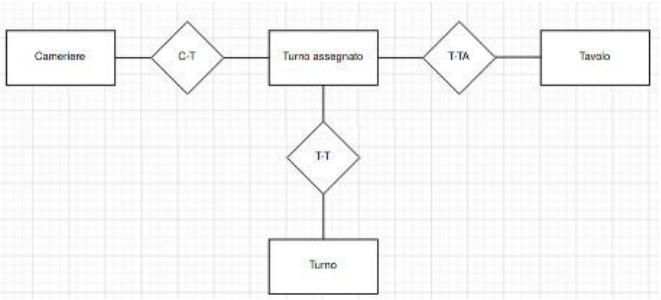






3) Applicazione di design pattern (reificazione relazione ternaria e instance-of)





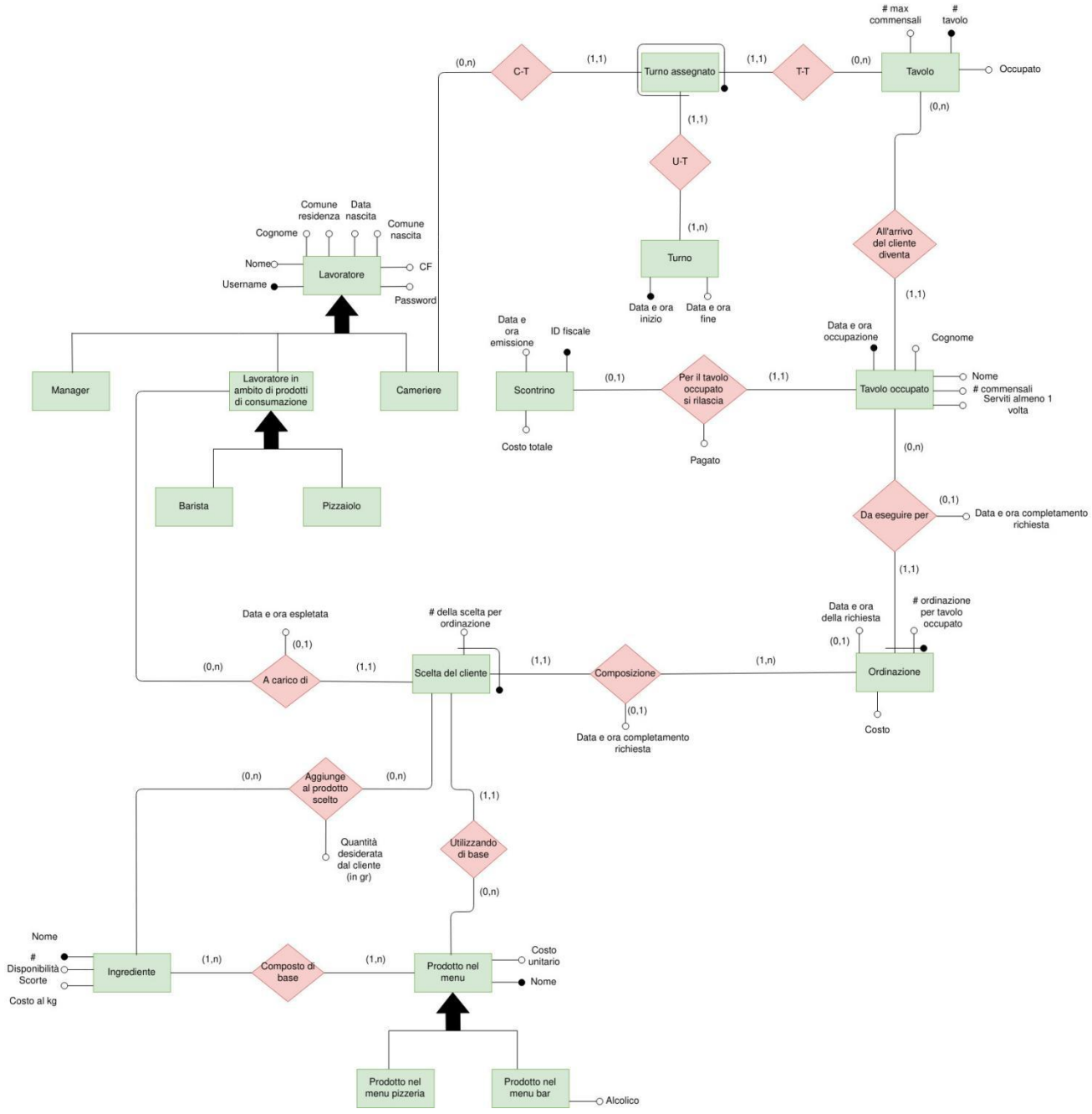
4) Merge dei pezzi risultanti

Si uniscono i pezzi dello schema

5) Aggiunta di attributi, cardinalità, identificatori esterni e identificatori primari

In questa fase si aggiungono gli attributi, le cardinalità, identificatori esterni e identificatori primari

Integrazione finale



Regole aziendali

- RA1: Un tavolo è considerato occupato al momento se tutte le seguenti condizioni sono soddisfatte:
  - E' presente una entry tra "Tavolo occupato" e "Tavolo"
  - Il "Tavolo" è attivo nel turno corrente
  - Lo scontrino ancora non è stato rilasciato **oppure** è stato rilasciato ma ancora non è

stato pagato

- RA2: Data e ora completamento richiesta (attributo della relazione "Da eseguire per") assume:
  - NESSUN VALORE: L'ordinazione ancora non è stata completamente consegnata (rimangono delle "Scelte del cliente", che fanno parte dell'ordinazione, da consegnare)
  - VALORE data e ora: L'ordinazione è stata completamente consegnata (corrisponde con l'ultima "Scelta del cliente" consegnata per l'ordinazione)
- RA3: Data e ora completamento richiesta (attributo della relazione "Composizione") assume:
  - NESSUN VALORE: La singola scelta del cliente ancora non è stata consegnata
  - VALORE data e ora: La singola scelta del cliente è stata consegnata
- RA4: Data e ora espletata (attributo della relazione "A carico di") assume:
  - NESSUN VALORE: La singola scelta ancora non è stata espletata
  - VALORE data e ora: La singola scelta è stata espletata
- RA5: Data e ora della richiesta (attributo della relazione "Ordinazione") assume:
  - NESSUN VALORE: il cameriere ancora sta prendendo l'ordinazione, non è pertanto visibile ai pizzaioli e barman
  - VALORE data e ora: l'ordinazione è stata presa, visibile e in attesa di presa in carico da parte dei pizzaioli e barman
- RA6: I valori opzionali possono assumere un valore sotto le seguenti condizioni:
  - Data e ora della richiesta ("Ordinazione"): l'ordinazione deve essere necessariamente chiusa
  - Data e ora espletata ("A carico di"):
    - Data e ora della richiesta ("Ordinazione") deve avere un valore
  - Data e ora completamento richiesta ("Composizione"):
    - Data e ora della richiesta ("Ordinazione") deve avere un valore
    - Data e ora espletata ("A carico di") deve avere un valore
  - Data e ora completamento richiesta ("Da eseguire per"):
    - Data e ora completamento richiesta ("Composizione") deve avere un valore in tutte le entry (per la particolare Ordinazione).
    - Data e ora della richiesta ("Ordinazione") deve avere un valore
    - Data e ora espletata ("A carico di") deve avere un valore
- RA7: La scelta del cliente è visibile solo ai barman o ai pizzaioli in base al prodotto nel menu (in particolare all'attributo IsBarMenu)

- RA8: L'attributo "Pagato" nella relazione "Per il tavolo occupato si rilascia" è di default impostato a false, settabile a true se e solo se lo scontrino viene rilasciato (cioè uno Scontrino partecipa alla relazione)
- RA9: # ordinazione per tavolo occupato ("Ordinazione") viene incrementata da 1 .. n per ogni "Tavolo occupato"
- RA10: # scelta per ordinazione ("Scelta del cliente") viene incrementata da 1 .. n per ogni "Ordinazione"
- RA11: ogni cameriere vede solo i tavoli di sua competenza per il turno attuale
- RA12: alla prima ordinazione servita, per il particolare "Tavolo occupato" si setta "Serviti almeno una volta" pari a true
- RA13: Un tavolo precedentemente occupato viene indicato come libero quando:
  - Tutte le ordinazioni sono state servite (di conseguenza il tavolo è stato servito almeno una volta)
  - Lo scontrino è stato rilasciato
  - Pagato è impostato su true
- RA14: Un tavolo viene selezionato per l'assegnazione a un cliente quando:
  - Soddisfa: # commensali <= # max commensali
  - E' attivo nel turno corrente (cioè assegnato ad un cameriere)
  - Tavolo.Occupato = false

Nota: NESSUN VALORE è presente perchè per il modello ER possiamo avere attributi opzionali
Nota1: VALORE indica la presenza del valore

Regole di derivazione

- RD1: Il costo di una scelta del cliente si ricava dalla seguente formula:
  - $CostoSdc(i) = Sdc_i.UtilizzandoDiBase.ProdottoMenu.CostoUnitario + \sum cvt(Sdc_i.Agg.Ing.Extra.QuantitaInGr, |Sdc_i.Agg.Ing.Extra.Ingred$
    ,
- RD2: Il costo di una ordinazione si ricava dalla seguente formula:
  - $CostoOrd(k) = \sum_{j=1}^{SdcPerOrd(k)} CostoSdc(j)$
- RD3: Il costo totale si ricava dalla seguente formula:

- $CostoTotale(t) = \sum_{n=1}^{OrdPerTavolo(t)} CostoOrd(n)$

Nota2: cvt(x,y) calcola il costo in base alla quantita in gr e al costo per kg.

Dizionario dei dati

Entità	Descrizione	Attributi	Identificatori
Lavoratore	Rappresenta tutti i lavoratori del ristorante in maniera generica	Username, Nome, Cognome, Comune residenza, Data nascita, Comune nascita, CF, Password	Username
Manager	Rappresenta i manager del ristorante	Username, Nome, Cognome, Comune residenza, Data nascita, Comune nascita, CF, Password	Username
Cameriere	Rappresenta i camerieri del ristorante	Username, Nome, Cognome, Comune residenza, Data nascita, Comune nascita, CF, Password	Username
Lavoratore in ambito di prodotti di consumazione	Rappresenta i lavoratori che preparano le ordinazioni in maniera più generica (in quanto entrambi si fanno carico delle scelte del cliente)	Username, Nome, Cognome, Comune residenza, Data nascita, Comune nascita, CF, Password	Username
Barista	Il barista si fa carico della preparazione di prodotti del bar	Username, Nome, Cognome, Comune residenza, Data nascita, Comune nascita, CF, Password	Username
Pizzaiolo	Il pizzaiolo si fa carico dei prodotti della pizzeria	Username, Nome, Cognome, Comune residenza, Data nascita, Comune nascita, CF, Password	Username

Turno assegnato	Un particolare turno assegnato dal manager al cameriere, coprendo uno o più tavoli		# tavolo, Username, Data e ora inizio
Turno	Rappresenta un turno con data e ora di inizio e fine	Data e ora inizio, Data e ora fine	Data e ora inizio
Tavolo	Tavolo del ristorante che può essere occupato da almeno una persona più un numero max di commensali	# max commensali, # tavolo, Occupato	# tavolo
Tavolo occupato	Rappresenta una particolare istanza di tavolo che viene occupato (non necessita di identificazione esterna)	Data e ora occupazione, Cognome, Nome, # commensali, Servito almeno una volta	Data e ora occupazione
Scontrino	Rappresenta uno scontrino emesso associato a un particolare tavolo occupato.	ID fiscale, Data e ora emissione, Costo totale	ID fiscale
Ordinazione	Rappresenta un ordinazione complessiva per tavolo, composta di più scelte.	# ordinazione per tavolo occupato, Data e ora della richiesta, Costo	# ordinazione per tavolo occupato, Data e ora occupazione
Scelta del cliente	La singola scelta che fa parte dell'ordinazione	# scelta per ordinazione	# scelta per ordinazione, # ordinazione per tavolo occupato, Data e ora occupazione
Prodotto nel menu	Rappresenta un prodotto nel menu. Tutte le istanze di questa entità formano il menu	Nome, Costo unitario	Nome
Ingrediente	Rappresenta un ingrediente che (di base) compone il prodotto oppure che viene aggiunto alla scelta particolare del cliente.	Nome, # disponibilità scorte, Costo al kg	Nome
Prodotto nel menu pizzeria	Particolare prodotto nel menu	Nome, Costo unitario	Nome
Prodotto nel menu bar	Particolare prodotto nel menu	Nome, Costo unitario, Alcolico	Nome

4. Progettazione logica

Volume dei dati

Concetto nello schema	Tipo[1]	Volume atteso
Lavoratore	E	18
Manager	E	2
Lavoratore in ambito di prodotti di consumazione	E	6
Barista	E	3
Pizzaiolo	E	3
Cameriere	E	10
Tavolo	E	20
Tavolo occupato	E	10000
Scontrino	E	10000
Ordinazione	E	15000
Scelta del cliente	E	60000
Prodotto nel menu	E	20
Ingrediente	E	100
Turno	E	3360
Turno assegnato	E	33600
C-T	R	3360
U-T	R	3360
T-T	R	3360
All'arrivo del cliente diventa	R	10000
Per il tavolo occupato si rilascia	R	10000
Da eseguire per	R	15000
Composizione	R	60000
A carico di	R	60000
Utilizzando di base	R	60000
Composto di base	R	80
Aggiunge al prodotto scelto	R	30000

Tavola delle operazioni

Cod.	Descrizione	Frequenza attesa
1	Aggiunta di un nuovo prodotto nel menu **(inizio op. manager)**	20/mese
2	Creazione di un nuovo ingrediente	100/anno
3	Assegnazione di ingrediente di base al prodotto gia aggiunto nel menu	80/mese
4	Rimozione di prodotti nel menu	5/mese
5	Rimozione di ingredienti	10/anno
6	Aggiunta di un nuovo tavolo	15/anno
7	Contrassegna scontrino stampato come	20/giorno

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

	pagato	
8	Assegnazione tavolo ad un cliente	20/giorno
9	Rilascio scontrino per tavolo occupato	20/giorno
10	Visualizza tavoli con possibilità di stampare scontrino	20/giorno
11	Visualizza scontrini stampati ma non pagati	1/ora
12	Visualizza statistiche giornaliere	1/giorno
13	Visualizza statistiche mensili	1/mese
14	Creazione di un nuovo utente lavoratore	4/anno
15	Ripristino password per utente	2/anno
16	Creazione turno	6/sett
17	Assegnazione turno e tavolo a cameriere	35/sett
18	Incremento disponibilità ingrediente	75/sett
19	Visualizza disponibilità ingredienti	1/sett
20	Rimozione associazione prodotto e ingredidente	1/mese
21	Visualizza situazione complessiva tavoli	3/giorno
22	Visualizza utenti lavoratori nel sistema	1/mese
23	Visualizza menu	1/sett
24	Visualizza prodotti assoc. Ingredienti di base	1/sett
25	Visualizza turni (>= ora)	1/giorno
26	Visualizza turni assegnati (>= ora)	1/giorno
27	Visualizza turno attuale e assegnazioni **(fine op. manager)**	3/giorno
28	Prendi ordinazione **(inizio op. cameriere)**	60/giorno
29	Prendi scelta per ordinazione	240/giorno
30	Aggiungi ingrediente extra alla scelta	60/giorno
31	Chiudi ordinazione	60/giorno
32	Visualizza situazione complessiva tavoli assegnati	30/giorno
33	Visualizza tavoli assegnati per il turno corrente (TUTTI)	10/giorno
34	Visualizza scelte delle ordinazioni espletate	30/giorno
35	Consegna scelta dell'ordinazione espletata **(fine op. cameriere)**	240/giorno
36	Visualizza scelta dell'ordinazione da preparare non ancora presa in carico **(inizio op. pizzaiolo/barman)**	60/giorno
37	Prendi in carico scelta da preparare	240/giorno
38	Visualizza scelta presa in carico ancora da espletare	180/giorno
39	Visualizza maggiori informazioni sulle scelte prese in carico	240/giorno
40	Espleta scelta **(fine op. pizzaiolo/barman)**	240/giorno
41	Login nel sistema **(op. login)**	15/giorno

Costo delle operazioni

Op.	Accesso	Tipo	Costo
1	1 ProdottoNelMenu	S	2 * 1 * 20 = 40 accessi / mese
2	1 Ingrediente	S	2 *1 * 100 = 200 accessi / anno
3	1 Composto di base	S	2 * 1 * 80 = 160 accessi / mese
4	1 ProdottoNelMenu	S	2 * 1 * 5 = 10 accessi / mese
5	1 Ingrediente	S	2 * 1 * 10 = 20 accessi / anno
6	1 Tavolo	S	2 * 1 * 15 = 30 accessi / anno
7	1 Scontrino	S	2 * 1 * 20 = 40 accessi / giorno
8	10 Tavolo 1 Tavolo 1 All'arrivo del cliente diventa 1 TavoloOccupato	L S S S	(10 + 2 * 1 + 2 * 1 + 2 * 1) * 20 = 320 accessi / giorno
9	1 TavoloOccupato 2 Da eseguire per 2 Ordinazione 1 Per il tavolo occupato si rilascia 1 Scontrino	L L L S S	(1 + 2 + 2 + 2* 1 + 2 * 1) * 20 = 180 accessi / giorno
10	10 TavoloOccupato 20 Da eseguire per 20 Ordinazione	L L L	(10 + 20 + 20) * 20 = 100 accessi / giorno
11	5 Scontrino	L	5 * 1 = 5 accessi / ora
12	15 Scontrino	L	15 * 1 = 15 accessi / giorno
13	560 Scontrino	L	560 * 1 = 560 accessi / mese
14	1 UtenteLavoratore	S	2 * 1 * 4 = 8 accessi / anno
15	1 UtenteLavoratore	S	2 * 1 * 2 = 4 accessi / anno
16	1 Turno	S	2 * 1 * 6 = 12 accessi / sett
17	1 Turno 1 C-T 1 UtenteLavoratore 1 U-T 1 Tavolo 1 T-T 1 TurnoAssegnato	L S L S L S S	(1 + 2 * 1 + 1 + 2 * 1 + 1 + 2 * 1 + 2 * 1) * 35 = 385 accessi / sett
18	1 Ingrediente	S	2 * 1 * 75 = 150 accessi / sett
19	100 Ingrediente	L	100 * 1 = 100 accessi / sett
20	1 Composto di base	S	2 * 1 = 2 accessi / mese
21	15 Tavolo 10 All'arrivo del cliente diventa 10 TavoloOccupato	L L L	(15 + 10 + 10) * 3 = 105 accessi / giorno
22	18 Lavoratore	L	18 * 1 = 18 accessi / mese
23	20 ProdottoNelMenu	L	20 * 1 = 20 accessi / sett
24	40 Composto di base	L	40 * 1 = 40 accessi / sett
25	4 Turno	L	4 * 1 = 4 accessi / giorno
26	3 Turno 3 U-T 3 TurnoAssegnato 3 T-T	L L L L	(3 + 3 + 3 + 3 + 18 + 10 + 10) * 1 = 50 accessi / giorno

	18 Tavolo 10 C-T 10 UtenteLavoratore	L L L	
27	1 Turno 1 U-T 1 TurnoAssegnato 1 T-T 10 Tavolo 5 C-T 5 UtenteLavoratore	L L L L L L L	(1 + 1 + 1 + 1 + 10 + 5 + 5) * 3 = 72 accessi / giorno
28	1 TavoloOccupato 1 Per il tavolo occupato si rilascia 1 Scontrino 1 Da eseguire per 1 Ordinazione	L L L S S	(1 + 1 + 1 + 2 * 1 + 2 * 1) * 60 = 420 accessi / giorno
29	1 TavoloOccupato 1 Da eseguire per 1 Ordinazione 1 Composizione 1 SceltaDelCliente 1 Utilizzando di base 1 ProdottoNelMenu 1 Composto di base 1 Ingrediente 1 Ingrediente	L L L S S L L L L S	(1 + 1 + 1 + 2 * 1 + 2 * 1 + 1 + 1 + 1 + 1 + 2 * 1) * 240 = 3120 accessi / giorno
30	1 TavoloOccupato 1 Da eseguire per 1 Ordinazione 1 Composizione 1 SceltaDelCliente 1 Aggiunge al prodotto scelto 1 Ingrediente 1 Ingrediente	L L L L L S L S	(1 + 1 + 1 + 1 + 1 + 2 * 1 + 1 + 2 * 1) * 60 = 600 accessi / giorno
31	1 TavoloOccupato 1 Da eseguire per 1 Ordinazione	L L S	(1 + 1 + 2 * 1) * 60 = 240 accessi / giorno
32	1 Turno 1 U-T 1 C-T 1 T-T 1 TurnoAssegnato 1 UtenteLavoratore 2 Tavolo 1 All'arrivo del cliente diventa 1 TavoloOccupato 1 Per il tavolo occupato si rilascia	L L L L L L L L L L	(1 + 1 + 1 + 1 + 1 + 1 + 2 + 1 + 1 + 1) * 30 = 330 accessi / giorno
33	1 Turno 1 U-T 1 C-T 1 T-T	L L L L	(1 + 1 + 1 + 1 + 1 + 1 + 2) * 10 = 80 accessi / giorno

	1 TurnoAssegnato 1 UtenteLavoratore 2 Tavolo	L L L	
34	6 A carico di 2 SceltaDelCliente 2 Composizione 2 Ordinazione 2 Da eseguire per 2 TavoloOccupato	L L L L L L	(6 + 2 + 2 + 2 + 2 + 2) * 30 = 480 accessi / giorno
35	1 SceltaDelCliente 1 Composizione 1 Ordinazione 1 Da eseguire per 1 TavoloOccupato	S S S S S	(2 * 1 + 2 * 1 + 2 * 1 + 2 * 1 + 2 * 1) * 240 = 2400 accessi / giorno
36	5 A carico di 5 SceltaDelCliente 5 Utilizzando di base 5 ProdottoNelMenu	L L L L	(5 + 5 + 5 + 5) * 60 = 1200 accessi / giorno
37	1 A carico di 1 SceltaDelCliente 1 Utilizzando di base 1 ProdottoNelMenu 1 A carico di	L L L L S	(1 + 1 + 1 + 1 + 2 * 1) * 240 = 1440 accessi / giorno
38	3 A carico di 3 SceltaDelCliente 3 Utilizzando di base 3 ProdottoNelMenu	L L L L	(3 + 3 + 3 + 3) * 180 = 2160 accessi / giorno
39	3 A carico di 3 SceltaDelCliente 3 Utilizzando di base 3 ProdottoNelMenu 6 Aggiunge al prodotto scelto 6 Ingrediente	L L L L L L	(3 + 3 + 3 + 3 + 6 + 6) * 240 = 5760 accessi / giorno
40	1 SceltaDelCliente 1 A carico di	S S	(2 * 1 + 2 * 1) * 240 = 960 accessi / giorno
41	1 UtenteLavoratore	L	1 * 15 = 15 accessi / giorno

Ristrutturazione dello schema E-R

Nello schema concettuale sono presenti le seguenti ridondanze:

- Costo totale in Scontrino (che è possibile ricavare sempre dalle ordinazioni effettuate per tavolo occupato)
- Costo in Ordinazione (che è possibile ricavare dalle associazioni di scelta del cliente - cioè per ogni scelta: dal costo unitario del prodotto nel menu più gli ingredienti extra rapportati alla quantità in grammi)

- Occupato in Tavolo: Lo stato di occupazione del tavolo può essere ricavato dal check di:
  - Attivazione del tavolo nel turno corrente
  - Presenza di associazione di tavolo occupato e tavolo stesso
  - Pagamento finale scontrino (rilasciato ma non pagato)

Data la complessità delle operazioni da effettuare per ricavare gli attributi (e dal volume dei dati), è risultato opportuno fornire il dato ridondante già "pronto".

Sono state eliminate 3 generalizzazioni:

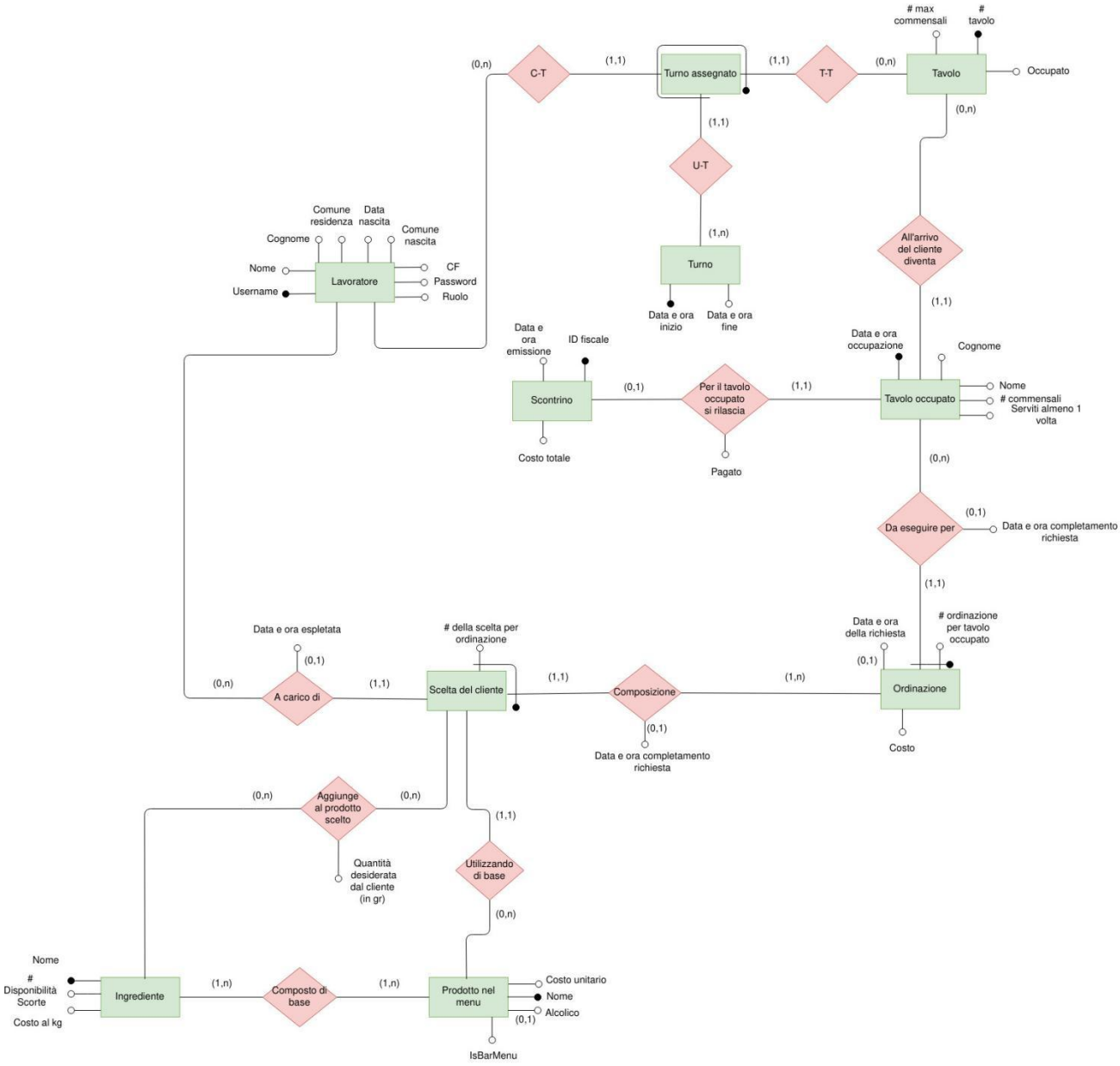
- Le entità Manager, Lavoratore in ambito di prodotti di consumazione (a loro volta Barista e Pizzaiolo), Cameriere, sono presenti per identificare concettualmente le diverse figure che interagiscono col sistema e scendendo più in dettagli di "autenticazione" è risultato opportuno utilizzare un unica entità "Lavoratore" (utenti del sistema) con un attributo Ruolo che indentifica il tipo di lavoratore e le possibilità di utilizzo delle funzionalità del sistema.
- Le entità Prodotto nel menu pizzeria e Prodotto nel menu bar identificano i prodotti offerti: condividono praticamente gli stessi attributi, solo che "Prodotto nel menu bar" aggiunge anche "IsAlcolico" per segnalare un prodotto alcolico. E' risultato opportuno incorporare il tutto in "Prodotto nel menu", aggiungendo "IsBarMenu" (booleano), per indicare se il prodotto fa parte del menu del bar o della pizzeria e un altro attributo che indica se il prodotto è alcolico o meno (IsAlcolico). Opzionale: se il prodotto fa parte del menu pizzeria allora non può essere alcolico o meno (non avrebbe senso)

Identificatori primari:

- Lavoratore: Username - è sufficiente per identificare ogni singolo lavoratore col proprio username
- Turno Assegnato: identificazione esterna (Username, Data e ora inizio, # tavolo) - attributi minimi necessari all'identificazione di un particolare turno assegnato a un particolare lavoratore per un determinato tavolo - in questo modo è possibile avere:
  - Più lavoratori in un singolo turno
  - Più tavoli assegnabili a più lavoratori
- Turno: E' sufficiente sapere la data e l'ora d'inizio del turno - inserire nella chiave primaria anche data e ora fine significa permettere che per una stessa data e ora inizio possiamo avere più date e ore fine diverse (oppure per la stessa data e ora fine, diverse date e ore inizio

diverse)

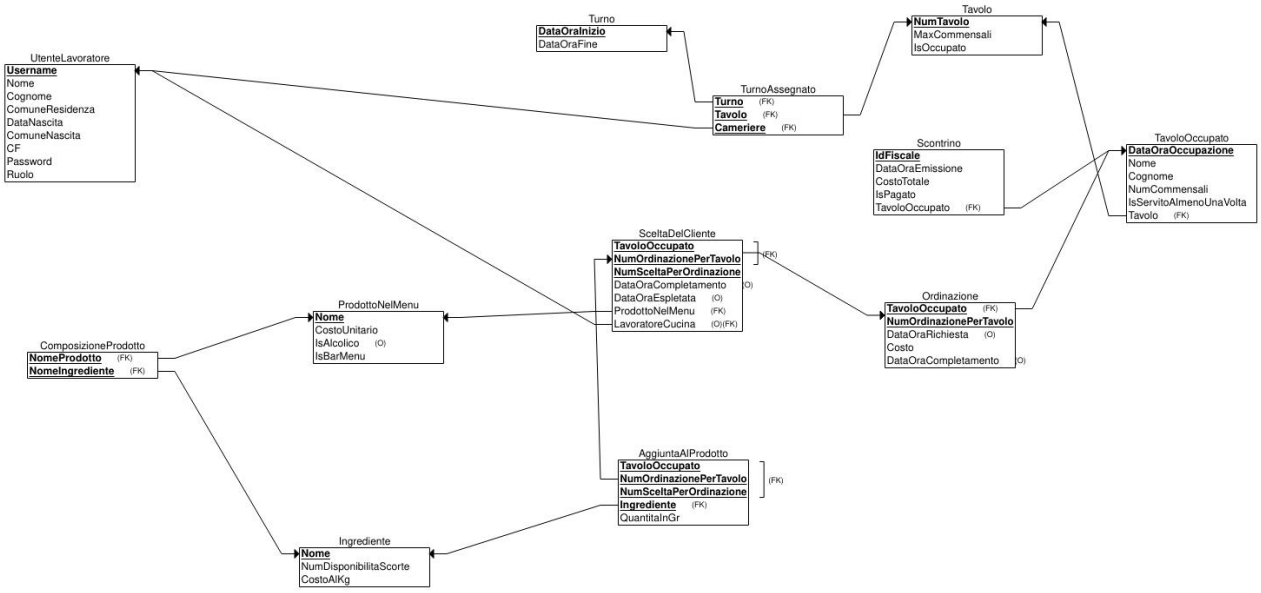
- Tavolo: # tavolo è sufficiente a identificare un tavolo nel locale
- Tavolo occupato: a una data e ora occupazione viene occupato solo un tavolo in particolare (a cui possiamo risalire grazie all'associazione)
- Scontrino: ID fiscale identifica univocamente uno scontrino
- Ordinazione: Il numero dell'ordinazione (con data e ora occupazione di tavolo occupato) identifica la particolare ordinazione per il particolare tavolo occupato
- Scelta del cliente: Il numero della scelta del cliente per ordinazione, del numero dell'ordinazione per tavolo occupato e della data e ora occupazione del tavolo identificano univocamente la particolare scelta del cliente
- Prodotto nel menu: il nome lo identifica univocamente
- Ingrediente: il nome lo identifica univocamente



Trasformazione di attributi e identificatori

Non viene applicata alcuna trasformazione particolare

Traduzione di entità e associazioni



Normalizzazione del modello relazionale

Lo schema si presenta già in 3NF

5. Progettazione fisica

Utenti e privilegi

L'assegnazione di privilegi agli utenti è stata effettuata seguendo il PoLP (Principle of Least Privileges). Per garantire la sicurezza e l'integrità dei dati, è infatti sufficiente per gli utenti eseguire le stored procedure che contengono la logica per la memorizzazione dei dati. Ogni altro tipo di accesso (select, update, delete, ...) alle tabelle base di dati da parte di qualsiasi utente verrà negato (è di fatto soltanto consentito chiamare le stored procedures tali che l'utente ha un grant execute su di esse).

- Manager: l'utente più importante del sistema, ricopre mansioni di amministrazione del sistema e del locale.
- Cameriere: l'utente rappresenta un cameriere che prende le ordinazioni dai clienti e le consegna una volta espletate.
- Barman e pizzaiolo: gli utenti sono praticamente identici, di fatto hanno gli stessi privilegi "execute" sulle stesse stored procedures. La differenza tra i due è i tipi di prodotti su cui possono lavorare: alcuni sono del bar, altri della pizzeria (rispettivamente destinati a barman e al pizzaiolo). La differenza è resa possibile grazie alla logica nelle stored procedures che verifica attributi degli utenti e dei prodotti (cioè una richiesta di manipolare un prodotto del bar da parte di un pizzaiolo e quella di manipolare un prodotto della pizzeria da parte del barman verranno negate).

La seguente tabella riporta i privilegi concessi e su quali risorse della BD:

Utente	Tipologia di permesso	Risorse

Manager	Execute	RegistraUtente,RipristinoPassword,AggiungiNuovoTavolo,AggiungiNuovoIngrediente,AggiungiProdottoNelMenu,AssociaProdottoAIngrediente,AggiungiTurno,RimuoviProdottoNelMenu,RimuoviIngrediente,RimuoviAssocProdottoEIngrediente,OttieniTurni,OttieniUtenti,OttieniTavoli,AssegnaTurno,OttieniTurnoAttuale,OttieniTurniAssegnati,OttieniMenu,OttieniComposizioneProdotto,OttieniIngredienti,IncDispIngrediente,OttieniEntrate,OttieniScontriniNonPagati,ContrassegnaScontrinoPagato,AssegnaTavoloACliente,OttieniTavoliScontriniStampabili,StampaScontrino
Pizzaiolo	Execute	OttieniScelteDaPreparare,PrendiInCaricoScelta,OttieniSceltePreseInCaricoNonEspletate,EspletaSceltaPresaInCarico,OttieniInfoProdottiDiScelteInCarico
Barman	Execute	OttieniScelteDaPreparare,PrendiInCaricoScelta,OttieniSceltePreseInCaricoNonEspletate,EspletaSceltaPresaInCarico,OttieniInfoProdottiDiScelteInCarico
Cameriere	Execute	OttieniTavoliDiCompetenza,PrendiOrdinazione,ChiudiOrdinazione,PrendiSceltaPerOrd,OttieniSceltePerOrdinazione,AggiungiIngExtraAllaScelta,OttieniScelteEspletate,EffettuaConsegna,OttieniTavoliAssegnati
Login	Execute	TentaLogin

Strutture di memorizzazione

Tabella UtenteLavoratore		
Attributo	**Tipo di dato**	**Attributi[2]**

[2] PK = primary key, NN = not null, UQ = unique, UN = unsigned, AI = auto increment. È ovviamente possibile specificare più di un attributo per ciascuna colonna.

Username	varchar(10)	PK
Nome	varchar(20)	NN
Cognome	varchar(20)	NN
ComuneResidenza	varchar(34)	NN
DataNascita	date	NN
ComuneNascita	varchar(34)	NN
CF	char(16)	NN, UQ
Passwd	char(40)	NN
Ruolo	tinyint	NN

Tabella Turno		
Attributo	**Tipo di dato**	**Attributi**
DataOraInizio	datetime	PK
DataOraFine	datetime	NN, UQ

Tabella Tavolo		
Attributo	**Tipo di dato**	**Attributi**
NumTavolo	smallint	PK
MaxCommensali	tinyint	NN, UN
IsOccupato	boolean	NN

Tabella TurnoAssegnato		
Attributo	**Tipo di dato**	**Attributi**
Turno	datetime	PK
Cameriere	varchar(10)	PK
Tavolo	smallint	PK

Tabella Scontrino		
Attributo	**Tipo di dato**	**Attributi**
IdFiscale	int	PK, AI
DataOraEmissione	datetime	NN, UQ
CostoTotale	float	NN
IsPagato	boolean	NN
TavoloOccupato	datetime	

Tabella TavoloOccupato		
Attributo	**Tipo di dato**	**Attributi**
DataOraOccupazione	datetime	PK
Nome	varchar(20)	NN
Cognome	varchar(20)	NN
NumCommensali	tinyint	NN, UN
IsServitoAlmenoUnaVolta	boolean	NN
Tavolo	smallint	NN

Tabella Ordinazione		
Attributo	**Tipo di dato**	**Attributi**
TavoloOccupato	datetime	PK
NumOrdinazionePerTavolo	tinyint	PK
DataOraRichiesta	datetime	
Costo	float	NN, UN
DataOraCompletamento	datetime	

Tabella ProdottoNelMenu		
Attributo	**Tipo di dato**	**Attributi**
Nome	varchar(20)	PK
CostoUnitario	float	NN
IsAlcolico	boolean	
IsBarMenu	boolean	NN

Tabella Ingrediente		
Attributo	**Tipo di dato**	**Attributi**
Nome	varchar(20)	PK
NumDisponibilitaScorte	int	NN, UN
CostoAlKg	float	NN

Tabella ComposizioneProdotto		
Attributo	**Tipo di dato**	**Attributi**
NomeProdotto	varchar(20)	PK
NomeIngrediente	varchar(20)	PK

Tabella SceltaDelCliente		
Attributo	**Tipo di dato**	**Attributi**
TavoloOccupato	datetime	PK
NumOrdinazionePerTavolo	tinyint	PK
NumSceltaPerOrdinazione	tinyint	PK
DataOraCompletamento	datetime	
DataOraEspletata	datetime	
ProdottoNelMenu	varchar(20)	
LavoratoreCucina	varchar(10)	

Tabella AggiuntaAlProdotto		
Attributo	**Tipo di dato**	**Attributi**
TavoloOccupato	datetime	PK
NumOrdinazionePerTavolo	tinyint	PK
NumSceltaPerOrdinazione	tinyint	PK
Ingrediente	varchar(20)	PK
QuantitaInGr	float	NN

Indici

Tabella ComposizioneProdotto	
Indice ComposizioneProdotto_NomeProdotto_fk	**Tipo[3]:**
NomeProdotto	IDX
Indice ComposizioneProdotto_NomeIngrediente_fk	**Tipo:**
NomeIngrediente	IDX
Indice PRIMARY	**Tipo:**
NomeProdotto, NomeIngrediente	PR

Tabella TurnoAssegnato	
Indice TurnoAssegnato_Turno_fk	**Tipo:**
Turno	IDX
Indice TurnoAssegnato_Tavolo_fk	**Tipo:**
Tavolo	IDX
Indice TurnoAssegnato_Cameriere_fk	**Tipo:**
Cameriere	IDX
Indice PRIMARY	**Tipo:**
Turno, Tavolo, Cameriere	PR

Tabella TavoloOccupato	
Indice TavoloOccupato_Tavolo_fk	**Tipo:**
Tavolo	IDX
Indice PRIMARY	**Tipo:**
DataOraOccupazione	PR

Tabella Scontrino	
Indice Scontrino_TavoloOccupato_fk	**Tipo:**
TavoloOccupato	IDX
Indice PRIMARY	**Tipo:**
DataOraEmissione	PR

Tabella Ordinazione	
Indice Ordinazione_TavoloOccupato_fk	**Tipo:**
TavoloOccupato	IDX
Indice PRIMARY	**Tipo:**
TavoloOccupato, NumOrdinazionePerTavolo	PR

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

Tabella SceltaDelCliente	
Indice SceltaDelCliente_ProdottoNelMenu_fk	**Tipo:**
ProdottoNelMenu	IDX
Indice SceltaDelCliente_LavoratoreCucina_fk	**Tipo:**
LavoratoreCucina	IDX
Indice SceltaDelCliente_TavoloOccupato_NumOrdinazionePerTavolo_fk	**Tipo:**
TavoloOccupato, NumOrdinazionePerTavolo	IDX
Indice PRIMARY	**Tipo:**
TavoloOccupato, NumOrdinazionePerTavolo, NumSceltaPerOrdinazione	PR

Tabella AggiuntaAlProdotto	
Indice AggiuntaAlProdotto_Ingrediente_fk	**Tipo:**
Ingrediente	IDX
Indice AggiuntaAlProdotto_TavoloOcc_NumOrdPerTa_NumSceltaPerOrd_fk	**Tipo:**
TavoloOccupato, NumOrdinazionePerTavolo, NumSceltaPerOrdinazione	IDX
Indice PRIMARY	**Tipo:**
TavoloOccupato, NumOrdinazionePerTavolo, NumSceltaPerOrdinazione, Ingrediente	PR

Tabella UtenteLavoratore	
Indice PRIMARY	**Tipo:**
Username	PR

Tabella Tavolo	
Indice PRIMARY	**Tipo:**
NumTavolo	PR

Tabella Turno	
Indice PRIMARY	**Tipo:**
DataOraInizio	PR

Tabella Ingrediente	
Indice PRIMARY	**Tipo:**
Nome	PR

Tabella ProdottoNelMenu

Indice PRIMARY	**Tipo:**
Nome	PR

Trigger

- Questo trigger serve a verificare che il turno che sta per essere inserito sia programmato a partire dall'istante attuale (non è quindi possibile creare un turno "nel passato")

```
create trigger TurnoCheckDataOraInizio_Insert
before insert on Turno for each row
begin
        if NEW.DataOraInizio < now() then
                signal sqlstate '45001'
                        set message_text = "Impossibile creare turno per il passato";
        end if;
end!
```

- Questo trigger serve a verificare che il turno che sta per essere inserito non si sovrapponga con altri turni

```
create trigger TurnoCheckOverlap_Insert
before insert on Turno for each row
begin
        declare overlapCount int;

        select
                count(*)
        into
                overlapCount
        from
                Turno
        where
                DataOraInizio <= NEW.DataOraFine and DataOraFine >=
NEW.DataOraInizio;
```

```
        if overlapCount > 0 then
            signal sqlstate '45003'
                set message_text = "Overlap dei turni";
        end if;
    end!
```

- Questo trigger serve a verificare che il turno sia assegnato effettivamente a un cameriere e non a un altro tipo di lavoratore

```
    create trigger TurnoAssegnatoCheckIsCameriere_Insert
    before insert on TurnoAssegnato for each row
    begin
        declare newUserRole tinyint;

        select
            Ruolo
        into
            newUserRole
        from
            UtenteLavoratore
        where
            Username = NEW.Cameriere;

        if newUserRole <> 2 then
            signal sqlstate '45002'
                set message_text = "Impossibile assegnare un turno/tavolo a un non-cameriere";
        end if;
    end!
```

- Questo trigger serve a verificare che lo scontrino che sta per essere inserito sia stato rilasciato per un tavolo occupato che è stato servito almeno una volta

```
create trigger ScontrinoCheckServizio_Insert
before insert on Scontrino for each row
begin
	declare isServito boolean;

	select
		IsServitoAlmenoUnaVolta
	into
		isServito
	from
		TavoloOccupato T
	where
		T.DataOraOccupazione = NEW.TavoloOccupato;

	if isServito = false then
		signal sqlstate '45004'
			set message_text = "il tavolo deve essere servito almeno una volta";
	end if;
end!
```

- Questo trigger serve a flaggare il tavolo come libero **dopo** *(after update)* che uno scontrino è stato pagato

```
create trigger ScontrinoCheckPagato_AfterUpdate
after update on Scontrino for each row
begin
	if NEW.IsPagato = true then
		update
			Tavolo
		set
			IsOccupato = false
		where
			NumTavolo = (
```

```
                    select
                        O.Tavolo
                    from
                        Scontrino S join TavoloOccupato O on
                            O.DataOraOccupazione = S.TavoloOccupato
                    where
                        S.TavoloOccupato = NEW.TavoloOccupato);
            end if;
        end!
```

- Questo trigger serve a verificare che una “scelta” sia stata espletata, prima di poter essere consegnata (di fatto, terminandone il “ciclo di vita”, quindi completata)

```
    create trigger SceltaDelClienteCheckDates_BeforeUpdate
    before update on SceltaDelCliente for each row
    begin
        if NEW.DataOraCompletamento is not NULL and
            NEW.DataOraEspletata is NULL then
            signal sqlstate '45007'
                set message_text = "Impossibile completare la scelta se prima non è
stata espletata";
        end if;
    end!
```

- Questo trigger impedisce che un ordinazione sia consegnata ancora prima che essa venga chiusa e quindi espletata (per chiusa si intende che viene accettata e “inviata” al pizzaiolo / barman)

```
    create trigger OrdinazioneCheckDates_BeforeUpdate
    before update on Ordinazione for each row
    begin
        if NEW.DataOraRichiesta is NULL and
            NEW.DataOraCompletamento is not NULL then
            signal sqlstate '45008'
```

```
                set message_text = "Impossibile completare l'ordinazione se prima non
è stata chiusa";
        end if;
    end!
```

- Questo trigger impedisce la creazione di nuove ordinazioni per lo stesso tavolo se:
  - Esiste un'ordinazione precedente non ancora completata (cioè in qualsiasi altro stato)
  - Lo scontrino per il tavolo è già stato stampato (ma non ancora pagato)

```
    create trigger OrdinazioneCheckAlreadyPendingOrScontrino_BeforeInsert
    before insert on Ordinazione for each row
    begin
        if exists(
            select
                *
            from
                Ordinazione
            where
                TavoloOccupato = NEW.TavoloOccupato and
                (DataOraRichiesta is NULL or DataOraCompletamento is NULL))
then

            signal sqlstate '45009'
                set message_text = "C'è un'ordinazione in attesa di completamento per
il tavolo scelto";
        end if;

        if exists(
            select
                *
            from
                Scontrino
            where
```

```
                    TavoloOccupato = NEW.TavoloOccupato) then

                signal sqlstate '45010'
                    set message_text = "Lo scontrino per questo tavolo è già stato
rilasciato";
            end if;

        end!
```

- Questo trigger serve a verificare la disponibilità di ingredienti su prodotto di base prima di effettuare una scelta su un ordinazione

```
    create trigger SceltaDelClienteCheckDisp_BeforeInsert
    before insert on SceltaDelCliente for each row
    begin
        declare nomeIng varchar(20);
        declare cntDisp int;
        declare shouldLeave boolean;
        declare cur1 cursor for
            select
                Nome, NumDisponibilitaScorte
            from
                ComposizioneProdotto join Ingrediente on
                    NomeIngrediente = Nome
            where
                NomeProdotto = NEW.ProdottoNelMenu;
        declare continue handler for not found set shouldLeave = true;

        set shouldLeave = false;

        open cur1;

        read_loop: loop
```

```
                fetch cur1 into nomeIng, cntDisp;

                if not shouldLeave then

                        if cntDisp = 0 then
                                leave read_loop;
                        end if;

                        update
                                Ingrediente
                        set
                                NumDisponibilitaScorte = NumDisponibilitaScorte - 1
                        where
                                Nome = nomeIng;

                else

                        leave read_loop;

                end if;

        end loop read_loop;

        close cur1;

        if cntDisp = 0 then
                signal sqlstate '45012'
                        set message_text = "Uno o più ingredienti del prodotto non sono
disponibili";
        end if;
    end!
```

- Questo trigger serve a sommare il costo complessivo di un ordinazione, dopo che una scelta per

essa è stata presa (inserita) con successo *(after insert)* in maniera incrementale

```
create trigger SceltaDelClienteAddCosto_AfterInsert
after insert on SceltaDelCliente for each row
begin
        update
                Ordinazione
        set
                Costo = Costo + (
                                                        select
                                                                CostoUnitario
                                                        from
                                                                ProdottoNelMenu
                                                        where
                                                                Nome = NEW.ProdottoNelMenu)
        where
                TavoloOccupato = NEW.TavoloOccupato and
                NumOrdinazionePerTavolo = NEW.NumOrdinazionePerTavolo;
end!
```

- Questo trigger serve a verificare la disponibilità di ingredienti per quanto riguarda le aggiunte extra ai prodotti di base

```
create trigger AggiuntaAlProdottoCheckDisp_BeforeInsert
before insert on AggiuntaAlProdotto for each row
begin
        declare numDispScorte int;

        select
                NumDisponibilitaScorte
        into
                numDispScorte
        from
```

```
            Ingrediente
        where
            Nome = NEW.Ingrediente;

        if numDispScorte = 0 then
            signal sqlstate '45013'
                set message_text = "L'ingrediente scelto non è disponibile";
        end if;

        update
            Ingrediente
        set
            NumDisponibilitaScorte = NumDisponibilitaScorte - 1
        where
            Nome = NEW.Ingrediente;
end!
```

- Questo trigger serve ad aggiungere il costo dell'aggiunta extra (tenendo conto di costo al kg e quantita richiesta) al prodotto di base, dopo che l'aggiunta extra è stata inserita con successo

```
create trigger AggiuntaAlProdottoAddCosto_AfterInsert
after insert on AggiuntaAlProdotto for each row
begin
    set @costoAlKg = (
        select
            CostoAlKg
        from
            Ingrediente
        where
            Nome = NEW.Ingrediente);

    update
        Ordinazione
```

```
        set
            Costo = Costo + ((NEW.QuantitaInGr / 1000) * @costoAlKg)
        where
            TavoloOccupato = NEW.TavoloOccupato and
            NumOrdinazionePerTavolo = NEW.NumOrdinazionePerTavolo;
    end!
```

- Questo trigger serve a verificare che tutte le scelte dell'ordinazione corrente sono state consegnate (ogni volta che se ne consegna una), se ciò è vero, flagga (per ogni # di ordinazione completata) il tavolo come servito almeno una volta e segnala l'ora attuale come la data e ora del completamento dell'ordinazione corrente (ordinazione completamente consegnata, cioè l'ordinazione è in stato finale, completata)

```
    create trigger SceltaDelClienteCheckTotalComplete_AfterUpdate
    after update on SceltaDelCliente for each row
    begin
        declare complDate datetime;
        declare shouldLeave boolean;
        declare cur1 cursor for select
                                    DataOraCompletamento
                                from
                                    SceltaDelCliente
                                where
                                    TavoloOccupato = NEW.TavoloOccupato and
                                    NumOrdinazionePerTavolo = NEW.NumOrdinazionePerTavolo;
        declare continue handler for not found set shouldLeave = true;

        set shouldLeave = false;

        open cur1;
```

```
read_loop: loop

        fetch cur1 into complDate;

        if not shouldLeave then

                if complDate is NULL then
                        leave read_loop;
                end if;

        else

                leave read_loop;

        end if;

end loop read_loop;

close cur1;

if complDate is not NULL then
        update
                Ordinazione
        set
                DataOraCompletamento = now()
        where
                TavoloOccupato = NEW.TavoloOccupato and
                NumOrdinazionePerTavolo = NEW.NumOrdinazionePerTavolo;

        update
                TavoloOccupato
        set
                IsServitoAlmenoUnaVolta = true
```

```
                where
                        DataOraOccupazione = NEW.TavoloOccupato;
        end if;
    end!
```

- Questo check in UtenteLavoratore forza l'assegnazione del ruolo in 4 possibili valori

```
    check (
                Ruolo in (1,2,3,4)
        )
```

- Questo check in Ingrediente forza il fatto che ciascun ingrediente non deve avere costo al kg pari a 0 (o negativo)

```
    check(
                CostoAlKg > 0
        )
```

- Questo check in ProdottoNelMenu forza il fatto che il costo del prodotto non deve avere costo pari a 0 (o negativo) e se IsBarMenu è false alora IsAlcolico deve essere NULL, al contrario se IsBarMenu è true allora IsAlcolico non deve essere NULL

```
    check(
                CostoUnitario > 0 and
                ((IsBarMenu and IsAlcolico is not NULL) or
                        (not IsBarMenu and IsAlcolico is NULL))
        )
```

- Questo check in Turno impedisce di inserire un turno invalido (la fine del turno prima dell'inizio del turno stesso)

```
    check(
                DataOraFine > DataOraInizio
        )
```

- Questo check in Scontrino impedisce di inserire uno scontrino con costo totale pari a 0 (o negativo)

```
    check(
                CostoTotale > 0
        )
```

- Questo check in Ordinazione forza la numerazione delle ordinazioni che va da 1, ..., n e che al

momento della chiusura dell'ordinazione, il costo sia maggiore di 0 (quindi, implicitamente, che l'ordinazione sia composta di almeno una scelta)

```
check(
        NumOrdinazionePerTavolo > 0 and
            (DataOraRichiesta is NULL or
                (DataOraRichiesta is not NULL and Costo > 0))
    )
```

- Questo check in SceltaDelCliente forza la numerazione delle scelte che va da 1, ..., n

```
check(
        NumSceltaPerOrdinazione > 0
    )
```

- Questo check in AggiuntaAlProdotto impedisce che la quantità in grammi di un ingrediente aggiuntivo sia pari a 0 (o negativo)

```
check(
        QuantitaInGr > 0
    )
```

Eventi

Non vi è stata necessità di implementare eventi

Viste

Non vi è stata necessità di implementare viste

Stored Procedures e transazioni

- Questa procedura è utilizzata dal thin client per effettuare il login nel sistema, usando l'utente "login"

```
create procedure TentaLogin(
        in usern varchar(10),
        in pwd varchar(45),
        out userRole tinyint)
begin
        set userRole = 0;
```

```
        set transaction read only;
        set transaction isolation level read committed;

        start transaction;

        select
                Ruolo
        into
                userRole
        from
                UtenteLavoratore
        where
                Username = usern and Passwd = SHA1(pwd);

        commit;
end!
```

- Questa procedura permette al manager di registrare nuovi utenti lavoratori

```
create procedure RegistraUtente(
        in username varchar(10),
        in nome varchar(20),
        in cognome varchar(20),
        in comuneResidenza varchar(34),
        in dataNascita date,
        in comuneNascita varchar(34),
        in cf char(16),
        in passwd varchar(45),
        in ruolo tinyint)
begin
        declare exit handler for sqlexception
        begin
                rollback;
```

```
            resignal;
        end;

        start transaction;

        insert into
            UtenteLavoratore(Username,
                                                Nome,
                                                Cognome,
                                                ComuneResidenza,
                                                DataNascita,
                                                ComuneNascita,
                                                CF,
                                                Passwd,
                                                Ruolo)
        values(
            username,
            nome,
            cognome,
            comuneResidenza,
            dataNascita,
            comuneNascita,
            cf,
            SHA1(passwd),
            ruolo
        );

        commit;
    end!
```

- Questa procedura consente al manager di effettuare un ripristino password se dovesse servire

```
    create procedure RipristinoPassword(
```

```
        in usern varchar(10),
        in pwd varchar(45))
begin
        set transaction isolation level read committed;

        start transaction;

        update
                UtenteLavoratore
        set
                Passwd = SHA1(pwd)
        where
                Username = usern;

        commit;
end!
```

- Questa procedura consente al manager di aggiungere un nuovo tavolo con numero di commensali (quindi per ogni tavolo vanno # persone >= 1)

```
create procedure AggiungiNuovoTavolo(
        in numt smallint,
        in maxcomm tinyint)
begin
        declare exit handler for sqlexception
        begin
                rollback;
                resignal;
        end;

        start transaction;
```

```
        insert into
                Tavolo(NumTavolo, MaxCommensali)
        values
                (numt, maxcomm);

        commit;
end!
```

- Questa procedura consente al manager di aggiungere un nuovo ingrediente con disp. iniziale e costo per kg

```
create procedure AggiungiNuovoIngrediente(
        in nomeIng varchar(20),
        in dispIniz int,
        in costoPerKg float)
begin
        declare exit handler for sqlexception
        begin
                rollback;
                resignal;
        end;

        start transaction;

        insert into
                Ingrediente(Nome, NumDisponibilitaScorte, CostoAlKg)
        values
                (nomeIng, dispIniz, costoPerKg);

        commit;
end!
```

- Questa procedura consente al manager di aggiungere un prodotto nel menu, il costo, se fa parte

del menu bar e se fa parte di quest'ultimo, se è alcolico o meno

```
create procedure AggiungiProdottoNelMenu(
                in prodNome varchar(20),
                in prodCostoUn float,
                in prodMenuBar boolean,
                in prodAlcolico boolean)
begin
        declare exit handler for sqlexception
        begin
                rollback;
                resignal;
        end;

        start transaction;

        insert into
                ProdottoNelMenu(Nome, CostoUnitario, IsBarMenu, IsAlcolico)
        values
                (prodNome, prodCostoUn, prodMenuBar, prodAlcolico);

        commit;
end!
```

- Questa procedura permette al manager di associare prodotto e ingrediente, in questo modo è possibile bloccare anche la scelta del prodotto di base se uno o più ingredienti che lo compongono non sono al momento disponibili

```
create procedure AssociaProdottoAIngrediente(
        in nomeProd varchar(20),
        in nomeIng varchar(20))
begin
        declare exit handler for sqlexception
```

```
        begin
            rollback;
            resignal;
        end;

        start transaction;

        insert into
            ComposizioneProdotto(NomeProdotto, NomeIngrediente)
        values
            (nomeProd, nomeIng);

        commit;
    end!
```

- Questa procedura consente al manager di aggiungere un turno

```
    create procedure AggiungiTurno(
        in inizio datetime,
        in fine datetime)
    begin
        declare exit handler for sqlexception
        begin
            rollback;
            resignal;
        end;

        start transaction;

        insert into
            Turno(DataOraInizio, DataOraFine)
        values
            (inizio, fine);
```

```
        commit;
end!
```

- Questa procedura consente al manager di rimuovere un prodotto dal menu

```
create procedure RimuoviProdottoNelMenu(in nomeProd varchar(20))
begin
        set transaction isolation level read committed;

        start transaction;

        delete from
                ProdottoNelMenu
        where
                Nome = nomeProd;

        commit;
end!
```

- Questa procedura consente al manager di rimuovere un ingrediente

```
create procedure RimuoviIngrediente(in nomeIng varchar(20))
begin
        declare exit handler for sqlexception
        begin
                rollback;
                resignal;
        end;

        set transaction isolation level read committed;

        start transaction;
```

```
        delete from
                Ingrediente
        where
                Nome = nomeIng;

        commit;
end!
```

- Questa procedura consente al manager di rimuovere un associazione prodotto e ingrediente

```
create procedure RimuoviAssocProdottoEIngrediente(
        in nomeProd varchar(20),
        in nomeIng varchar(20))
begin
        set transaction isolation level read committed;

        start transaction;

        delete from
                ComposizioneProdotto
        where
                NomeProdotto = nomeProd and NomeIngrediente = nomeIng;

        commit;
end!
```

- Questa procedura consente al manager di ottenere i turni a partire da questo istante (non ritorna mai i turni del passato) e il turno attivo.

```
create procedure OttieniTurni()
begin
        set transaction read only;
        set transaction isolation level read committed;
```

```
        start transaction;

        select
                DataOraInizio, DataOraFine
        from
                Turno
        where
                now() <= DataOraFine;

        commit;
end!
```

- Questa procedura consente al manager di ottenere gli utenti registrati nel sistema

```
create procedure OttieniUtenti()
begin
        set transaction read only;
        set transaction isolation level read committed;

        start transaction;

        select
                Username, Nome, Cognome, CF, ComuneResidenza,
                ComuneNascita, DataNascita, Ruolo
        from
                UtenteLavoratore;

        commit;
end!
```

- Questa procedura consente al manager di ottenere i tavoli registrati nel sistema, il # max di commensali, se sono occupati e se sono attivi (se non sono attivi non possono essere occupati)

```
create procedure OttieniTavoli()
begin
        set @current_time = now();

        set transaction read only;
        set transaction isolation level read committed;

        start transaction;

        select
                NumTavolo, MaxCommensali, IsOccupato, exists (
                        select
                                *
                        from
                                TurnoAssegnato Ta join Turno Tu on
                                        Ta.Turno = Tu.DataOraInizio
                        where
                                Ta.Tavolo = NumTavolo and
                                        Tu.DataOraInizio <= @current_time and
                                        @current_time <= Tu.DataOraFine)
        from
                Tavolo;

        commit;
end!
```

- Questa procedura consente al manager di ottenere solo ed esclusivamente i turni che hanno associato almeno un cameriere

```
create procedure OttieniTurniAssegnati()
begin
        set transaction read only;
```

```
        set transaction isolation level read committed;

        start transaction;

        select
            Tu.DataOraInizio, Tu.DataOraFine, Ta.Tavolo, U.Nome, U.Cognome,
U.Username
        from
            (TurnoAssegnato Ta join Turno Tu on
                Ta.Turno = Tu.DataOraInizio) join UtenteLavoratore U on
                    Ta.Cameriere = U.Username
        where
            now() <= Tu.DataOraFine
        order by
            Tu.DataOraInizio;

        commit;
    end!
```

- Questa procedura permette al manager di individuare il turno attualmente attivo con camerieri e tavoli attivi

```
    create procedure OttieniTurnoAttuale()
    begin
        set @current_time = now();

        set transaction read only;
        set transaction isolation level read committed;

        start transaction;

        select
            Tu.DataOraInizio, Tu.DataOraFine, Ta.Tavolo, U.Nome, U.Cognome,
```

```
U.Username
        from
            (TurnoAssegnato Ta right join Turno Tu on
                Ta.Turno = Tu.DataOraInizio) left join UtenteLavoratore U on
                    Ta.Cameriere = U.Username
        where
            Tu.DataOraInizio <= @current_time and
                @current_time <= Tu.DataOraFine;

        commit;
    end!
```

- Questa procedura consente al manager di assegnare a un utente cameriere un turno e un tavolo

```
    create procedure AssegnaTurno(
            in numTavolo smallint,
            in dataOraInizioTurno datetime,
            in camerieUsername varchar(10))
    begin
        declare exit handler for sqlexception
        begin
            rollback;
            resignal;
        end;

        start transaction;

        insert into
            TurnoAssegnato(Tavolo, Turno, Cameriere)
        values
            (numTavolo, dataOraInizioTurno, camerieUsername);

        commit;
```

```
end!
```

- Questa procedura consente al manager di ottenere le informazioni basilari sui prodotti attualmente sul menu

```
create procedure OttieniMenu()
begin
        set transaction read only;
        set transaction isolation level read committed;

        start transaction;

        select
                Nome, IsBarMenu, IsAlcolico, CostoUnitario
        from
                ProdottoNelMenu
        order by
                IsBarMenu, Nome asc;

        commit;
end!
```

- Questa procedura consente al manager di ottenere la composizione di prodotto

```
create procedure OttieniComposizioneProdotto()
begin
        set transaction read only;
        set transaction isolation level read committed;

        start transaction;

        select
                NomeProdotto, NomeIngrediente
```

```
        from
            ComposizioneProdotto;

        commit;
    end!
```

- Questa procedura consente al manager di ottenere informazioni di disponibilità e costo per kg di tutti gli ingredienti di cui si tiene traccia nel sistema

```
    create procedure OttieniIngredienti()
    begin
        set transaction read only;
        set transaction isolation level read committed;

        start transaction;

        select
            Nome, NumDisponibilitaScorte, CostoAlKg
        from
            Ingrediente;

        commit;
    end!
```

- Questa procedura consente al manager di incrementare la disponibilità unitaria di ingredienti

```
    create procedure IncDispIngrediente(
        in nomeIng varchar(20),
        in incBy int)
    begin
        set transaction isolation level read committed;

        start transaction;
```

```
        update
                Ingrediente
        set
                NumDisponibilitaScorte = NumDisponibilitaScorte + incBy
        where
                Nome = nomeIng;

        commit;
end!
```

- Questa procedura consente al manager di ottenere le entrate mensili o giornaliere, basandosi sul parametro in input mensili (true = per entrate per mese, false = entrate per giorno) (InTimeRange è una funzione disponibile nell'appendice) e ritorna due result set:
  - Il primo è la somma totale di scontrini nel lasso di tempo specificato e la somma di tutti i costi totali di essi (ottenendo quindi le entrate totali)
  - Il secondo sono i singoli scontrini (id fiscale, data ora emissione e costo totale del singolo scontrino)

```
create procedure OttieniEntrate(in mensili boolean)
begin
        set @current_time = now();

        set transaction read only;
        set transaction isolation level repeatable read;

        start transaction;

        select
                count(*) as NumScontrini,
                sum(CostoTotale) as IncassoTotale
        from
                Scontrino
```

```
        where
                IsPagato and InTimeRange(mensili, DataOraEmissione, @current_time);

        select
                IdFiscale, DataOraEmissione, CostoTotale
        from
                Scontrino
        where
                IsPagato and InTimeRange(mensili, DataOraEmissione, @current_time);

        commit;
end!
```

- Questa procedura consente al manager di ottenere tutti gli scontrini stampati ma non ancora pagati

```
create procedure OttieniScontriniNonPagati()
begin
        set transaction read only;
        set transaction isolation level read committed;

        start transaction;

        select
                IdFiscale, DataOraEmissione, CostoTotale
        from
                Scontrino
        where
                IsPagato = false;

        commit;
end!
```

- Questa procedura consente al manager di flaggare uno scontrino come pagato basandosi sul suo id fiscale

```
create procedure ContrassegnaScontrinoPagato(in idFisc int)
begin
        set transaction isolation level read committed;

        start transaction;

        update
                Scontrino
        set
                IsPagato = true
        where
                IdFiscale = idFisc and IsPagato = false;

        commit;
end!
```

- Questa procedura complessa consente al manager di assegnare un tavolo al cliente, viene verificata la disponibilità se il tavolo è attivo, se il numero di commensali rientra nel numero max. accettabile. Se queste caratteristiche non sono riscontrate, viene emesso un segnale con il messaggio d'errore opportuno altrimenti si inserisce una nuova istanza di TavoloOccupato e si aggiorna "Tavolo" flaggandolo come occupato, la stored procedure restituisce come result set una tabella 1x1: il numero di tavolo trovato e assegnato.

```
create procedure AssegnaTavoloACliente(
                in cliNome varchar(20),
                in cliCognome varchar(20),
                in numComm tinyint)
begin
        declare exit handler for sqlexception
        begin
```

```
        rollback;
        resignal;
    end;

    set @current_time = now();

    set transaction isolation level repeatable read;

    start transaction;

    set @numTavoloAdatto = (
        select
            NumTavolo
        from
            (Tavolo T join TurnoAssegnato Ta on
                T.NumTavolo = Ta.Tavolo) join Turno Tu on
                    Ta.Turno = Tu.DataOraInizio
        where
            T.IsOccupato = false and
            Tu.DataOraInizio <= @current_time and
            @current_time <= Tu.DataOraFine and
            numComm <= T.MaxCommensali
        limit 1);

    if @numTavoloAdatto is NULL then
        signal sqlstate '45005'
            set message_text = "Nessun tavolo disponibile per l'assegnazione";
    end if;

    insert into
        TavoloOccupato(DataOraOccupazione, Nome, Cognome, NumCommensali,
Tavolo)
    values
        (now(), cliNome, cliCognome, numComm, @numTavoloAdatto);
```

```
        update
                Tavolo
        set
                IsOccupato = true
        where
                NumTavolo = @numTavoloAdatto;

        select @numTavoloAdatto as NumTavolo;

        commit;
end!
```

- Questa procedura consente al manager di ottenere i tavoli occupati tali per cui è possibile stampare lo scontrino (serviti almeno una volta e tutte ordinazioni completate) (TutteOrdConcluse è una funzione disponibile nell'appendice)

```
create procedure OttieniTavoliScontrinoStampabile()
begin
        set transaction read only;
        set transaction isolation level repeatable read;

        start transaction;

        select
                Tavolo, DataOraOccupazione
        from
                TavoloOccupato
        where
                TutteOrdConcluse(DataOraOccupazione) and
                not exists (
                        select
                                *
```

```
                    from
                        Scontrino
                    where
                        TavoloOccupato = DataOraOccupazione);

        commit;
    end!
```

- Questa è una procedura complessa che consente al manager di stampare lo scontrino per un tavolo occupato: verifica se è possibile stampare lo scontrino in base alle condizioni specificate nel punto precedente (per esito negativo viene emesso un segnale), poi verifica se uno scontrino è già stato stampato (in tale caso, non viene stampato e viene quindi emesso uns segnale), altrimenti inserisce lo scontrino e "restituisce" al chiamante due result set: uno che contiene le singole ordinazioni, scelte, prodotti, ing. extra ordinati e l'altro che contiene l'id fiscale, la data e ora di emisssione e il costo totale (entrambi i RS verranno quindi utilizzati dal thin client per formattare correttamente lo scontrino e stamparlo in maniera "pretty") (CalcoloCostoTotale è una funzione disponibile nell'appendice)

```
create procedure StampaScontrino(in dataOraOcc datetime)
begin
    declare exit handler for sqlexception
    begin
        rollback;
        resignal;
    end;

    set transaction isolation level repeatable read;

    start transaction;

    if TutteOrdConcluse(dataOraOcc) = false then
        signal sqlstate '45014'
            set message_text = "Impossibile rilasciare lo scontrino. Esistono ordinazioni
```

```
aperte.";
        end if;

        if exists (
                        select
                                *
                        from
                                Scontrino
                        where
                                TavoloOccupato = dataOraOcc) then
                signal sqlstate '45006'
                        set message_text = "Scontrino già stampato";
        end if;

        insert into
                Scontrino(
                        DataOraEmissione,
                        CostoTotale,
                        TavoloOccupato)
        values
                (now(), CalcoloCostoTotale(dataOraOcc), dataOraOcc);

        select
                Ord.NumOrdinazionePerTavolo as NumOrdinazione,
                Sdc.NumSceltaPerOrdinazione as NumScelta,
                Sdc.ProdottoNelMenu as Prodotto,
                Ap.Ingrediente as IngredienteExtra,
                Ap.QuantitaInGr as QuantitaInGr
        from
                (Ordinazione Ord join SceltaDelCliente Sdc on
                        Ord.TavoloOccupato = Sdc.TavoloOccupato and
                        Ord.NumOrdinazionePerTavolo = Sdc.NumOrdinazionePerTavolo)
                left join AggiuntaAlProdotto Ap on
                        Ord.TavoloOccupato = Ap.TavoloOccupato and
```

```
                Ord.NumOrdinazionePerTavolo = Ap.NumOrdinazionePerTavolo and
                Sdc.NumSceltaPerOrdinazione = Ap.NumSceltaPerOrdinazione
        where
                Ord.TavoloOccupato = dataOraOcc
        order by
                Ord.NumOrdinazionePerTavolo,
                Sdc.NumSceltaPerOrdinazione,
                Sdc.ProdottoNelMenu asc;

        select
                IdFiscale, DataOraEmissione, CostoTotale
        from
                Scontrino
        where
                TavoloOccupato = dataOraOcc;

        commit;
end!
```

- Questa procedura consente al cameriere di ottenere i tavoli occupati di sua competenza per questo turno (da ora si utilizzano gli username per differenziare i camerieri e i tavoli di loro competenza)

```
create procedure OttieniTavoliDiCompetenza(in username varchar(10))
begin
        set @current_time = now();

        set transaction read only;
        set transaction isolation level read committed;

        start transaction;

        select
```

```
        Ta.NumTavolo as NumTavolo,
        Ta.IsOccupato as IsOccupato,
        Tocc.DataOraOccupazione as DataOraOccupazione,
        Tocc.NumCommensali as NumCommensali,
        Tocc.IsServitoAlmenoUnaVolta as IsServitoAlmenoUnaVolta
    from
        ((Tavolo Ta left join TavoloOccupato Tocc on
            Ta.NumTavolo = Tocc.Tavolo) join TurnoAssegnato Tu on
                Tu.Tavolo = Ta.NumTavolo) join Turno T on
                    T.DataOraInizio = Tu.Turno
    where
        Tu.Cameriere = username and
        T.DataOraInizio <= @current_time and
        @current_time <= T.DataOraFine and
        Ta.IsOccupato = true and
        not exists (
            select
                *
            from
                Scontrino
            where
                TavoloOccupato = Tocc.DataOraOccupazione and
                IsPagato = true);

    commit;
end!
```

- Questa procedura consente al cameriere di ottenere tutti i tavoli di competenza del cameriere (anche quelli non occupati)

```
create procedure OttieniTavoliAssegnati(in username varchar(10))
begin
    set @current_time = now();
```

```
        set transaction read only;
        set transaction isolation level read committed;

        start transaction;

        select
                Ta.NumTavolo as NumTavolo
        from
                (Tavolo Ta join TurnoAssegnato Tu on
                        Ta.NumTavolo = Tu.Tavolo) join Turno T on
                                Tu.Turno = T.DataOraInizio
        where
                Tu.Cameriere = username and
                T.DataOraInizio <= @current_time and
                @current_time <= T.DataOraFine;

        commit;
end!
```

- Questa procedura consente al cameriere di iniziare una nuova ordinazione per il tavolo: questo avviene se e soltanto se il tavolo è abilitato al funzionamento col cameriere attuale, è occupato, ed è attivo nel turno corrente (altrimenti viene emesso un segnale). In esito positivo si procede all'inserimento dell'ordinazione progressivamente a quelle precedenti per il tavolo occupato. I trigger descritti prima consentono di effettuare le verifiche di presenza di ordinazioni precedenti non ancora completate. In tal caso viene effettuato il rollback e resignal al caller e quindi non viene aperta la nuova ordinazione. (CanWorkOnTable è una funzione presente nell'appendice)

```
create procedure PrendiOrdinazione(
        in dataOraOcc datetime,
        in usern varchar(10))
begin
        declare exit handler for sqlexception
```

```
        begin
                rollback;
                resignal;
        end;

        set transaction isolation level read committed;

        start transaction;

        if CanWorkOnTable(dataOraOcc, usern, now()) then
                set @counter = (
                        select
                                count(*)
                        from
                                Ordinazione
                        where
                                TavoloOccupato = dataOraOcc) + 1;

                insert into
                        Ordinazione(TavoloOccupato, NumOrdinazionePerTavolo)
                values
                        (dataOraOcc, @counter);
        else
                signal sqlstate '45011'
                        set message_text = "Non è possibile selezionare il tavolo (ad esempio,
non è occupato al momento)";
        end if;

        commit;
    end!
```

- Questa procedura consente al cameriere di chiudere l'ultima ordinazione aperta, avvengono le stesse verifiche del punto precedente, solo che invece che verificare per le vecchie ordinazioni,

si verifica che esista una ordinazione aperta in attesa di chiusura (grazie ai trigger descritti sopra) (come al solito, in caso negativo viene emesso un segnale). I check constraint definiti nelle tabelle forzano il fatto che alla chiusura dell'ordinazione, deve essere valido il fatto che Costo di Ordinazione sia maggiore di 0, quindi che sia presente almeno una scelta nell'ordinazione complessiva

```
create procedure ChiudiOrdinazione(
        in dataOraOcc datetime,
        in usern varchar(20))
begin
        declare exit handler for sqlexception
        begin
                rollback;
                resignal;
        end;

        set @current_time = now();

        set transaction isolation level repeatable read;

        start transaction;

        if CanWorkOnTable(dataOraOcc, usern, @current_time) then
                set @numOrd =(
                        select
                                count(*)
                        from
                                Ordinazione
                        where
                                TavoloOccupato = dataOraOcc);

                update
                        Ordinazione
                set
```

```
                DataOraRichiesta = @current_time
        where
                TavoloOccupato = dataOraOcc and
                DataOraRichiesta is NULL and
                DataOraCompletamento is NULL and
                NumOrdinazionePerTavolo = @numOrd;
    else
        signal sqlstate '45011'
            set message_text = "Non è possibile selezionare il tavolo (ad esempio, non è occupato al momento)";
    end if;

    commit;
end!
```

- Questa procedura consente al cameriere di prendere una scelta per l'ordinazione attualmente aperta, avvengono verifiche similari ai punti precedenti.

```
create procedure PrendiSceltaPerOrd(
    in dataOraOcc datetime,
    in nomeProd varchar(20),
    in usern varchar(10))
begin
    declare exit handler for sqlexception
    begin
        rollback;
        resignal;
    end;

    set transaction isolation level repeatable read;

    start transaction;
```

```
if CanWorkOnTable(dataOraOcc, usern, now()) then
        set @numOrd = (
                select
                        NumOrdinazionePerTavolo
                from
                        Ordinazione
                where
                        TavoloOccupato = dataOraOcc and
                        DataOraRichiesta is NULL);

        if @numOrd is NULL then
                signal sqlstate '45015'
                        set message_text = "Ordinazione chiusa";
        end if;

        set @numSc = (
                select
                        count(*)
                from
                        SceltaDelCliente
                where
                        TavoloOccupato = dataOraOcc and
                NumOrdinazionePerTavolo = @numOrd) + 1;

        insert into
                SceltaDelCliente(TavoloOccupato,
                                                NumOrdinazionePerTavolo,
                                                NumSceltaPerOrdinazione,
                                                ProdottoNelMenu)
        values(
                dataOraOcc,
                @numOrd,
                @numSc,
                nomeProd);
```

```
        else
            signal sqlstate '45011'
                set message_text = "Non è possibile selezionare il tavolo (ad esempio, non è occupato al momento)";
        end if;

        commit;
    end!
```

- Questa procedura consente al cameriere di ottenere le scelte per l'ordinazione attualmente aperta. Se nessuna ordinazione è aperta, essa ritorna un RS vuoto (a priori deve essere possibile lavorare su quel particolare tavolo per il cameriere)

```
    create procedure OttieniSceltePerOrdinazione(
        in dataOraOcc datetime,
        in usern varchar(10))
    begin
        set transaction read only;
        set transaction isolation level read committed;

        start transaction;

        if CanWorkOnTable(dataOraOcc, usern, now()) then
            select
                Sdc.NumOrdinazionePerTavolo as NumOrdinazionePerTavolo,
                Sdc.NumSceltaPerOrdinazione as NumSceltaPerOrdinazione,
                Sdc.ProdottoNelMenu as ProdottoNelMenu
            from
                SceltaDelCliente Sdc join Ordinazione Ord on
                    Sdc.TavoloOccupato = Ord.TavoloOccupato and
                    Sdc.NumOrdinazionePerTavolo = Ord.NumOrdinazionePerTavolo
            where
```

```
                    Sdc.TavoloOccupato = dataOraOcc and
                    Ord.DataOraRichiesta is NULL;
        else
            signal sqlstate '45011'
                    set message_text = "Non è possibile selezionare il tavolo (ad esempio, non è occupato al momento)";
        end if;

        commit;
    end!
```

- Questa procedura consente al cameriere di aggiungere ingredienti extra alla particolare scelta dell'ordinazione specificando il nome dell'ingrediente e la quantità desiderata, in modo da poter poi aggiungerne il costo (al solito è a priori necessario che sia possibile lavorare sul tavolo per il cameriere).

```
    create procedure AggiungiIngExtraAllaScelta(
        in dataOraOcc datetime,
        in numOrdPerTav tinyint,
        in numSceltaPerOrd tinyint,
        in ing varchar(20),
        in qtGr float,
        in usern varchar(10))
    begin
        declare exit handler for sqlexception
        begin
            rollback;
            resignal;
        end;

        set transaction isolation level read committed;

        start transaction;
```

```
        if CanWorkOnTable(dataOraOcc, usern, now()) then
            insert into
                AggiuntaAlProdotto(
                    TavoloOccupato,
                    NumOrdinazionePerTavolo,
                    NumSceltaPerOrdinazione,
                    Ingrediente,
                    QuantitaInGr)
            values
                (
                    dataOraOcc,
                    numOrdPerTav,
                    numSceltaPerOrd,
                    ing,
                    qtGr
                );
        else
            signal sqlstate '45011'
                set message_text = "Non è possibile selezionare il tavolo (ad esempio, non è occupato al momento)";
        end if;

        commit;
    end!
```

- Questa procedura consente al cameriere di ottenere le singole scelte espletate (di ordinazioni chiuse) e abilitate alla consegna

```
    create procedure OttieniScelteEspletate(in usern varchar(10))
    begin
        set transaction read only;
        set transaction isolation level repeatable read;
```

```
        start transaction;

        select
                TavoloOccupato,
                NumOrdinazionePerTavolo,
                NumSceltaPerOrdinazione,
                Tavolo,
                ProdottoNelMenu
        from
                SceltaDelCliente join TavoloOccupato on
                        TavoloOccupato = DataOraOccupazione
        where
                CanWorkOnTable(TavoloOccupato, usern, now()) and
                DataOraEspletata is not NULL and
                DataOraCompletamento is NULL;

        commit;
end!
```

- Questa procedura consente al cameriere di effettuare una consegna di una scelta espletata

```
create procedure EffettuaConsegna(
        in dataOraOcc datetime,
        in numOrd tinyint,
        in numScelta tinyint,
        in usern varchar(10))
begin
        declare exit handler for sqlexception
        begin
                rollback;
                resignal;
        end;
```

```
        set @current_time = now();

        set transaction isolation level read committed;

        start transaction;

        if CanWorkOnTable(dataOraOcc, usern, @current_time) then
            update
                SceltaDelCliente
            set
                DataOraCompletamento = @current_time
            where
                TavoloOccupato = dataOraOcc and
                NumOrdinazionePerTavolo = numOrd and
                NumSceltaPerOrdinazione = numScelta;
        else
            signal sqlstate '45011'
                set message_text = "Non è possibile selezionare il tavolo (ad esempio,
non è occupato al momento)";
        end if;

        commit;
    end!
```

- Questa procedura consente al pizzaiolo / barman di visualizzare le scelte da preparare

```
    create procedure OttieniScelteDaPreparare()
    begin
        set transaction read only;
        set transaction isolation level read committed;

        start transaction;
```

```
        select
                Sdc.TavoloOccupato as TavoloOccupato,
                Tocc.Tavolo as NumTavolo,
                Sdc.NumOrdinazionePerTavolo as NumOrdinazionePerTavolo,
                Sdc.NumSceltaPerOrdinazione as NumSceltaPerOrdinazione,
                Sdc.ProdottoNelMenu as ProdottoNelMenu
        from
                (SceltaDelCliente Sdc join Ordinazione Ord on
                        Sdc.TavoloOccupato = Ord.TavoloOccupato and
                        Sdc.NumOrdinazionePerTavolo = Ord.NumOrdinazionePerTavolo)
join TavoloOccupato Tocc on
                                Sdc.TavoloOccupato = Tocc.DataOraOccupazione
        where
                Sdc.LavoratoreCucina is NULL and
                Sdc.DataOraEspletata is NULL and
                Sdc.DataOraCompletamento is NULL and
                Ord.DataOraCompletamento is NULL and
                Ord.DataOraRichiesta is not NULL;

        commit;
    end!
```

- Questa procedura consente al pizzaiolo / barman di prendere in carico una delle scelte. Per prima cosa si verifica che il prodotto fa parte del menu bar, quindi i ruoli in UtenteLavoratore devono corrispondere (sta volta è necessario passare l'username) (barman = prodotto menu bar, pizzaiolo = prodotto menu pizzeria) per procedere con la presa in carico (in esito negativo viene emesso un segnale).

```
    create procedure PrendiInCaricoScelta(
            in dataOraOcc datetime,
            in numOrd int,
            in numSc int,
```

```
        in nomeProd varchar(20),
        in usern varchar(10))
begin
        declare exit handler for sqlexception
        begin
                rollback;
                resignal;
        end;

        set transaction isolation level read committed;

        start transaction;

        set @role = (
                select
                        Ruolo
                from
                        UtenteLavoratore
                where
                        Username = usern);

        set @isBarMenu = (
                select
                        IsBarMenu
                from
                        ProdottoNelMenu
                where
                        Nome = nomeProd);

        if @isBarMenu = true and @role <> 4 then
                signal sqlstate '45016'
                        set message_text = "Non sei un barista!";
        elseif @isBarMenu = false and @role <> 3 then
                signal sqlstate '45017'
```

```
                set message_text = "Non sei un pizzaiolo!";
        end if;

        update
                SceltaDelCliente
        set
                LavoratoreCucina = usern
        where
                TavoloOccupato = dataOraOcc and
                NumOrdinazionePerTavolo = numOrd and
                NumSceltaPerOrdinazione = numSc and
                LavoratoreCucina is NULL;

        commit;
end!
```

- Questa procedura consente al pizzaiolo / barman di visualizzare le scelte prese in carico ancora da espletare

```
create procedure OttieniSceltePreseInCaricoNonEspletate(in usern varchar(10))
begin
        set transaction read only;
        set transaction isolation level read committed;

        start transaction;

        select
                Sdc.TavoloOccupato as TavoloOccupato,
                Tocc.Tavolo as NumTavolo,
                Sdc.NumOrdinazionePerTavolo as NumOrdinazionePerTavolo,
                Sdc.NumSceltaPerOrdinazione as NumSceltaPerOrdinazione,
                Sdc.ProdottoNelMenu as ProdottoNelMenu
        from
```

```
            SceltaDelCliente Sdc join TavoloOccupato Tocc on
                Sdc.TavoloOccupato = Tocc.DataOraOccupazione
        where
            Sdc.DataOraEspletata is NULL and
            Sdc.LavoratoreCucina = usern;

        commit;
    end!
```

- Questa procedura consente al pizzaiolo / barman di espletare una scelta già presa in carico

```
    create procedure EspletaSceltaPresaInCarico(
        in dataOraOcc datetime,
        in numOrd int,
        in numSc int,
        in usern varchar(10)
    )
    begin
        declare exit handler for sqlexception
        begin
            rollback;
            resignal;
        end;

        set transaction isolation level read committed;

        start transaction;

        update
            SceltaDelCliente
        set
            DataOraEspletata = now()
        where
```

```
                TavoloOccupato = dataOraOcc and
                NumOrdinazionePerTavolo = numOrd and
                NumSceltaPerOrdinazione = numSc and
                DataOraEspletata is NULL and
                LavoratoreCucina = usern;

        commit;
    end!
```

- Questa procedura consente al pizzaiolo / barman di sapere esattamente cosa è necessario preparare per la scelta presa in carico (includendo quindi ingrediente extra e la quantita in gr)

```
    create procedure OttieniInfoProdottiDiScelteInCarico(in usern varchar(10))
    begin
        set transaction read only;
        set transaction isolation level read committed;

        start transaction;

        select
            Tocc.Tavolo as Tavolo,
            Sdc.NumOrdinazionePerTavolo as NumOrdPerTavolo,
            Sdc.NumSceltaPerOrdinazione as NumSceltaPerOrd,
            Sdc.ProdottoNelMenu as Prodotto,
            Ap.Ingrediente as IngredienteExtra,
            Ap.QuantitaInGr as QuantitaIngredienteExtraInGr
        from
            ((SceltaDelCliente Sdc join ProdottoNelMenu Prod on
                Sdc.ProdottoNelMenu = Prod.Nome) left join AggiuntaAlProdotto Ap
on
                    Ap.TavoloOccupato = Sdc.TavoloOccupato and
                    Ap.NumOrdinazionePerTavolo =
Sdc.NumOrdinazionePerTavolo and
```

```
                                        Ap.NumSceltaPerOrdinazione                    =
Sdc.NumSceltaPerOrdinazione) join TavoloOccupato Tocc on
                                                Sdc.TavoloOccupato = Tocc.DataOraOccupazione
            where
                    Sdc.LavoratoreCucina = usern and
                    Sdc.DataOraEspletata is NULL;

            commit;
    end!
```

Appendice: Implementazione

Codice SQL per instanziare il database

```
drop database if exists pizzeriadb;

create database pizzeriadb;

use pizzeriadb;


-- passwd SHA1

create table UtenteLavoratore (

    Username varchar(10) primary key,

    Nome varchar(20) not null,

    Cognome varchar(20) not null,

    ComuneResidenza varchar(34) not null,

    DataNascita date not null,

    ComuneNascita varchar(34) not null,

    CF char(16) not null,

    Passwd char(40) not null,

    Ruolo tinyint not null,

    unique (

        CF

    ),

    check (
```

```
        Ruolo in (1,2,3,4)
    )
);

create table Tavolo (
    NumTavolo smallint primary key,
    MaxCommensali tinyint unsigned not null,
    IsOccupato boolean not null default false
);

create table Ingrediente (
    Nome varchar(20) primary key,
    NumDisponibilitaScorte int unsigned not null,
    CostoAlKg float not null,
    check(
        CostoAlKg > 0
    )
);

create table ProdottoNelMenu (
    Nome varchar(20) primary key,
```

```
    CostoUnitario float not null,
    IsBarMenu boolean not null,
    IsAlcolico boolean,
    check(
        CostoUnitario > 0 and
        ((IsBarMenu and IsAlcolico is not NULL) or
                (not IsBarMenu and isAlcolico is NULL))
    )
);

create table ComposizioneProdotto(
    NomeProdotto varchar(20)
        references ProdottoNelMenu(Nome)
        on delete cascade,
    NomeIngrediente varchar(20)
        references Ingrediente(Nome)
        on delete no action,
    primary key(
        NomeProdotto,
        NomeIngrediente
    )
```

```
);

create index ComposizioneProdotto_NomeProdotto_fk
    on ComposizioneProdotto(NomeProdotto asc);

create index ComposizioneProdotto_NomeIngrediente_fk
    on ComposizioneProdotto(NomeIngrediente asc);

create table Turno (
    DataOraInizio datetime primary key,
    DataOraFine datetime not null,
    check(
        DataOraFine > DataOraInizio
    ),
    unique(
        DataOraFine
    )
);

create table TurnoAssegnato (
    Turno datetime
```

```
        references Turno(DataOraInizio)
        on delete no action,
    Tavolo smallint
        references Tavolo(NumTavolo)
        on delete cascade,
    Cameriere varchar(10)
        references UtenteLavoratore(Username)
        on delete no action,
    primary key(
        Turno,
        Tavolo,
        Cameriere
    )
);

create index TurnoAssegnato_Turno_fk
    on TurnoAssegnato(Turno asc);

create index TurnoAssegnato_Tavolo_fk
    on TurnoAssegnato(Tavolo asc);
```

```
create index TurnoAssegnato_Cameriere_fk
    on TurnoAssegnato(Cameriere asc);

create table TavoloOccupato (
    DataOraOccupazione datetime primary key,
    Nome varchar(20) not null,
    Cognome varchar(20) not null,
    NumCommensali tinyint unsigned not null,
    IsServitoAlmenoUnaVolta boolean not null default false,
    Tavolo smallint not null
        references Tavolo(NumTavolo)
        on delete cascade
);

create index TavoloOccupato_Tavolo_fk
    on TavoloOccupato(Tavolo asc);

create table Scontrino (
    IdFiscale int auto_increment primary key,
    DataOraEmissione datetime not null,
    CostoTotale float not null,
```

```
    IsPagato boolean not null default false,
    TavoloOccupato datetime
        references TavoloOccupato(DataOraOccupazione)
        on delete set null,
    check(
        CostoTotale > 0
    ),
    unique(
        DataOraEmissione
    )
);

create index Scontrino_TavoloOccupato_fk
    on Scontrino(TavoloOccupato asc);

create table Ordinazione (
    TavoloOccupato datetime
        references TavoloOccupato(DataOraOccupazione)
        on delete cascade,
    NumOrdinazionePerTavolo tinyint,
    DataOraRichiesta datetime default NULL,
```

```
    Costo float unsigned not null default 0,
    DataOraCompletamento datetime default NULL,
    primary key (
            TavoloOccupato,
            NumOrdinazionePerTavolo
    ),
    check(
        NumOrdinazionePerTavolo > 0 and
            (DataOraRichiesta is NULL or
                (DataOraRichiesta is not NULL and Costo > 0))
    )
);

create index Ordinazione_TavoloOccupato_fk
    on Ordinazione(TavoloOccupato asc);

create table SceltaDelCliente (
    TavoloOccupato datetime,
    NumOrdinazionePerTavolo tinyint,
    NumSceltaPerOrdinazione tinyint,
    DataOraCompletamento datetime default NULL,
```

```
DataOraEspletata datetime default NULL,
ProdottoNelMenu varchar(20)
    references ProdottoNelMenu(Nome)
    on delete set NULL,
LavoratoreCucina varchar(10) default NULL
    references UtenteLavoratore(Username)
    on delete set default,
foreign key (
    TavoloOccupato,
    NumOrdinazionePerTavolo)
    references Ordinazione(
            TavoloOccupato,
            NumOrdinazionePerTavolo
)
on delete cascade,
primary key (
    TavoloOccupato,
    NumOrdinazionePerTavolo,
    NumSceltaPerOrdinazione
),
check(
```

```
        NumSceltaPerOrdinazione > 0
    )
);

create index SceltaDelCliente_ProdottoNelMenu_fk
    on SceltaDelCliente(ProdottoNelMenu asc);

create index SceltaDelCliente_LavoratoreCucina_fk
    on SceltaDelCliente(LavoratoreCucina asc);

create index
    SceltaDelCliente_TavoloOccupato_NumOrdinazionePerTavolo_fk
    on SceltaDelCliente(TavoloOccupato asc,
        NumOrdinazionePerTavolo asc);

create table AggiuntaAlProdotto (
    TavoloOccupato datetime,
    NumOrdinazionePerTavolo tinyint,
    NumSceltaPerOrdinazione tinyint,
    Ingrediente varchar(20)
        references Ingrediente(Nome)
```

```
    on delete cascade,
QuantitaInGr float not null,
foreign key (
    TavoloOccupato,
    NumOrdinazionePerTavolo,
    NumSceltaPerOrdinazione)
    references SceltaDelCliente(
            TavoloOccupato,
            NumOrdinazionePerTavolo,
            NumSceltaPerOrdinazione
)
on delete cascade,
primary key (
    TavoloOccupato,
    NumOrdinazionePerTavolo,
    NumSceltaPerOrdinazione,
    Ingrediente
),
check(
    QuantitaInGr > 0
)
```

```
);


create index AggiuntaAlProdotto_Ingrediente_fk on
    AggiuntaAlProdotto(Ingrediente asc);


create index
    AggiuntaAlProdotto_TavoloOcc_NumOrdPerTa_NumSceltaPerOrd_fk
    on AggiuntaAlProdotto(TavoloOccupato asc,
        NumOrdinazionePerTavolo asc,
        NumSceltaPerOrdinazione asc);

delimiter !

create function InTimeRange(
        monthly boolean,
        tm datetime,
        nowtime datetime)
returns boolean deterministic
begin
        set @by_month = YEAR(nowtime) = YEAR(tm) and
                        MONTH(nowtime) = MONTH(tm);

        if monthly then
                return @by_month;
        end if;

        return @by_month and DAY(nowtime) = DAY(tm);
end!
```

```
create function TutteOrdConcluse(dataOraOcc datetime)
returns boolean deterministic
begin
	declare numComplete int;
	declare numTotali int;

	select
		count(*)
	into
		numComplete
	from
		Ordinazione
	where
		TavoloOccupato = dataOraOcc and
		DataOraRichiesta is not NULL and
		DataOraCompletamento is not NULL;

	select
		count(*)
	into
		numTotali
	from
		Ordinazione
	where
		TavoloOccupato = dataOraOcc;

	return numTotali > 0 and numComplete > 0 and
		numTotali = numComplete;
end!

create function CalcoloCostoTotale(dataOraOcc datetime)
returns float deterministic
begin
```

```
declare costoTmp float;
declare shouldLeave boolean;
declare cur1 cursor for
        select
                Costo
        from
                Ordinazione
        where
                TavoloOccupato = dataOraOcc;
declare continue handler for not found set shouldLeave = true;

set shouldLeave = false;

set @costoTotale = 0;

open cur1;

read_loop: loop
        fetch cur1 into costoTmp;

        if not shouldLeave then

                set @costoTotale = @costoTotale + costoTmp;

        else

                leave read_loop;

        end if;

end loop read_loop;

close cur1;
```

```
        return @costoTotale;
end!

create function CanWorkOnTable(
        dataOraOcc datetime,
        usern varchar(10),
        nowtime datetime)
returns boolean deterministic
begin
        if exists(
                select
                        *
                from
                        (TavoloOccupato Tocc join Tavolo Ta on
                                Tocc.Tavolo = Ta.NumTavolo) join TurnoAssegnato TuAs on
                                        TuAs.Tavolo = Ta.NumTavolo join Turno Tu on
                                                Tu.DataOraInizio = TuAs.Turno and
                                                Tu.DataOraInizio <= nowtime and
                                                nowtime <= Tu.DataOraFine
                where
                        TuAs.Cameriere = usern and
                        Tocc.DataOraOccupazione = dataOraOcc) then

                return true;
        end if;

        return false;
end!

delimiter ;

drop user if exists login;
create user 'login'@'%' identified by 'login';
```

```
drop user if exists pizzaiolo;
create user 'pizzaiolo'@'%' identified by 'pizzaiolo';

drop user if exists barman;
create user 'barman'@'%' identified by 'barman';

drop user if exists cameriere;
create user 'cameriere'@'%' identified by 'cameriere';

drop user if exists manager;
create user 'manager'@'%' identified by 'manager';

grant execute on procedure TentaLogin to 'login';

grant execute on procedure RegistraUtente to 'manager';
grant execute on procedure RipristinoPassword to 'manager';
grant execute on procedure AggiungiNuovoTavolo to 'manager';
grant execute on procedure AggiungiNuovoIngrediente to 'manager';
grant execute on procedure AggiungiProdottoNelMenu to 'manager';
grant execute on procedure AssociaProdottoAIngrediente to 'manager';
grant execute on procedure AggiungiTurno to 'manager';
grant execute on procedure RimuoviProdottoNelMenu to 'manager';
grant execute on procedure RimuoviIngrediente to 'manager';
grant execute on procedure RimuoviAssocProdottoEIngrediente to 'manager';
grant execute on procedure OttieniTurni to 'manager';
grant execute on procedure OttieniUtenti to 'manager';
grant execute on procedure OttieniTavoli to 'manager';
grant execute on procedure AssegnaTurno to 'manager';
grant execute on procedure OttieniTurnoAttuale to 'manager';
grant execute on procedure OttieniTurniAssegnati to 'manager';
grant execute on procedure OttieniMenu to 'manager';
grant execute on procedure OttieniComposizioneProdotto to 'manager';
grant execute on procedure OttieniIngredienti to 'manager';
grant execute on procedure IncDispIngrediente to 'manager';
```

```
grant execute on procedure OttieniEntrate to 'manager';
grant execute on procedure OttieniScontriniNonPagati to 'manager';
grant execute on procedure ContrassegnaScontrinoPagato to 'manager';
grant execute on procedure AssegnaTavoloACliente to 'manager';
grant execute on procedure OttieniTavoliScontrinoStampabile to 'manager';
grant execute on procedure StampaScontrino to 'manager';

grant execute on procedure OttieniTavoliDiCompetenza to 'cameriere';
grant execute on procedure PrendiOrdinazione to 'cameriere';
grant execute on procedure ChiudiOrdinazione to 'cameriere';
grant execute on procedure PrendiSceltaPerOrd to 'cameriere';
grant execute on procedure OttieniSceltePerOrdinazione to 'cameriere';
grant execute on procedure AggiungiIngExtraAllaScelta to 'cameriere';
grant execute on procedure OttieniScelteEspletate to 'cameriere';
grant execute on procedure EffettuaConsegna to 'cameriere';
grant execute on procedure OttieniTavoliAssegnati to 'cameriere';

grant execute on procedure OttieniScelteDaPreparare to 'pizzaiolo';
grant execute on procedure PrendiInCaricoScelta to 'pizzaiolo';
grant execute on procedure OttieniSceltePreseInCaricoNonEspletate to 'pizzaiolo';
grant execute on procedure EspletaSceltaPresaInCarico to 'pizzaiolo';
grant execute on procedure OttieniInfoProdottiDiScelteInCarico to 'pizzaiolo';

grant execute on procedure OttieniScelteDaPreparare to 'barman';
grant execute on procedure PrendiInCaricoScelta to 'barman';
grant execute on procedure OttieniSceltePreseInCaricoNonEspletate to 'barman';
grant execute on procedure EspletaSceltaPresaInCarico to 'barman';
grant execute on procedure OttieniInfoProdottiDiScelteInCarico to 'barman';

-- fine --

--
--------------------
-- Utenti di test --
```

```
--------------------
--
-- Username: pizzamanag ;;; Password: manage
-- Username: pizzacamer ;;; Password: came
-- Username: pizzapizza ;;; Password: pizza
-- Username: pizzabarma ;;; Password: bar
--

-- call RegistraUtente("pizzamanag","Manager","Principale", "Roma", '1999-10-08',"Roma", "XXXXYYYYZZZZTTTT", "manage", 1);
-- call RegistraUtente("pizzacamer", "Cameriere", "Cognome", "Roma", '2002-01-10', "Milano", "ABCDEFGHILMN0123", "came", 2);
-- call RegistraUtente("pizzapizza", "Pizza", "Yolo", "Milano", '2003-01-20', "Torino", "ZZZZYYYYTTTTXXXX", "pizza", 3);
-- call RegistraUtente("pizzabarma", "Bar", "Man", "Milano", '2003-01-21', "Torino", "ZZZZYYYYTTTTXXXZ", "bar", 4);
```

Codice del Front-End

```
#include "op.h"
#include "mysql_utils.h"
#include "global_config.h"
#include "goodmalloc.h"

typedef struct {
        int num_tavolo;
        mybool is_occupato;
        MYSQL_TIME data_ora_occupazione;
        int num_commensali;
        mybool is_servito_almeno_una_volta;
} situazione_tavolo;

typedef struct {
        char nome_prod[21];
        int num_ord_per_tavolo;
```

```
        int num_sc_per_ord;
} scelta_del_cliente;

typedef struct {
        char nome_prod[21];
        MYSQL_TIME data_ora_occ;
        int num_ord_per_tavolo;
        int num_sc_per_ord;
        int num_t;
} scelta_del_cliente_espletata;

static mybool checked_show_form_action(form_field* fields, int nf) {
        checked_show_form(fields, nf);
        return TRUE;
}

static mybool checked_execute_stmt_action(MYSQL_STMT* stmt) {
        checked_execute_stmt(stmt);
        return TRUE;
}

static mybool __cameriere_agg_ing_extra_alla_scelta_perform(
        MYSQL_TIME* data_ora_occ, int num_ord_per_tav, int num_sc_per_ord,
        const char *ing, double qt_gr) {

        char nome_ing[21] = { 0 };
        memcpy(nome_ing, ing , 20);

        MYSQL_STMT *stmt = init_and_prepare_stmt("call
AggiungiIngExtraAllaScelta(?,?,?,?,?,?)");

        INIT_MYSQL_BIND(params, 6);
        set_inout_param_datetime(0, data_ora_occ, params);
        set_inout_param_int(1, &num_ord_per_tav, params);
```

```
	set_inout_param_int(2, &num_sc_per_ord, params);
	set_in_param_string(3, nome_ing, params);
	set_inout_param_double(4, &qt_gr, params);
	set_in_param_string(5, cfg.username, params);
	bind_param_stmt(stmt, params);

	checked_execute_stmt(stmt);

	close_everything_stmt(stmt);
	return TRUE;
}

static mybool __cameriere_get_situazione_tavolo(
		situazione_tavolo** st, unsigned long long *n_st) {

	*n_st = 0;

	MYSQL_STMT *stmt = init_and_prepare_stmt("call OttieniTavoliDiCompetenza(?)");

	INIT_MYSQL_BIND(params, 1);
	set_in_param_string(0, cfg.username, params);
	bind_param_stmt(stmt, params);

	checked_execute_stmt(stmt);

	*n_st = mysql_stmt_num_rows(stmt);
	good_malloc(*st, situazione_tavolo, *n_st);

	situazione_tavolo myst;
	memset(&myst.data_ora_occupazione, 0, sizeof(myst.data_ora_occupazione));

	INIT_MYSQL_BIND(res, 5);
	set_inout_param_int(0, &myst.num_tavolo, res);
	set_inout_param_int(1, &myst.is_occupato, res);
```

```
        set_inout_param_datetime(2, &myst.data_ora_occupazione, res);
        set_inout_param_int(3, &myst.num_commensali, res);
        set_inout_param_int(4, &myst.is_servito_almeno_una_volta, res);
        bind_result_stmt(stmt, res);

        begin_fetch_stmt(stmt);
        memcpy(&(*st)[i], &myst, sizeof(myst));
    memset(&myst.data_ora_occupazione, 0, sizeof(myst.data_ora_occupazione));
    end_fetch_stmt();

        close_everything_stmt(stmt);
        return TRUE;
}

static mybool __cameriere_prendi_ordinazione_common(mybool close) {
        situazione_tavolo *st = NULL;

        unsigned long long n_st;

        if (!__cameriere_get_situazione_tavolo(&st, &n_st)) {
                return FALSE;
        }

        for (unsigned long long i = 0; i < n_st; ++i) {
                printf("(%llu) --> tavolo: %d, occupato: %s", i + 1,
                        st[i].num_tavolo,
                        mybool_to_str(st[i].is_occupato));

                if (st[i].is_occupato) {
                        printf(", commensali: %d, servito: %s\n", st[i].num_commensali,
                                mybool_to_str(st[i].is_servito_almeno_una_volta));
                } else {
                        puts("");
                }
```

```
}

unsigned long long opt = 0;

form_field fields[1];
int_form_field(fields, 0, "Opzione", 1, 19, &opt);
if(!checked_show_form_action(fields, 1)) {
        good_free(st);
        return FALSE;
}

if(opt < 1 || opt > n_st) {
        puts("scelta non valida");
        good_free(st);
        return FALSE;
}

MYSQL_STMT *stmt = init_and_prepare_stmt(
        close ? "call ChiudiOrdinazione(?,?)" : "call PrendiOrdinazione(?,?)");

INIT_MYSQL_BIND(params, 2);
set_inout_param_datetime(0, &st[opt - 1].data_ora_occupazione, params);
set_in_param_string(1, cfg.username, params);
bind_param_stmt(stmt, params);
if(!checked_execute_stmt_action(stmt)) {
        good_free(st);
        return FALSE;
}

mybool is_ok = TRUE;

if(close) {
        if(mysql_stmt_affected_rows(stmt) == 0) {
                printf("Impossibile chiudere l'ordinazione (opt: %llu)\n",
```

```
                                        opt);
                        is_ok = FALSE;
                }
        }

        good_free(st);
        close_everything_stmt(stmt);
        return is_ok;
}

static mybool __cameriere_get_scelte_del_cliente(
                MYSQL_TIME* data_ora_occ, scelta_del_cliente** sdc,
                unsigned long long *n_sdc) {

        *n_sdc = 0;

        MYSQL_STMT* stmt = init_and_prepare_stmt("call OttieniSceltePerOrdinazione(?,?)");
        INIT_MYSQL_BIND(params, 2);
        set_inout_param_datetime(0, data_ora_occ, params);
        set_in_param_string(1, cfg.username, params);
        bind_param_stmt(stmt, params);

        checked_execute_stmt(stmt);

        *n_sdc = mysql_stmt_num_rows(stmt);
        good_malloc(*sdc, scelta_del_cliente, *n_sdc);

        scelta_del_cliente base;
        memset(&base, 0, sizeof(base));

        INIT_MYSQL_BIND(res_params, 3);
        set_inout_param_int(0, &base.num_ord_per_tavolo, res_params);
        set_inout_param_int(1, &base.num_sc_per_ord, res_params);
        set_out_param_string(2, base.nome_prod, res_params);
```

```
        bind_result_stmt(stmt, res_params);

        begin_fetch_stmt(stmt);
        memcpy(&(*sdc)[i], &base, sizeof(base));
        memset(&base, 0, sizeof(base));
        end_fetch_stmt();

        close_everything_stmt(stmt);
        return TRUE;
}

static mybool __cameriere_get_scelte_espletate(
                scelta_del_cliente_espletata** esp,
                unsigned long long *n_esp) {

        *n_esp = 0;

        MYSQL_STMT *stmt = init_and_prepare_stmt("call OttieniScelteEspletate(?)");
        INIT_MYSQL_BIND(params, 1);
        set_in_param_string(0, cfg.username, params);
        bind_param_stmt(stmt, params);

        checked_execute_stmt(stmt);

        *n_esp = mysql_stmt_num_rows(stmt);
        good_malloc(*esp, scelta_del_cliente_espletata, *n_esp);

        scelta_del_cliente_espletata sdc_esp;
        memset(&sdc_esp, 0, sizeof(sdc_esp));

        INIT_MYSQL_BIND(resp, 5);
        set_inout_param_datetime(0, &sdc_esp.data_ora_occ, resp);
        set_inout_param_int(1, &sdc_esp.num_ord_per_tavolo, resp);
        set_inout_param_int(2, &sdc_esp.num_sc_per_ord, resp);
```

```
        set_inout_param_int(3, &sdc_esp.num_t, resp);
        set_out_param_string(4, sdc_esp.nome_prod, resp);
        bind_result_stmt(stmt, resp);

        begin_fetch_stmt(stmt);
        memcpy(&(*esp)[i], &sdc_esp, sizeof(sdc_esp));
        memset(&sdc_esp, 0, sizeof(sdc_esp));
        end_fetch_stmt();

        close_everything_stmt(stmt);
        return TRUE;
}

static mybool __cameriere_effettua_consegna_perform(
        scelta_del_cliente_espletata* esp, unsigned long long opt) {

        MYSQL_STMT *stmt = init_and_prepare_stmt("call EffettuaConsegna(?,?,?,?)");
        INIT_MYSQL_BIND(params, 4);
        set_inout_param_datetime(0, &(esp->data_ora_occ), params);
        set_inout_param_int(1, &(esp->num_ord_per_tavolo), params);
        set_inout_param_int(2, &(esp->num_sc_per_ord), params);
        set_in_param_string(3, cfg.username, params);
        bind_param_stmt(stmt, params);

        checked_execute_stmt(stmt);

        mybool is_ok = TRUE;
        check_affected_stmt_rows(is_ok, stmt, "Impossibile effettuare la consegna (opt: %llu)\n",
                                                        opt);

        close_everything_stmt(stmt);
        return is_ok;
}
```

```
mybool cameriere_visualizza_situazione_tavoli() {
	situazione_tavolo *st = NULL;
	unsigned long long n_st;

	if(!__cameriere_get_situazione_tavolo(&st, &n_st)) {
		return FALSE;
	}

	for(unsigned long long i = 0; i < n_st; ++i) {
		printf("--> tavolo: %d, occupato: %s",
			st[i].num_tavolo,
			mybool_to_str(st[i].is_occupato));
		if(st[i].is_occupato) {
			printf(", commensali: %d, servito: %s\n",
				st[i].num_commensali,
				mybool_to_str(st[i].is_servito_almeno_una_volta));
		} else {
			puts("");
		}
	}

	good_free(st);
	return TRUE;
}

mybool cameriere_chiudi_ordinazione() {
	return __cameriere_prendi_ordinazione_common(TRUE);
}

mybool cameriere_prendi_ordinazione() {
	return __cameriere_prendi_ordinazione_common(FALSE);
}

mybool cameriere_prendi_scelta_per_ordinazione() {
```

```
situazione_tavolo *st = NULL;
unsigned long long n_st;

if (!__cameriere_get_situazione_tavolo(&st, &n_st)) {
	return FALSE;
}

for (unsigned long long i = 0; i < n_st; ++i) {
	printf("(%llu) --> tavolo: %d, occupato: %s", i + 1,
		st[i].num_tavolo,
		mybool_to_str(st[i].is_occupato));

	if (st[i].is_occupato) {
		printf(", commensali: %d, servito: %s\n", st[i].num_commensali,
			mybool_to_str(st[i].is_servito_almeno_una_volta));
	} else {
		puts("");
	}
}

unsigned long long opt = 0;
char nome_prod[21] = { 0 };

form_field fields[2];
int_form_field(fields, 0, "Opzione", 1, 19, &opt);
string_form_field(fields, 1, "Prodotto", 1, 20, 20, nome_prod);
if(!checked_show_form_action(fields, 2)) {
	good_free(st);
	return FALSE;
}

if(opt < 1 || opt > n_st) {
	puts("scelta non valida");
	good_free(st);
```

```
            return FALSE;
        }

        MYSQL_STMT *stmt = init_and_prepare_stmt("call PrendiSceltaPerOrd(?,?,?)");
        INIT_MYSQL_BIND(params, 3);
        set_inout_param_datetime(0, &st[opt - 1].data_ora_occupazione, params);
        set_in_param_string(1, nome_prod, params);
        set_in_param_string(2, cfg.username, params);
        bind_param_stmt(stmt, params);

        if(!checked_execute_stmt_action(stmt)) {
            good_free(st);
            return FALSE;
        }

        good_free(st);
        return TRUE;
}

mybool cameriere_aggiungi_ing_extra_per_scelta() {
        situazione_tavolo *st = NULL;
        unsigned long long n_st;

        if (!__cameriere_get_situazione_tavolo(&st, &n_st)) {
            return FALSE;
        }

        for (unsigned long long i = 0; i < n_st; ++i) {
            printf("(%llu) --> tavolo: %d, occupato: %s", i + 1,
                st[i].num_tavolo,
                mybool_to_str(st[i].is_occupato));

            if (st[i].is_occupato) {
                printf(", commensali: %d, servito: %s\n", st[i].num_commensali,
```

```
                        mybool_to_str(st[i].is_servito_almeno_una_volta));
        } else {
                puts("");
        }
}

unsigned long long opt = 0;

form_field fields[1];
int_form_field(fields, 0, "Opzione", 1, 19, &opt);
if(!checked_show_form_action(fields, 1)) {
        good_free(st);
        return FALSE;
}

if(opt < 1 || opt > n_st) {
        puts("scelta non valida");
        good_free(st);
        return FALSE;
}

unsigned long long opt_minus_one = opt - 1;

scelta_del_cliente* sdc = NULL;
unsigned long long n_sdc;
if(__cameriere_get_scelte_del_cliente(
        &st[opt_minus_one].data_ora_occupazione, &sdc, &n_sdc) == FALSE) {

        good_free(st);
        return FALSE;
}

if(n_sdc > 0) {
        printf("--> Ordinazione %d\n", sdc[0].num_ord_per_tavolo);
```

```
		for(unsigned long long i = 0; i < n_sdc; ++i) {
			printf("\t** SCELTA %d **\n\t++ Prodotto: %s\n",
				sdc[i].num_sc_per_ord, sdc[i].nome_prod);
		}
}

double q_gr;
char nome_ing[21] = { 0 };
unsigned long long opt_1 = 0;

form_field fields_1[3];
int_form_field(fields_1, 0, "Opzione", 1, 19, &opt_1);
string_form_field(fields_1, 1, "Ingrediente", 1, 20, 20, nome_ing);
double_form_field(fields_1, 2, "Quantita (gr)", 1, 10, &q_gr);
if(!checked_show_form_action(fields_1, 3)) {
		good_free(st);
		good_free(sdc);
		return FALSE;
}

if(opt_1 < 1 || opt_1 > n_sdc){
		puts("scelta non valida");
		good_free(st);
		good_free(sdc);
		return FALSE;
}

unsigned long long opt_1_minus_one = opt_1 - 1;

mybool is_ok = __cameriere_agg_ing_extra_alla_scelta_perform(
		&st[opt_minus_one].data_ora_occupazione,
		sdc[opt_1_minus_one].num_ord_per_tavolo,
		sdc[opt_1_minus_one].num_sc_per_ord, nome_ing, q_gr);
```

```
	good_free(st);
	good_free(sdc);
	return is_ok;
}

mybool cameriere_visualizza_scelte_espletate() {
	scelta_del_cliente_espletata* esp = NULL;
	unsigned long long n_esp;

	if(__cameriere_get_scelte_espletate(&esp, &n_esp) == FALSE) {
		return FALSE;
	}

	for(unsigned long long i = 0; i < n_esp; ++i) {
		printf("Tavolo %d, # ord: %d, # scelta: %d, prodotto: %s\n",
			esp[i].num_t, esp[i].num_ord_per_tavolo, esp[i].num_sc_per_ord,
			esp[i].nome_prod);
	}

	good_free(esp);
	return TRUE;
}

mybool cameriere_effettua_consegna() {
	scelta_del_cliente_espletata* esp = NULL;
	unsigned long long n_esp;

	if(__cameriere_get_scelte_espletate(&esp, &n_esp) == FALSE) {
		return FALSE;
	}

	for(unsigned long long i = 0; i < n_esp; ++i) {
		printf("(%llu) Tavolo %d, # ord: %d, # scelta: %d, prodotto: %s\n",
			i + 1, esp[i].num_t, esp[i].num_ord_per_tavolo,
```

```
                esp[i].num_sc_per_ord, esp[i].nome_prod);
        }

        unsigned long long opt = 0;

        form_field fields[1];
        int_form_field(fields, 0, "Opzione", 1, 19, &opt);
        if(!checked_show_form_action(fields, 1)) {
                good_free(esp);
                return FALSE;
        }

        if(opt < 1 || opt > n_esp) {
                puts("scelta non valida");
                good_free(esp);
                return FALSE;
        }

        mybool is_ok = __cameriere_effettua_consegna_perform(esp, opt);

        good_free(esp);
        return is_ok;
}

mybool cameriere_visualizza_tavoli_assegnati() {
        MYSQL_STMT *stmt = init_and_prepare_stmt("call OttieniTavoliAssegnati(?)");
        INIT_MYSQL_BIND(params, 1);
        set_in_param_string(0, cfg.username, params);
        bind_param_stmt(stmt, params);

        checked_execute_stmt(stmt);

        int num_t;
```

```
	RESET_MYSQL_BIND(params);
	set_inout_param_int(0, &num_t, params);
	bind_result_stmt(stmt, params);

	begin_fetch_stmt(stmt);
	printf("%d\n", num_t);
	end_fetch_stmt();

	close_everything_stmt(stmt);
	return TRUE;
}
#include "op.h"
#include "global_config.h"
#include "goodmalloc.h"
#include "mysql_utils.h"

typedef struct {
	char nome_prod[21];
	MYSQL_TIME tavolo_occupato;
	int num_tavolo;
	int num_ord_per_tavolo;
	int num_sc_per_ord;
} scelta_da_preparare;

static mybool checked_show_form_action(form_field* fields, int nf) {
	checked_show_form(fields, nf);
	return TRUE;
}

static mybool checked_execute_stmt_action(MYSQL_STMT* stmt) {
	checked_execute_stmt(stmt);
	return TRUE;
}
```

```
static mybool __lavoratore_cucina_get_scelte_da_preparare(
        mybool da_esp, scelta_da_preparare **sdp_out, unsigned long long *n_sdp_out) {

    *n_sdp_out = 0;

    MYSQL_STMT* stmt = init_and_prepare_stmt(
        da_esp ? "call OttieniSceltePreseInCaricoNonEspletate(?)" : "call
OttieniScelteDaPreparare()");

    INIT_MYSQL_BIND(esp_params, 1);

    if(da_esp) {
        set_in_param_string(0, cfg.username, esp_params);
        bind_param_stmt(stmt, esp_params);
    }

    checked_execute_stmt(stmt);

    *n_sdp_out = mysql_stmt_num_rows(stmt);
    good_malloc(*sdp_out, scelta_da_preparare, *n_sdp_out);

    scelta_da_preparare sdp;
    memset(&sdp, 0, sizeof(sdp));

    INIT_MYSQL_BIND(params, 5);
    set_inout_param_datetime(0, &sdp.tavolo_occupato, params);
    set_inout_param_int(1, &sdp.num_tavolo, params);
    set_inout_param_int(2, &sdp.num_ord_per_tavolo, params);
    set_inout_param_int(3, &sdp.num_sc_per_ord, params);
    set_out_param_string(4, sdp.nome_prod, params);
    bind_result_stmt(stmt, params);

    begin_fetch_stmt(stmt);
    memcpy(&(*sdp_out)[i], &sdp, sizeof(sdp));
```

```
	memset(&sdp, 0, sizeof(sdp));
	end_fetch_stmt();

	close_everything_stmt(stmt);
	return TRUE;
}

static mybool __lavoratore_cucina_visualizza_scelte_da_preparare_oresp_common(mybool esp) {
	scelta_da_preparare *sdp = NULL;
	unsigned long long n_sdp;
	if(!__lavoratore_cucina_get_scelte_da_preparare(esp, &sdp, &n_sdp)) {
		return FALSE;
	}

	for(unsigned long long i = 0; i < n_sdp; ++i) {
		printf("--> Tavolo: %d, # ord: %d, # sc per ord: %d, prodotto: %s\n",
			sdp[i].num_tavolo, sdp[i].num_ord_per_tavolo, sdp[i].num_sc_per_ord,
			sdp[i].nome_prod);
	}

	good_free(sdp);
	return TRUE;
}

mybool lavoratore_cucina_prendi_in_carico_scelta_da_preparare() {
	scelta_da_preparare *sdp = NULL;
	unsigned long long n_sdp;
	if(!__lavoratore_cucina_get_scelte_da_preparare(FALSE, &sdp, &n_sdp)) {
		return FALSE;
	}

	for(unsigned long long i = 0; i < n_sdp; ++i) {
		printf("(%llu) Tavolo: %d, # ord: %d, # sc per ord: %d, prodotto: %s\n",
			i + 1, sdp[i].num_tavolo, sdp[i].num_ord_per_tavolo,
```

```
                sdp[i].num_sc_per_ord,  sdp[i].nome_prod);
}

unsigned long long opt = 0;

form_field fields[1];
int_form_field(fields, 0, "Opzione", 1, 19, &opt);
if(!checked_show_form_action(fields, 1)) {
        good_free(sdp);
        return FALSE;
}

if(opt < 1 || opt > n_sdp) {
        puts("opzione non valida");
        good_free(sdp);
        return FALSE;
}

int opt_minus_one = opt - 1;

MYSQL_STMT *stmt = init_and_prepare_stmt("call PrendiInCaricoScelta(?,?,?,?,?)");
INIT_MYSQL_BIND(params, 5);
set_inout_param_datetime(0, &sdp[opt_minus_one].tavolo_occupato, params);
set_inout_param_int(1, &sdp[opt_minus_one].num_ord_per_tavolo, params);
set_inout_param_int(2, &sdp[opt_minus_one].num_sc_per_ord, params);
set_in_param_string(3, sdp[opt_minus_one].nome_prod, params);
set_in_param_string(4, cfg.username, params);
bind_param_stmt(stmt, params);

if(!checked_execute_stmt_action(stmt)) {
        good_free(sdp);
        return FALSE;
}
```

```
	mybool is_ok = TRUE;
	check_affected_stmt_rows(is_ok, stmt,
		"non è stato possibile prendere in carico la scelta (opt: %llu)\n",
		opt);

	good_free(sdp);
	close_everything_stmt(stmt);
	return is_ok;
}

mybool lavoratore_cucina_espleta_scelta() {
	scelta_da_preparare *sdp = NULL;
	unsigned long long n_sdp;
	if(!__lavoratore_cucina_get_scelte_da_preparare(TRUE, &sdp, &n_sdp)) {
		return FALSE;
	}

	for(unsigned long long i = 0; i < n_sdp; ++i) {
		printf("(%llu) Tavolo: %d, # ord: %d, # sc per ord: %d, prodotto: %s\n",
			i + 1, sdp[i].num_tavolo, sdp[i].num_ord_per_tavolo,
			sdp[i].num_sc_per_ord,  sdp[i].nome_prod);
	}

	unsigned long long opt = 0;

	form_field fields[1];
	int_form_field(fields, 0, "Opzione", 1, 19, &opt);
	if(!checked_show_form_action(fields, 1)) {
		good_free(sdp);
		return FALSE;
	}

	if(opt < 1 || opt > n_sdp) {
		puts("opzione non valida");
```

```
            good_free(sdp);
            return FALSE;
        }

        int opt_minus_one = opt - 1;

        MYSQL_STMT* stmt = init_and_prepare_stmt("call EspletaSceltaPresaInCarico(?,?,?,?)");

        INIT_MYSQL_BIND(params, 4);
        set_inout_param_datetime(0, &sdp[opt_minus_one].tavolo_occupato, params);
        set_inout_param_int(1, &sdp[opt_minus_one].num_ord_per_tavolo, params);
        set_inout_param_int(2, &sdp[opt_minus_one].num_sc_per_ord, params);
        set_in_param_string(3, cfg.username, params);
        bind_param_stmt(stmt, params);

        if(!checked_execute_stmt_action(stmt)) {
            good_free(sdp);
            return FALSE;
        }

        mybool is_ok = TRUE;
        check_affected_stmt_rows(is_ok, stmt,
            "non è stato possibile espletare la scelta (opt: %llu)\n", opt);

        good_free(sdp);
        close_everything_stmt(stmt);
        return is_ok;
}

mybool lavoretore_cucina_visualizza_info_scelte_prese_in_carico() {
        int num_tavolo;
        int num_ord_per_tavolo;
        int num_sc_per_ord;
        char nome_prod[21] = { 0 };
```

```
    char nome_ing[21] = { 0 };
    double qt_in_gr;
    my_bool ing_is_null;

    MYSQL_STMT *stmt =
init_and_prepare_stmt("call OttieniInfoProdottiDiScelteInCarico(?)");

    INIT_MYSQL_BIND(params_in, 1);
    set_in_param_string(0, cfg.username, params_in);
    bind_param_stmt(stmt, params_in);

    checked_execute_stmt(stmt);

    INIT_MYSQL_BIND(params, 6);
    set_inout_param_int(0, &num_tavolo, params);
    set_inout_param_int(1, &num_ord_per_tavolo, params);
    set_inout_param_int(2, &num_sc_per_ord, params);
    set_out_param_string(3, nome_prod, params);
    set_out_param_maybe_null_string(4, nome_ing, &ing_is_null, params);
    set_inout_param_double(5, &qt_in_gr, params);
    bind_result_stmt(stmt, params);

    begin_fetch_stmt(stmt);
    printf("--> Tavolo: %d, # ord: %d, # sc per ord: %d\n\t"
                    "* Prodotto: %s", num_tavolo, num_ord_per_tavolo,
                    num_sc_per_ord, nome_prod);

    if(!ing_is_null) {
            printf("\n\t* Ingrediente extra: %s\n\t* Quantita in gr: %lf gr\n",
                            nome_ing, qt_in_gr);
    } else {
            puts("");
    }
```

```
	memset(nome_prod, 0, sizeof(nome_prod));
	memset(nome_ing, 0, sizeof(nome_ing));
	end_fetch_stmt();

	close_everything_stmt(stmt);
	return TRUE;
}

mybool lavoratore_cucina_visualizza_scelte_da_preparare() {
	return __lavoratore_cucina_visualizza_scelte_da_preparare_oresp_common(FALSE);
}

mybool lavoratore_cucina_visualizza_scelte_presa_in_carico_da_espletare() {
	return __lavoratore_cucina_visualizza_scelte_da_preparare_oresp_common(TRUE);
}

#include <linux/limits.h>

#include "global_config.h"
#include "mybool.h"
#include "menu_entries.h"
#include "mysql_utils.h"
#include "parse_dbms_conn_config.h"
#include "macros.h"
#include "role.h"

static mybool set_menu_based_on_role(role r) {
	if(r == ROLE_UNKNOWN)
		return FALSE;

	if(r == ROLE_MANAGER) {
		cfg.menu_entries = entries_manager;
		cfg.menu_entries_len = ENTRIES_LEN_MANAGER;
   } else if(r == ROLE_CAMERIERE) {
```

```
                cfg.menu_entries = entries_cameriere;
                cfg.menu_entries_len = ENTRIES_LEN_CAMERIERE;
    } else if (r == ROLE_BARMAN || r == ROLE_PIZZAIOLO) {
                cfg.menu_entries = entries_barman_e_pizzaiolo;
                cfg.menu_entries_len = ENTRIES_LEN_BARMAN_E_PIZZAIOLO;
    }

        return TRUE;
}

static void reparse_and_change_user(role r, const char* users_dir) {
        const char* what;

        if(r == ROLE_MANAGER)
                what = MANAGER_JSON_FILE;
        else if(r == ROLE_BARMAN)
                what = BARMAN_JSON_FILE;
        else if(r == ROLE_PIZZAIOLO)
                what = PIZZAIOLO_JSON_FILE;
        else if(r == ROLE_CAMERIERE)
                what = CAMERIERE_JSON_FILE;
        else {
                printf("*** (login.c:reparse_and_change_user) ERRORE: valore sconosciuto r=%d\n",
r);
                close_and_exit(EXIT_FAILURE);
        }

        char json_file_path[PATH_MAX + 2] = {0};
        snprintf(json_file_path, PATH_MAX + 1, "%s/%s", users_dir, what);

        dbms_conn_config dbms_conf;
        if (parse_dbms_conn_config(json_file_path, &dbms_conf) == FALSE) {
                printf("impossibile parsare il file json (%s)\n", json_file_path);
                close_and_exit(EXIT_FAILURE);
```

```
        }

    if (mysql_change_user(cfg.db_conn,
                                                dbms_conf.db_username,
                                                dbms_conf.db_password,
                                                dbms_conf.db_name)) {
        MYSQL_BASIC_PRINTERROR_EXIT("mysql_change_user");
    }

        free_dbms_conn_config(dbms_conf);
}

mybool attempt_login(const char* password, const char* users_dir) {
        role r = ROLE_UNKNOWN;

        MYSQL_STMT* stmt = init_and_prepare_stmt("call TentaLogin(?,?,?)");

        INIT_MYSQL_BIND(params, 3);
        set_in_param_string(0, cfg.username, params);
        set_in_param_string(1, (char*) password, params);
        set_inout_param_tinyint(2, (int*)&r, params);
        bind_param_stmt(stmt, params);

        if(!execute_stmt(stmt)) {
                close_only_stmt(stmt);
                return FALSE;
        }

        RESET_MYSQL_BIND(params);
        set_inout_param_int(0, (int*)&r, params);
        bind_result_stmt(stmt, params);

        if(mysql_stmt_fetch(stmt)) {
                MYSQL_STMT_BASIC_PRINTERROR_EXIT("mysql_stmt_fetch", stmt);
```

```
	}

	close_everything_stmt(stmt);

	mybool login = set_menu_based_on_role(r);

	if(login == TRUE) {
		reparse_and_change_user(r, users_dir);
	}

	return login;
}

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <signal.h>
#include <linux/limits.h>

#include "parse_dbms_conn_config.h"
#include "mysql_utils.h"
#include "global_config.h"
#include "macros.h"

#ifndef USERS_DIR_DFL
#define USERS_DIR_DFL "./users"
#endif

#define OPT_WITH_ARG(arg, set) \
	if(strcmp(argv[i], arg) == 0) { \
		int i_plus_one = i + 1; \
		if(i_plus_one == argc) { \
			fprintf(stderr, "errore: %s richiede un argomento\n", arg); \
			print_help_and_exit(EXIT_FAILURE, "usage corretto"); \
```

```
                } \
                set = argv[i + 1]; \
                i = i_plus_one; \
                continue; \
        }

#define OPT_HELP_AND_EXIT() \
        if(strcmp(argv[i], "--help") == 0) { \
                print_help_and_exit(EXIT_SUCCESS, "help"); \
        }

config cfg;

mybool attempt_login(const char* password, const char* users_dir);

void print_help_and_exit(int code, const char* msg) {
        FILE* f = stdout;

        if(code != EXIT_SUCCESS) {
                f = stderr;
        }

        fprintf(f,
                "%s\n\t--username <username>\n\t--password <password>\n\t[--users
<users_dir>]\n\t[--help]\n",
                msg);

        exit(code);
}

void close_and_exit(int code) {
        mysql_close(cfg.db_conn);
        mysql_library_end();
        exit(code);
```

```
}

void handler_close_before_exit() {
	close_and_exit(EXIT_SUCCESS);
}

int main(int argc, char** argv)  {
	cfg.username = NULL;
	char* password = NULL;
	char* users_dir = USERS_DIR_DFL;

	for(int i = 1; i < argc; ++i) {
		OPT_HELP_AND_EXIT();
		OPT_WITH_ARG("--username", cfg.username);
		OPT_WITH_ARG("--password", password);
		OPT_WITH_ARG("--users", users_dir);
		printf("warning -- %s: opzione sconosciuta\n", argv[i]);
	}

	if(cfg.username == NULL || password == NULL) {
		print_help_and_exit(EXIT_FAILURE, "username e password devono essere
specificati");
	}

	char json_file_path[PATH_MAX + 2] = { 0 };
	snprintf(json_file_path, PATH_MAX + 1, "%s/%s", users_dir, LOGIN_JSON_FILE);

	dbms_conn_config dbms_conf;
	if(parse_dbms_conn_config(json_file_path, &dbms_conf) == FALSE) {
		printf("impossibile parsare il file json (%s)\n", json_file_path);
		exit(EXIT_FAILURE);
	}

	if(dbms_conf.db_port < 0 || dbms_conf.db_port > 65535) {
```

```
            puts("port range invalido (valido: 0 <= port <= 65535)");
            exit(EXIT_FAILURE);
        }

        signal(SIGINT, SIG_IGN);
        signal(SIGTERM, SIG_IGN);

        if((cfg.db_conn = mysql_init(NULL)) == NULL) {
            puts("impossibile iniziallizzare la libreria mysql (mysql_init)");
            exit(EXIT_FAILURE);
        }

        signal(SIGINT, (void (*)(int)) handler_close_before_exit);
        signal(SIGTERM, (void (*)(int)) handler_close_before_exit);

        printf("tentativo di connessione: %s in %s@%s:%d...\n",
            dbms_conf.db_name,
            dbms_conf.db_username,
            dbms_conf.db_hostname,
            dbms_conf.db_port);

        if(mysql_real_connect(cfg.db_conn,
                                    dbms_conf.db_hostname,
                                    dbms_conf.db_username,
                                    dbms_conf.db_password,
                                    dbms_conf.db_name,
                                    dbms_conf.db_port,
                                    NULL,
                                    CLIENT_MULTI_RESULTS                    |
CLIENT_MULTI_STATEMENTS) == NULL) {
            MYSQL_BASIC_PRINTERROR_EXIT("mysql_real_connect");
        }

        free_dbms_conn_config(dbms_conf);
```

```
	if(mysql_autocommit(cfg.db_conn, FALSE)) {
		MYSQL_BASIC_PRINTERROR_EXIT("mysql_autocommit");
	}

	if(attempt_login(password, users_dir) == FALSE) {
		puts("credenziali per l'accesso al sistema non valide");
		close_and_exit(EXIT_FAILURE);
	}

	while(show_menu());

	close_and_exit(EXIT_SUCCESS);
}
#include "mysql_utils.h"
#include "op.h"
#include "goodmalloc.h"

typedef struct {
	char comune_residenza[35];
	char comune_nascita[35];
	char nome[21];
	char cognome[21];
	char cf[17];
	char username[11];
	MYSQL_TIME data_nascita;
	role ruolo;
} utente;

typedef struct {
	int giorno;
	int mese;
	int anno;
	int ora;
```

```
        int minuto;
} orario;

typedef struct {
        orario inizio;
        orario fine;
} turno;

typedef struct {
        int num_tavolo;
        int num_max_commensali;
        mybool is_occupato;
        mybool is_attivo;
} tavolo;

typedef struct {
        int id_fiscale;
        MYSQL_TIME data_ora_emissione;
        double costo_totale;
} scontrino;

typedef struct {
        MYSQL_TIME data_ora_occupazione;
        int num_t;
} tavolo_scontrino_stampabile;

typedef struct {
        unsigned long long idx_cameriere;
        unsigned long long idx_tavolo;
        unsigned long long idx_turno;
} __assegna_turno_choice;

static mybool checked_show_form_action(form_field *fields, int nf) {
        checked_show_form(fields, nf);
```

```
	return TRUE;
}

static mybool checked_execute_stmt_action(MYSQL_STMT *stmt) {
	checked_execute_stmt(stmt);
	return TRUE;
}

static mybool __manager_get_tavoli(tavolo **out_t, unsigned long long *n) {
	*n = 0;

	MYSQL_STMT *stmt = init_and_prepare_stmt("call OttieniTavoli()");
	checked_execute_stmt(stmt);

	tavolo t;

	INIT_MYSQL_BIND(params, 4);
	set_inout_param_int(0, &t.num_tavolo, params);
	set_inout_param_int(1, &t.num_max_commensali, params);
	set_inout_param_int(2, &t.is_occupato, params); /*MYBOOL*/
	set_inout_param_int(3, &t.is_attivo, params); /*MYBOOL*/
	bind_result_stmt(stmt, params);

	*n = mysql_stmt_num_rows(stmt);
	good_malloc(*out_t , tavolo, *n);

	begin_fetch_stmt(stmt);
	memcpy(&(*out_t)[i], &t, sizeof(t));
	end_fetch_stmt();

	close_everything_stmt(stmt);
	return TRUE;
}
```

```
static mybool __manager_assegna_turno_perform(tavolo* ta, MYSQL_TIME* tu, utente *u) {
	MYSQL_STMT *stmt = init_and_prepare_stmt("call AssegnaTurno(?,?,?)");

	INIT_MYSQL_BIND(params, 3);
	set_inout_param_smallint(0, &(ta->num_tavolo), params);
	set_inout_param_datetime(1, tu, params);
	set_in_param_string(2, u->username, params);
	bind_param_stmt(stmt, params);

	checked_execute_stmt(stmt);

	close_everything_stmt(stmt);
	return TRUE;
}

static void __manager_assegna_turno_print(tavolo *ta,
	MYSQL_TIME *tu_inizio, MYSQL_TIME *tu_fine,
	utente *u, unsigned long long n_ta, unsigned long long n_tu,
	unsigned long long n_u) {

	puts("TAVOLI");
	for(unsigned long long i = 0; i < n_ta; ++i) {
		printf("%llu) tavolo: %d, max commensali: %d\n",
			i + 1, ta[i].num_tavolo, ta[i].num_max_commensali);
	}

	puts("\nUTENTI");
	for(unsigned long long i = 0; i < n_u; ++i) {
		printf("%llu) %s %s (%s)\n",
			i + 1, u[i].nome, u[i].cognome,
			role_to_str(u[i].ruolo));
	}

	puts("\nTURNI");
```

```
	for(unsigned long long i = 0; i < n_tu; ++i) {
		printf("%llu) inizio: %d/%d/%d %d:%d -- fine: %d/%d/%d %d:%d\n",
			i + 1, tu_inizio[i].day, tu_inizio[i].month,
			tu_inizio[i].year, tu_inizio[i].hour, tu_inizio[i].minute,
			tu_fine[i].day, tu_fine[i].month, tu_fine[i].year,
			tu_fine[i].hour, tu_fine[i].minute);
	}
}

static mybool __manager_assegna_turno_readchoice(__assegna_turno_choice* choice) {
	form_field fields[3];
	int_form_field(fields, 0, "Tavolo", 1, 5, &(choice->idx_tavolo));
	int_form_field(fields, 1, "Cameriere", 1, 19, &(choice->idx_cameriere));
	int_form_field(fields, 2, "Turno", 1, 19, &(choice->idx_turno));

	checked_show_form(fields, 3);

	return TRUE;
}

static mybool __manager_visualizza_entrate_common_resultset(mybool monthly) {
	MYSQL_STMT *stmt = init_and_prepare_stmt("call OttieniEntrate(?)");

	INIT_MYSQL_BIND(params, 1);
	set_inout_param_tinyint(0, &monthly, params);
	bind_param_stmt(stmt, params);

	checked_execute_stmt(stmt);

	int num_tot_scontrini;
	double tot_pagato_scontrini = 0;
	INIT_MYSQL_BIND(rs_tot, 2);
	set_inout_param_int(0, &num_tot_scontrini, rs_tot);
	set_inout_param_double(1, &tot_pagato_scontrini, rs_tot);
```

```
bind_result_stmt(stmt, rs_tot);

begin_fetch_stmt(stmt); //oneshot
end_fetch_stmt();

printf("***TOTALE SCONTRINI EMESSI E PAGATI: %d\n"
                "***COSTO TOTALE (INCASSO): %lf\n"
                "+-------------+\n",
        num_tot_scontrini, tot_pagato_scontrini);

next_result_stmt(stmt); // prossimo result set

int id_fiscale_scontrino;
MYSQL_TIME data_ora_emissione_scontrino;
double costo_totale_scontrino;
INIT_MYSQL_BIND(rs_each, 3);
set_inout_param_int(0, &id_fiscale_scontrino, rs_each);
set_inout_param_datetime(1, &data_ora_emissione_scontrino, rs_each);
set_inout_param_double(2, &costo_totale_scontrino, rs_each);
bind_result_stmt(stmt, rs_each);

begin_fetch_stmt(stmt); //mul res
printf("--> Id fiscale: %d\n\t"
                "* Data e ora emissione: "
                "%d/%d/%d %d:%d:%d\n\t"
                "* Totale scontrino: %lf\n",
                id_fiscale_scontrino,
                data_ora_emissione_scontrino.day,
                data_ora_emissione_scontrino.month,
                data_ora_emissione_scontrino.year,
                data_ora_emissione_scontrino.hour,
                data_ora_emissione_scontrino.minute,
                data_ora_emissione_scontrino.second,
                costo_totale_scontrino);
```

```
        end_fetch_stmt();

        close_everything_stmt(stmt);
        return TRUE;
}

static mybool __manager_get_turni(
                MYSQL_TIME** out_inizio,
                MYSQL_TIME** out_fine,
                unsigned long long* n) {

        *n = 0;

        MYSQL_STMT *stmt = init_and_prepare_stmt("call OttieniTurni()");
        checked_execute_stmt(stmt);

        MYSQL_TIME inizio;
        MYSQL_TIME fine;
        INIT_MYSQL_BIND(params, 2);
        set_inout_param_datetime(0, &inizio, params);
        set_inout_param_datetime(1, &fine, params);
        bind_result_stmt(stmt, params);

        *n = mysql_stmt_num_rows(stmt);
        good_malloc(*out_inizio, MYSQL_TIME, *n);
        good_malloc(*out_fine, MYSQL_TIME, *n);

        begin_fetch_stmt(stmt);
        (*out_inizio)[i] = inizio;
        (*out_fine)[i] = fine;
        end_fetch_stmt();

   close_everything_stmt(stmt);
   return TRUE;
```

```
}

static mybool __manager_get_utenti(utente** out_u, unsigned long long *n) {
	*n = 0;

	MYSQL_STMT *stmt = init_and_prepare_stmt("call OttieniUtenti()");
	checked_execute_stmt(stmt);

	utente u;
	memset(&u, 0, sizeof(u));

	INIT_MYSQL_BIND(params, 8);
	set_out_param_string(0, u.username, params);
  set_out_param_string(1, u.nome, params);
  set_out_param_string(2, u.cognome, params);
  set_out_param_string(3, u.cf, params);
  set_out_param_string(4, u.comune_residenza, params);
  set_out_param_string(5, u.comune_nascita, params);
  set_inout_param_date(6, &u.data_nascita, params);
  set_inout_param_int(7, &u.ruolo, params);
	bind_result_stmt(stmt, params);

  *n = mysql_stmt_num_rows(stmt);
  good_malloc(*out_u, utente, *n);

  begin_fetch_stmt(stmt);
	memcpy(&(*out_u)[i], &u, sizeof(u));
	memset(&u, 0, sizeof(u));
	end_fetch_stmt();

  close_everything_stmt(stmt);
  return TRUE;
}
```

```
#define NE_TIME(x, y) \
	( \
		(x.day != y.day) || \
		(x.month != y.month) || \
		(x.year != y.year) || \
		(x.hour != y.hour) || \
		(x.minute != y.minute) \
	)

static mybool __manager_visualizza_turni_common_resultset(const char* query) {
	MYSQL_STMT *stmt = init_and_prepare_stmt(query);
	checked_execute_stmt(stmt);

	MYSQL_TIME turno_inizio;
	MYSQL_TIME turno_fine;
	int num_tavolo;
	char nome[21] = { 0 };
	char cognome[21] = { 0 };
	char username[11] = { 0 };
	my_bool num_tavolo_null;
	my_bool nome_null;
	my_bool cognome_null;
	my_bool username_null;

	INIT_MYSQL_BIND(params, 6);
	set_inout_param_datetime(0, &turno_inizio, params);
	set_inout_param_datetime(1, &turno_fine, params);
	set_out_param_maybe_null_int(2, &num_tavolo, &num_tavolo_null, params);
	set_out_param_maybe_null_string(3, nome, &nome_null, params);
	set_out_param_maybe_null_string(4, cognome, &cognome_null, params);
	set_out_param_maybe_null_string(5, username, &username_null, params);
	bind_result_stmt(stmt, params);

	MYSQL_TIME turno_inizio_cur;
```

```
	memset(&turno_inizio_cur, 0, sizeof(turno_inizio_cur));

	begin_fetch_stmt(stmt);

	if(NE_TIME(turno_inizio_cur, turno_inizio)) {
		memcpy(&turno_inizio_cur, &turno_inizio, sizeof(turno_inizio));
		puts("*****");
		printf("turno inizio: %d/%d/%d %d:%d -- fine: %d/%d/%d %d:%d\n",
			turno_inizio.day, turno_inizio.month, turno_inizio.year,
			turno_inizio.hour, turno_inizio.minute, turno_fine.day,
			turno_fine.month, turno_fine.year, turno_fine.hour,
			turno_fine.minute);
	}

	if(num_tavolo_null && nome_null && cognome_null && username_null)
		printf("\tnessun tavolo attivato e cameriere assegnato\n");
	else
		printf("\ttavolo %d servito da %s %s (%s)\n", num_tavolo, nome, cognome,
username);

	memset(nome, 0, sizeof(nome));
	memset(cognome, 0, sizeof(cognome));
	memset(username, 0, sizeof(username));

	end_fetch_stmt();

	close_everything_stmt(stmt);
	return TRUE;
}

#undef NE_TIME

static mybool __manager_get_scontrini_non_pagati(scontrino** scont, unsigned long long *n_scont)
{
```

```
	*n_scont = 0;

	MYSQL_STMT *stmt = init_and_prepare_stmt("call OttieniScontriniNonPagati()");
	checked_execute_stmt(stmt);

	*n_scont = mysql_stmt_num_rows(stmt);
	good_malloc(*scont, scontrino, *n_scont);

	scontrino s;

	INIT_MYSQL_BIND(params, 3);
	set_inout_param_int(0, &s.id_fiscale, params);
	set_inout_param_datetime(1, &s.data_ora_emissione, params);
	set_inout_param_double(2, &s.costo_totale, params);
	bind_result_stmt(stmt, params);

	begin_fetch_stmt(stmt);
	(*scont)[i].id_fiscale = s.id_fiscale;
	(*scont)[i].data_ora_emissione = s.data_ora_emissione;
	(*scont)[i].costo_totale = s.costo_totale;
	end_fetch_stmt();

	close_everything_stmt(stmt);
	return TRUE;
}

static void __manager_stampa_scontrino_tavolo_occupato_print(MYSQL_STMT* stmt) {
	int num_ord;
	int num_sc_per_ord;
	char nome_prod[21] = { 0 };
	char nome_ing[21] = { 0 };
	double qt_in_gr;
	my_bool ing_is_null;
```

```
INIT_MYSQL_BIND(params, 5);
set_inout_param_int(0, &num_ord, params);
set_inout_param_int(1, &num_sc_per_ord, params);
set_out_param_string(2, nome_prod, params);
set_out_param_maybe_null_string(3, nome_ing, &ing_is_null, params);
set_inout_param_double(4, &qt_in_gr, params);
bind_result_stmt(stmt, params);

begin_fetch_stmt(stmt);
printf(" --> # ord: %d, # sc per ord: %d\n\t* %s\n",
        num_ord, num_sc_per_ord, nome_prod);

if(!ing_is_null) {
        printf("\t* Extra ingrediente: %s (%lf gr)\n", nome_ing, qt_in_gr);
}
end_fetch_stmt();

next_result_stmt(stmt);

int id_fiscale;
MYSQL_TIME data_ora_emissione;
double costo_totale;

RESET_MYSQL_BIND(params);
set_inout_param_int(0, &id_fiscale, params);
set_inout_param_datetime(1, &data_ora_emissione, params);
set_inout_param_double(2, &costo_totale, params);
bind_result_stmt(stmt, params);

begin_fetch_stmt(stmt);
end_fetch_stmt();

printf("***** SCONTRINO %d %d/%d/%d %d:%d:%d *****\n"
"***** COSTO TOTALE: %lf *****\n"
```

```
        "***** FINE SCONTRINO *****\n",
                id_fiscale, data_ora_emissione.day,
                data_ora_emissione.month, data_ora_emissione.year,
                data_ora_emissione.hour, data_ora_emissione.minute,
                data_ora_emissione.second, costo_totale);
}

static mybool __manager_get_tavoli_scontrino_stampabile(
                tavolo_scontrino_stampabile** tss,
                unsigned long long *n_tss) {

        *n_tss = 0;

        MYSQL_STMT *stmt = init_and_prepare_stmt("call OttieniTavoliScontrinoStampabile()");

        checked_execute_stmt(stmt);

        *n_tss = mysql_stmt_num_rows(stmt);
        good_malloc(*tss, tavolo_scontrino_stampabile, *n_tss);

        tavolo_scontrino_stampabile tss_base;
        memset(&tss_base, 0, sizeof(tss_base));

        INIT_MYSQL_BIND(params, 2);
        set_inout_param_int(0, &tss_base.num_t, params);
        set_inout_param_datetime(1, &tss_base.data_ora_occupazione, params);
        bind_result_stmt(stmt, params);

        begin_fetch_stmt(stmt);
        memcpy(&(*tss)[i], &tss_base, sizeof(tss_base));
        memset(&tss_base, 0, sizeof(tss_base));
        end_fetch_stmt();

        close_everything_stmt(stmt);
```

```
        return TRUE;
}

static mybool __manager_stampa_scontrino_tavolo_occupato_perform(
                tavolo_scontrino_stampabile* tss) {
        MYSQL_STMT *stmt = init_and_prepare_stmt("call StampaScontrino(?)");
        INIT_MYSQL_BIND(params, 1);
        set_inout_param_datetime(0, &(tss->data_ora_occupazione), params);
        bind_param_stmt(stmt, params);

        checked_execute_stmt(stmt);

        __manager_stampa_scontrino_tavolo_occupato_print(stmt);

        close_everything_stmt(stmt);
        return TRUE;
}

mybool manager_crea_nuovo_utente() {
        char nome[21] = { 0 };
   char cognome[21] = { 0 };
   char cf[17] = { 0 };
   char comuneResidenza[35] = { 0 };
   char comuneNascita[35] = { 0 };
   char username[11] = {0};
   char password[46] = {0};
   char ruolo[10] = {0};
   int giornoNascita;
   int meseNascita;
   int annoNascita;

        form_field fields[11];
        string_form_field(fields, 0, "Nome", 1, 20, 20, nome);
   string_form_field(fields, 1, "Cognome", 1, 20, 20, cognome);
```

```
string_form_field(fields, 2, "Codice fiscale", 16, 16, 16, cf);
string_form_field(fields, 3, "Comune di residenza", 1, 34, 34, comuneResidenza);
string_form_field(fields, 4, "Comune di nascita", 1, 34, 34, comuneNascita);
	int_form_field(fields, 5, "Giorno di nascita", 1, 2, &giornoNascita);
	int_form_field(fields, 6, "Mese di nascita", 1, 2, &meseNascita);
	int_form_field(fields, 7, "Anno di nascita", 4, 4, &annoNascita);
	string_form_field(fields, 8, "Username", 1, 10, 10, username);
	string_form_field(fields, 9, "Password", 1, 45, 45, password);
	string_form_field(fields, 10, "Ruolo", 6, 9, 9, ruolo);

	checked_show_form(fields, 11);
	checked_date_check(giornoNascita, meseNascita, annoNascita);

	role r;
	if((r = str_to_role(ruolo)) == ROLE_UNKNOWN) {
		puts("ruolo sconosciuto. possibili valori:\n"
				" * manager\n"
				" * cameriere\n"
				" * pizzaiolo\n"
				" * barman");
		puts("operazione annullata.");
		return FALSE;
	}

	MYSQL_STMT* stmt = init_and_prepare_stmt("call RegistraUtente(?,?,?,?,?,?,?,?,?)");

	INIT_MYSQL_TIME_ONLYDATE(nascita, giornoNascita, meseNascita, annoNascita);

	INIT_MYSQL_BIND(params, 9);
	set_in_param_string(0, username, params);
set_in_param_string(1, nome, params);
set_in_param_string(2, cognome, params);
set_in_param_string(3, comuneResidenza, params);
	set_inout_param_date(4, &nascita, params);
```

```
	set_in_param_string(5, comuneNascita, params);
	set_in_param_string(6, cf, params);
	set_in_param_string(7, password, params);
	set_inout_param_tinyint(8, (int*)&r, params);
	bind_param_stmt(stmt, params);

	checked_execute_stmt(stmt);

	close_everything_stmt(stmt);
	return TRUE;
}

mybool manager_ripristina_password_utente_esistente() {
	char username[11] = { 0 };
	char newpasswd[46] = { 0 };

	form_field fields[2];
	string_form_field(fields, 0, "Username", 1, 10, 10, username);
	string_form_field(fields, 1, "Nuova password", 1, 45, 45, newpasswd);

	checked_show_form(fields, 2);

	MYSQL_STMT *stmt = init_and_prepare_stmt("call RipristinoPassword(?,?)");
	INIT_MYSQL_BIND(params, 2);
	set_in_param_string(0, username, params);
	set_in_param_string(1, newpasswd, params);
	bind_param_stmt(stmt, params);

	checked_execute_stmt(stmt);

	mybool is_ok = TRUE;
	check_affected_stmt_rows(is_ok, stmt, "non esiste alcun utente che si chiami \"%s\"\n"
                                                                "password non
alterate.\n", username);
```

```
	close_everything_stmt(stmt);
	return is_ok;
}

mybool manager_aggiungi_nuovo_tavolo() {
	int numero_tavolo;
	int max_commensali;

	form_field fields[2];
	int_form_field(fields, 0, "Numero del tavolo", 1, 5, &numero_tavolo);
   int_form_field(fields, 1, "Max commensali", 1, 3, &max_commensali);

	checked_show_form(fields, 2);

	MYSQL_STMT *stmt = init_and_prepare_stmt("call AggiungiNuovoTavolo(?,?)");
	INIT_MYSQL_BIND(params, 2);
	set_inout_param_smallint(0, &numero_tavolo, params);
	set_inout_param_tinyint(1, &max_commensali, params);
	bind_param_stmt(stmt, params);

	checked_execute_stmt(stmt);

	close_everything_stmt(stmt);
	return TRUE;
}

mybool manager_aggiungi_nuovo_ingrediente() {
	char nome[21] = { 0 };
	int disp_iniziale;
	double costo_al_kg;

	form_field fields[3];
	string_form_field(fields, 0, "Nome", 1, 20, 20, nome);
```

```
        int_form_field(fields, 1, "Disponibilità iniziale", 1, 19, &disp_iniziale);
        double_form_field(fields, 2, "Costo al kg", 1, 10, &costo_al_kg);

        checked_show_form(fields, 3);

    MYSQL_STMT *stmt = init_and_prepare_stmt("call AggiungiNuovoIngrediente(?,?,?)");
    INIT_MYSQL_BIND(params, 3);
    set_in_param_string(0, nome, params);
    set_inout_param_int(1, &disp_iniziale, params);
        set_inout_param_double(2, &costo_al_kg, params);
    bind_param_stmt(stmt, params);

    checked_execute_stmt(stmt);

    close_everything_stmt(stmt);
    return TRUE;
}

mybool manager_aggiungi_prodotto_del_menu() {
        char nome[21] = { 0 };
        double costo_unitario;
        mybool is_bar_menu;
        mybool is_alcolico;

        form_field fields[3];
        string_form_field(fields, 0, "Nome", 1, 20, 20, nome);
        double_form_field(fields, 1, "Costo unitario", 1, 10, &costo_unitario);
        mybool_form_field(fields, 2, "Fa parte del menu bar?", &is_bar_menu);
        checked_show_form(fields, 3);

        if(is_bar_menu) {
                form_field fields_alcolico[1];
                mybool_form_field(fields_alcolico, 0, "Alcolico?", &is_alcolico);
                checked_show_form(fields_alcolico, 1);
```

```
        }

    MYSQL_STMT *stmt = init_and_prepare_stmt("call AggiungiProdottoNelMenu(?,?,?,?)");
        INIT_MYSQL_BIND(params, 4);
        set_in_param_string(0, nome, params);
        set_inout_param_double(1, &costo_unitario, params);
        set_inout_param_mybool(2, &is_bar_menu, params);

        if(is_bar_menu) {
                set_inout_param_mybool(3, &is_alcolico, params);
        } else {
                set_in_param_null(3, params);
        }

        bind_param_stmt(stmt, params);

        checked_execute_stmt(stmt);

        close_everything_stmt(stmt);
    return TRUE;
}

mybool manager_associa_prodotto_e_ingrediente() {
        char nome_prodotto[21] = { 0 };
        char nome_ingrediente[21] = { 0 };

        form_field fields[2];
        string_form_field(fields, 0, "Nome del prodotto", 1, 20, 20, nome_prodotto);
    string_form_field(fields, 1, "Nome dell'ingrediente", 1, 20, 20, nome_ingrediente);
        checked_show_form(fields, 2);

        MYSQL_STMT* stmt = init_and_prepare_stmt("call AssociaProdottoAIngrediente(?,?)");
        INIT_MYSQL_BIND(params, 2);
        set_in_param_string(0, nome_prodotto, params);
```

```
        set_in_param_string(1, nome_ingrediente, params);
        bind_param_stmt(stmt, params);

        checked_execute_stmt(stmt);

        close_everything_stmt(stmt);
        return TRUE;
}

mybool manager_crea_turno() {
        turno t;

        form_field fields[10];
        int_form_field(fields, 0, "Giorno inizio", 1, 2, &t.inizio.giorno);
   int_form_field(fields, 1, "Mese inizio", 1, 2, &t.inizio.mese);
   int_form_field(fields, 2, "Anno inizio", 4, 4, &t.inizio.anno);
   int_form_field(fields, 3, "Ora inizio", 1, 2, &t.inizio.ora);
   int_form_field(fields, 4, "Minuto inizio", 1, 2, &t.inizio.minuto);

   int_form_field(fields, 5, "Giorno fine", 1, 2, &t.fine.giorno);
   int_form_field(fields, 6, "Mese fine", 1, 2, &t.fine.mese);
   int_form_field(fields, 7, "Anno fine", 4, 4, &t.fine.anno);
   int_form_field(fields, 8, "Ora fine", 1, 2, &t.fine.ora);
   int_form_field(fields, 9, "Minuto fine", 1, 2, &t.fine.minuto);

        checked_show_form(fields, 10);

        checked_date_check(t.inizio.giorno, t.inizio.mese, t.inizio.anno);
   checked_date_check(t.fine.giorno, t.fine.mese, t.fine.anno);

        INIT_MYSQL_TIME_DATETIME(start_dt,
                t.inizio.giorno, t.inizio.mese,
                t.inizio.anno, t.inizio.ora,
                t.inizio.minuto, 0);
```

```
    INIT_MYSQL_TIME_DATETIME(end_dt,
                    t.fine.giorno, t.fine.mese,
        t.fine.anno, t.fine.ora,
                    t.fine.minuto, 0);

        MYSQL_STMT *stmt = init_and_prepare_stmt("call AggiungiTurno(?,?)");
        INIT_MYSQL_BIND(params, 2);
        set_inout_param_datetime(0, &start_dt, params);
        set_inout_param_datetime(1, &end_dt, params);
        bind_param_stmt(stmt, params);

    checked_execute_stmt(stmt);

    close_everything_stmt(stmt);
    return TRUE;
}

mybool manager_rimuovi_prodotto_del_menu() {
        char nome_prodotto[21] = { 0 };

        form_field fields[1];
        string_form_field(fields, 0, "Nome del prodotto", 1, 20, 20, nome_prodotto);
        checked_show_form(fields, 1);

        MYSQL_STMT *stmt = init_and_prepare_stmt("call RimuoviProdottoNelMenu(?)");
        INIT_MYSQL_BIND(params, 1);
        set_in_param_string(0, nome_prodotto, params);
        bind_param_stmt(stmt, params);

        checked_execute_stmt(stmt);

        mybool is_ok = TRUE;
        check_affected_stmt_rows(is_ok, stmt, "il prodotto \"%s\" non esiste.\n", nome_prodotto);
```

```
	close_everything_stmt(stmt);
	return is_ok;
}

mybool manager_rimuovi_ingrediente() {
	char nome_ingrediente[21] = {0};

	form_field fields[1];
	string_form_field(fields, 0, "Nome dell'ingrediente", 1, 20, 20, nome_ingrediente);
	checked_show_form(fields, 1);

	MYSQL_STMT *stmt = init_and_prepare_stmt("call RimuoviIngrediente(?)");
	INIT_MYSQL_BIND(params, 1);
	set_in_param_string(0, nome_ingrediente, params);
	bind_param_stmt(stmt, params);

	checked_execute_stmt(stmt);

	mybool is_ok = TRUE;
	check_affected_stmt_rows(is_ok, stmt, "l'ingrediente \"%s\" non esiste.\n",
nome_ingrediente);

   close_everything_stmt(stmt);
   return is_ok;
}

mybool manager_rimuovi_prodotto_e_ingrediente() {
	char nome_prodotto[21] = { 0 };
	char nome_ingrediente[21] = { 0 };

	form_field fields[2];
	string_form_field(fields, 0, "Nome del prodotto", 1, 20, 20, nome_prodotto);
	string_form_field(fields, 1, "Nome dell'ingrediente", 1, 20, 20, nome_ingrediente);
```

```
        checked_show_form(fields, 2);

        MYSQL_STMT *stmt = init_and_prepare_stmt("call
RimuoviAssocProdottoEIngrediente(?,?)");
        INIT_MYSQL_BIND(params, 2);
        set_in_param_string(0, nome_prodotto, params);
        set_in_param_string(1, nome_ingrediente, params);
        bind_param_stmt(stmt, params);

        checked_execute_stmt(stmt);

        mybool is_ok = TRUE;
        check_affected_stmt_rows(is_ok, stmt, "l'associazione (\"%s\",\"%s\") non esiste.\n",
                nome_prodotto, nome_ingrediente);

   close_everything_stmt(stmt);
   return is_ok;
}

mybool manager_visualizza_turni() {
        MYSQL_TIME *turno_inizio = NULL;
        MYSQL_TIME *turno_fine = NULL;
        unsigned long long n_turni;
        if(!__manager_get_turni(&turno_inizio, &turno_fine, &n_turni)) {
                return FALSE;
        }

        for(unsigned long long i = 0; i < n_turni; ++i) {
      printf("inizio: %d/%d/%d %d:%d -- fine: %d/%d/%d %d:%d\n",
            turno_inizio[i].day, turno_inizio[i].month,
            turno_inizio[i].year, turno_inizio[i].hour,
            turno_inizio[i].minute, turno_fine[i].day,
            turno_fine[i].month, turno_fine[i].year,
            turno_fine[i].hour, turno_fine[i].minute);
```

```
    }

        good_free(turno_inizio);
        good_free(turno_fine);
        return TRUE;
}

mybool manager_visualizza_utenti() {
        unsigned long long n_ut;
        utente *ut = NULL;

        if(!__manager_get_utenti(&ut, &n_ut)) {
                return FALSE;
        }

        for(unsigned long long i = 0; i < n_ut; ++i) {
                printf("\"%s\" [%s]: %s %s residente a: %s, nato a: %s, il: %d/%d/%d (%s)\n",
                        ut[i].username, role_to_str(ut[i].ruolo),ut[i].nome, ut[i].cognome,
                        ut[i].comune_residenza,ut[i].comune_nascita, ut[i].data_nascita.day,
                        ut[i].data_nascita.month, ut[i].data_nascita.year, ut[i].cf);
        }

        good_free(ut);
        return TRUE;
}

mybool manager_visualizza_tavoli() {
        unsigned long long n_t;
        tavolo *t = NULL;

        if(!__manager_get_tavoli(&t, &n_t)) {
                return FALSE;
        }
```

```
	for(unsigned long long i = 0; i < n_t; ++i) {
		printf("tavolo: %d, max comm: %d, occupato: %s, attivo: %s\n",
			t[i].num_tavolo, t[i].num_max_commensali,
			mybool_to_str(t[i].is_occupato), mybool_to_str(t[i].is_attivo));
	}

	good_free(t);
	return TRUE;
}

mybool manager_assegna_turno() {
	tavolo *ta = NULL;
	utente *u = NULL;
	MYSQL_TIME *tu_inizio = NULL;
	MYSQL_TIME *tu_fine = NULL;

	unsigned long long n_ta;
	unsigned long long n_u;
	unsigned long long n_tu;

	if(!__manager_get_tavoli(&ta, &n_ta)) {
		return FALSE;
	}

	if(!__manager_get_utenti(&u, &n_u)) {
		good_free(ta);
		return FALSE;
	}

	if(!__manager_get_turni(&tu_inizio, &tu_fine, &n_tu)) {
		good_free(ta);
		good_free(u);
		return FALSE;
	}
```

```
__manager_assegna_turno_print(ta, tu_inizio, tu_fine, u, n_ta, n_tu, n_u);

if(n_ta == 0 || n_tu == 0 || n_u == 0) {
        puts("una o più informazioni necessarie per l'assegnazione"
        " del turno non sono presenti, è necessario aggiungerle.");
        good_free(ta);
        good_free(u);
        good_free(tu_inizio);
        good_free(tu_fine);
        return FALSE;
}

mybool is_ok = FALSE;

__assegna_turno_choice choice;
memset(&choice, 0, sizeof(choice));

if((is_ok = __manager_assegna_turno_readchoice(&choice))) {
        if((choice.idx_tavolo < 1 || choice.idx_tavolo > n_ta) ||
                (choice.idx_turno < 1 || choice.idx_turno > n_tu) ||
                (choice.idx_cameriere < 1 || choice.idx_cameriere > n_u)) {
                puts("opzione non valida");
                good_free(ta);
                good_free(u);
                good_free(tu_inizio);
                good_free(tu_fine);
                return FALSE;
        }

        is_ok = __manager_assegna_turno_perform(
                &(ta[choice.idx_tavolo - 1]),
                &(tu_inizio[choice.idx_turno - 1]),
                &(u[choice.idx_cameriere - 1]));
```

```
	}

	good_free(ta);
	good_free(u);
	good_free(tu_inizio);
	good_free(tu_fine);
	return is_ok;
}

mybool manager_visualizza_turni_assegnati() {
	return __manager_visualizza_turni_common_resultset("call OttieniTurniAssegnati()");
}

mybool manager_visualizza_turno_attuale() {
	return __manager_visualizza_turni_common_resultset("call OttieniTurnoAttuale()");
}

mybool manager_visualizza_menu() {
	MYSQL_STMT *stmt = init_and_prepare_stmt("call OttieniMenu()");
	checked_execute_stmt(stmt);

	char nome[21] = { 0 };
	double costo_unitario;
	mybool is_menu_bar;
	mybool is_alcolico;

	INIT_MYSQL_BIND(params, 4);
	set_out_param_string(0, nome, params);
	set_inout_param_int(1, &is_menu_bar, params);
	set_inout_param_int(2, &is_alcolico, params);
	set_inout_param_double(3, &costo_unitario, params);
	bind_result_stmt(stmt, params);

	mybool first_cycle = TRUE;
```

```
        mybool is_menu_bar_cur = FALSE;

        begin_fetch_stmt(stmt);
        if(first_cycle) {
                if(is_menu_bar == is_menu_bar_cur)
                        puts("---PIZZERIA---");

                first_cycle = FALSE;
        }

        if(is_menu_bar != is_menu_bar_cur) {
                puts("---BAR---");
                is_menu_bar_cur = is_menu_bar;
        }

    const char* alcolico = is_menu_bar && is_alcolico ? "[ALCOLICO] " : "";
    printf("\t%s%s costo %lf\n", alcolico, nome, costo_unitario);
        memset(nome, 0, sizeof(nome));
        end_fetch_stmt();

        close_everything_stmt(stmt);
        return TRUE;
}

mybool manager_visualizza_situazione_ingredienti() {
        MYSQL_STMT *stmt = init_and_prepare_stmt("call OttieniIngredienti()");
        checked_execute_stmt(stmt);

        char nome[21] = { 0 };
        int disp_scorte;
        double costo_al_kg;

        INIT_MYSQL_BIND(params, 3);
        set_out_param_string(0, nome, params);
```

```
	set_inout_param_int(1, &disp_scorte, params);
	set_inout_param_double(2, &costo_al_kg, params);
	bind_result_stmt(stmt, params);

	begin_fetch_stmt(stmt);
	printf("\t%s, %d unità disponibili, %lf / kg\n", nome, disp_scorte, costo_al_kg);

	memset(nome, 0, sizeof(nome));
	end_fetch_stmt();

	close_everything_stmt(stmt);
	return TRUE;
}

mybool manager_visualizza_assoc_prodotti_ingredienti() {
	MYSQL_STMT *stmt = init_and_prepare_stmt("call OttieniComposizioneProdotto()");
	checked_execute_stmt(stmt);

	char nome_prod[21] = { 0 };
	char nome_ing[21] = { 0 };

	INIT_MYSQL_BIND(params, 2);
	set_out_param_string(0, nome_prod, params);
	set_out_param_string(1, nome_ing, params);
	bind_result_stmt(stmt, params);

	char nome_prod_cur[21] = { 0 };

	begin_fetch_stmt(stmt);
	if(strcmp(nome_prod_cur, nome_prod)) {
		memcpy(nome_prod_cur, nome_prod, sizeof(nome_prod));
		printf("\tPRODOTTO %s COMPOSTO DA:\n", nome_prod);
	}
```

```
        printf("\t\t* %s\n", nome_ing);

        memset(nome_prod, 0, sizeof(nome_prod));
        memset(nome_ing, 0, sizeof(nome_ing));
        end_fetch_stmt();

        close_everything_stmt(stmt);
        return TRUE;
}

mybool manager_inc_disp_ingrediente() {
        char nome_ing[21] = { 0 };
        int inc_disp;

        form_field fields[2];
        string_form_field(fields, 0, "Nome ingrediente", 1, 20, 20, nome_ing);
        int_form_field(fields, 1, "Incremento disponibilita", 1, 19, &inc_disp);
        checked_show_form(fields, 2);

        MYSQL_STMT* stmt = init_and_prepare_stmt("call IncDispIngrediente(?,?)");

        INIT_MYSQL_BIND(params, 2);
        set_in_param_string(0, nome_ing, params);
        set_inout_param_int(1, &inc_disp, params);
        bind_param_stmt(stmt, params);

        checked_execute_stmt(stmt);

        mybool is_ok = TRUE;
        check_affected_stmt_rows(is_ok, stmt, "non esiste alcun ingrediente \"%s\"\n", nome_ing);

        close_everything_stmt(stmt);
        return is_ok;
}
```

```
mybool manager_visualizza_entrate_giornaliere() {
	return __manager_visualizza_entrate_common_resultset(FALSE);
}

mybool manager_visualizza_entrate_mensili() {
	return __manager_visualizza_entrate_common_resultset(TRUE);
}

mybool manager_visualizza_scontrini_non_pagati() {
	scontrino *sct = NULL;
	unsigned long long n_sct;
	if(!__manager_get_scontrini_non_pagati(&sct, &n_sct)) {
		return FALSE;
	}

	for(unsigned long long i = 0; i < n_sct; ++i) {
		printf("--> Id fiscale: %d\n\t"
				"* Emissione: %d/%d/%d %d:%d:%d\n\t"
				"* Costo totale: %lf\n",
				sct[i].id_fiscale,
				sct[i].data_ora_emissione.day,
				sct[i].data_ora_emissione.month,
				sct[i].data_ora_emissione.year,
				sct[i].data_ora_emissione.hour,
				sct[i].data_ora_emissione.minute,
				sct[i].data_ora_emissione.second,
				sct[i].costo_totale);
   }

	good_free(sct);
	return TRUE;
}
```

```
mybool manager_contrassegna_scontrino_pagato() {
	scontrino *sct = NULL;
	unsigned long long n_sct;
	if(!__manager_get_scontrini_non_pagati(&sct, &n_sct)) {
		return FALSE;
	}

	if(n_sct == 0) {
		puts("nessuno scontrino presente");
		good_free(sct);
		return FALSE;
	}

	for(unsigned long long i = 0; i < n_sct; ++i) {
		printf("--> (%llu) Id fiscale: %d\n\t"
				"* Emissione: %d/%d/%d %d:%d:%d\n\t"
				"* Costo totale: %lf\n",
				i + 1,
				sct[i].id_fiscale,
				sct[i].data_ora_emissione.day,
				sct[i].data_ora_emissione.month,
				sct[i].data_ora_emissione.year,
				sct[i].data_ora_emissione.hour,
				sct[i].data_ora_emissione.minute,
				sct[i].data_ora_emissione.second,
				sct[i].costo_totale);
  }

	unsigned long long opt = 0;

	form_field fields[1];
	int_form_field(fields, 0, "Scontrino", 1, 19, &opt);
	if(checked_show_form_action(fields, 1) == FALSE) {
		good_free(sct);
```

```
            return FALSE;
        }

        if(opt < 1 || opt > n_sct) {
            puts("opzione non valida");
            good_free(sct);
            return FALSE;
        }

        MYSQL_STMT* stmt = init_and_prepare_stmt("call ContrassegnaScontrinoPagato(?)");

        INIT_MYSQL_BIND(params, 1);
        set_inout_param_int(0, &(sct[opt - 1].id_fiscale), params);
        bind_param_stmt(stmt, params);

        if(checked_execute_stmt_action(stmt) == FALSE) {
            good_free(sct);
            return FALSE;
        }

        mybool is_ok = TRUE;
        check_affected_stmt_rows(is_ok, stmt,
            "non è stato possibile aggiornare lo scontrino indicato con %llu\n"
            "\tRagioni:\n\t* Lo scontrino è gia"
            " stato pagato\n", opt);

        good_free(sct);
        close_everything_stmt(stmt);
        return is_ok;
}

mybool manager_assegna_tavolo_a_cliente() {
        char nome[21] = { 0 };
        char cognome[21] = { 0 };
```

```
	int num_comm;

	form_field fields[3];
	string_form_field(fields, 0, "Nome", 1, 20, 20, nome);
	string_form_field(fields, 1, "Cognome", 1, 20, 20, cognome);
	int_form_field(fields, 2, "Commensali", 1, 19, &num_comm);
	checked_show_form(fields, 3);

	MYSQL_STMT *stmt = init_and_prepare_stmt("call AssegnaTavoloACliente(?,?,?)");

	INIT_MYSQL_BIND(params, 3);
	set_in_param_string(0, nome, params);
	set_in_param_string(1, cognome, params);
	set_inout_param_int(2, &num_comm, params);
	bind_param_stmt(stmt, params);

	checked_execute_stmt(stmt);

	int num_tavolo_trovato;
	RESET_MYSQL_BIND(params);
	set_inout_param_int(0, &num_tavolo_trovato, params);
	bind_result_stmt(stmt, params);

	begin_fetch_stmt(stmt);
	end_fetch_stmt();

	printf("Tavolo assegnato: %d\n", num_tavolo_trovato);

	close_everything_stmt(stmt);
	return TRUE;
}

mybool manager_visualizza_tavoli_poss_stampare_scontrino() {
	tavolo_scontrino_stampabile *tss = NULL;
```

```
	unsigned long long n_tss;
	if(!__manager_get_tavoli_scontrino_stampabile(&tss, &n_tss)) {
		return FALSE;
	}

	for(unsigned long long i = 0; i < n_tss; ++i) {
		printf("--> Tavolo: %d\n", tss[i].num_t);
	}

	good_free(tss);
	return TRUE;
}

mybool manager_stampa_scontrino_tavolo_occupato() {
	tavolo_scontrino_stampabile *tss = NULL;
	unsigned long long n_tss;
	if(!__manager_get_tavoli_scontrino_stampabile(&tss, &n_tss)) {
		return FALSE;
	}

	for(unsigned long long i = 0; i < n_tss; ++i) {
		printf("(%llu) Tavolo: %d\n", i + 1, tss[i].num_t);
	}

	unsigned long long opt = 0;

	form_field fields[1];
	int_form_field(fields, 0, "Opzione", 1, 19, &opt);
	if(!checked_show_form_action(fields, 1)) {
		good_free(tss);
		return FALSE;
	}

	if(opt < 1 || opt > n_tss) {
```

```
                puts("opzione non valida");
                good_free(tss);
                return FALSE;
        }

        mybool is_ok = __manager_stampa_scontrino_tavolo_occupato_perform(&(tss[opt - 1]));

        good_free(tss);
        return is_ok;
}
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include "myezcvt.h"

mybool cvt_str_to_int(const char* in, int* out)  {
        char *endptr;
        *out = strtol(in, &endptr, 10);

        return *endptr == 0;
}

mybool cvt_str_to_double(const char* in, double* out) {
        char *endptr;
        *out = strtod(in, &endptr);

        return (in + strlen(in)) == endptr;
}

mybool cvt_str_yesno_to_mybool(const char* in, mybool* out) {
        size_t in_len = strlen(in);

        if(in_len > 3)
                return FALSE;
```

```
	char lowerin[3] = { 0 };
	for(size_t i = 0; i < in_len; ++i)
		lowerin[i] = tolower(in[i]);

	if(strcmp("yes", lowerin) == 0 || strcmp("y", lowerin) == 0) {
		*out = TRUE;
		return TRUE;
	} else if(strcmp("no", lowerin) == 0 || strcmp("n", lowerin) == 0) {
		*out = FALSE;
		return TRUE;
	}

	return FALSE;
}
#include "mysql_utils.h"

MYSQL_STMT* init_and_prepare_stmt(const char* query) {
	MYSQL_STMT* stmt = mysql_stmt_init(cfg.db_conn);
	if(stmt == NULL) {
		close_and_exit(EXIT_FAILURE);
	}

	if(mysql_stmt_prepare(stmt, query, strlen(query))) {
		MYSQL_STMT_BASIC_PRINTERROR_EXIT("mysql_stmt_prepare", stmt);
	}

	return stmt;
}

mybool execute_stmt(MYSQL_STMT* stmt) {
  if (mysql_stmt_execute(stmt)) {
	  MYSQL_STMT_BASIC_PRINTERROR("mysql_stmt_execute", stmt);
	  return FALSE;
```

```
 }

 if (mysql_stmt_store_result(stmt)) {
        MYSQL_STMT_BASIC_PRINTERROR_EXIT("mysql_stmt_store_result", stmt);
 }

 return TRUE;
}

void close_everything_stmt(MYSQL_STMT* stmt) {
        //consuma tutti i result set non consumati
        int code;
        while(!(code=mysql_stmt_next_result(stmt))) {}

        if(code > 0) {
                MYSQL_STMT_BASIC_PRINTERROR_EXIT("mysql_stmt_next_result", stmt);
        }
        //tutti i result set sono stati consumati

        if(mysql_stmt_close(stmt)) {
                MYSQL_BASIC_PRINTERROR_EXIT("mysql_stmt_close");
        }
}
#define MAX_LINE 1024

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

#include "global_config.h"
#include "goodmalloc.h"
#include "myutils.h"
#include "myezcvt.h"

typedef struct {
```

```
  char* buf;
  int buf_len;
} line;

static const char* roles[] =
{
  "unknown",
  "manager",
  "cameriere",
  "pizzaiolo",
  "barman"
};

static line read_line() {
  line ln;
  good_malloc(ln.buf, char, MAX_LINE + 2);

  fgets(ln.buf, MAX_LINE, stdin);

  int efflen = strlen(ln.buf) - 1;
  ln.buf[efflen] = 0;

  ln.buf_len = efflen;

  return ln;
}

static void any_key() {
  printf("\nPress any key to continue...");
  getchar();
}

mybool show_menu() {
  clear();
```

```
printf("---Menu principale (%s)---\n", cfg.username);

for (int i = 0; i < cfg.menu_entries_len; ++i) {
  printf("(%d) %s\n", i + 1, cfg.menu_entries[i].entry);
}

int entries_len_plus_one = cfg.menu_entries_len + 1;
printf("(%d) Exit application\n", entries_len_plus_one);

int choice;

printf("> ");
line ln = read_line();
if (cvt_str_to_int(ln.buf, &choice) == FALSE) {
  good_free(ln.buf);
  return TRUE;
}

good_free(ln.buf);

if (choice < 1 || choice > entries_len_plus_one) {
  return TRUE;
}

if (choice == entries_len_plus_one) {
  return FALSE;
}

clear();

int choice_minus_one = choice - 1;
printf(">>> %s\n", cfg.menu_entries[choice_minus_one].entry);

mybool res = cfg.menu_entries[choice_minus_one].handler();
```

```
  if (res == TRUE) {
    puts(">>> OK");
  } else {
    puts(">>> FAIL");
  }

  any_key();

  return TRUE;
}

#define INVALID(x)                    \
  if (x == FALSE) {                   \
    puts("il dato inserito non è valido"); \
    good_free(curln.buf);             \
    return FALSE;                     \
  }

mybool show_form(const form_field* fields,
                 int fields_len) {
  puts("---Completa il form---");

  for (int i = 0; i < fields_len; ++i) {
    form_field field = fields[i];

    printf("--> %s: ", field.field_name);
    line curln = read_line();

    if (curln.buf_len > field.expected_max_len ||
        curln.buf_len < field.expected_min_len) {
      printf("field \"%s\" expects %d as max length and %d as min length.\n",
             field.field_name, field.expected_max_len, field.expected_min_len);

      good_free(curln.buf);
```

```
    return FALSE;
   }

   if (field.output_type == STRING) {
    memcpy(field.output, curln.buf, field.output_size);
   } else if (field.output_type == INTEGER) {
    INVALID(cvt_str_to_int(curln.buf, (int*)field.output));
   } else if (field.output_type == DOUBLE) {
    INVALID(cvt_str_to_double(curln.buf, (double*)field.output));
   } else if (field.output_type == MYBOOL) {
    INVALID(cvt_str_yesno_to_mybool(curln.buf, (mybool*)field.output));
   }

   good_free(curln.buf);
  }

  mybool is_ok = FALSE;

  printf("Va bene? [y/n]: ");
  line ln = read_line();
  cvt_str_yesno_to_mybool(ln.buf, &is_ok);
  good_free(ln.buf);

  return is_ok;
}

#undef INVALID

mybool date_check(int day, int month, int year) {
        if (year >= 1900 && year <= 9999) {
                if (month >= 1 && month <= 12) {
                if ((day >= 1 && day <= 31) &&
                (month == 1 || month == 3 || month == 5 || month == 7 || month == 8 ||
                month == 10 || month == 12))
```

```
                return TRUE;
                else if ((day >= 1 && day <= 30) &&
                (month == 4 || month == 6 || month == 9 || month == 11))
                return TRUE;
                else if ((day >= 1 && day <= 28) && (month == 2))
                return TRUE;
                else if (day == 29 && month == 2 &&
                (year % 400 == 0 || (year % 4 == 0 && year % 100 != 0)))
                return TRUE;
                else
                return FALSE;
        } else {
                        return FALSE;
                }
        }

 return FALSE;
}

role str_to_role(char* role) {
  for(size_t i = 0; i < strlen(role); ++i) {
    role[i] = tolower(role[i]);
  }

  if(strcmp(role, "manager") == 0) {
    return ROLE_MANAGER;
  } else if(strcmp(role, "barman") == 0) {
    return ROLE_BARMAN;
  } else if(strcmp(role, "pizzaiolo") == 0) {
    return ROLE_PIZZAIOLO;
  } else if(strcmp(role, "cameriere") == 0) {
    return ROLE_CAMERIERE;
  }
```

```
  return ROLE_UNKNOWN;
}

const char* role_to_str(role r) {
    return roles[(int)r];
}
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "parse_dbms_conn_config.h"
#include "jsmn.h"

#define BUFF_SIZE 4096

static char config[BUFF_SIZE];

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(const 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;
}

mybool parse_dbms_conn_config(const char *path, dbms_conn_config *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("(parse_dbms_conn_config) failed to parse JSON: %d\n", r);
		return FALSE;
	}

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

```
        return FALSE;
}

/* 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->db_hostname =
                        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) {
                char* endptr;
                conf->db_port = strtol(config + t[i + 1].start, &endptr, 10);
                if(endptr != config + t[i + 1].end) {
                        puts("(parse_dbms_conn_config) expecting integer");
                        return FALSE;
                }
                i++;
        } else if (jsoneq(config, &t[i], "database") == 0) {
                conf->db_name =
                        strndup(config + t[i + 1].start, t[i + 1].end - t[i + 1].start);
                i++;
        } else {
                printf("(parse_dbms_conn_config) unexpected key: %.*s\n", t[i].end - t[i].start,
                        config + t[i].start);
        }
```

```
        }

        return TRUE;
}

#ifndef GLOBAL_CONFIG_H
#define GLOBAL_CONFIG_H

#include <mysql.h>
#include "myutils.h"

typedef struct {
        char* username;
        MYSQL* db_conn;
        menu_entry* menu_entries;
        int menu_entries_len;
} config;

extern config cfg;
void close_and_exit(int code);

#endif
#ifndef GOODMALLOC_H
#define GOODMALLOC_H

#include <stdlib.h>

#define good_free(ptr) \
  if (ptr != NULL) {   \
    free(ptr);         \
    ptr = NULL;        \
  }

#define good_malloc(ptr, type, size)                    \
```

```
 if ((ptr = (type*)malloc((size) * sizeof(type))) == NULL) { \
   fputs("memory exhausted", stderr);                    \
   exit(EXIT_FAILURE);                                   \
 }

#endif/*
 * MIT License
 *
 * Copyright (c) 2010 Serge Zaitsev
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * of this software and associated documentation files (the "Software"), to deal
 * in the Software without restriction, including without limitation the rights
 * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
 * copies of the Software, and to permit persons to whom the Software is
 * furnished to do so, subject to the following conditions:
 *
 * The above copyright notice and this permission notice shall be included in
 * all copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
 * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
 * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
 * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
 * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
 * SOFTWARE.
 */
```

```
#ifndef JSMN_H
#define JSMN_H

#include <stddef.h>

#ifdef __cplusplus
extern "C" {
#endif

#ifdef JSMN_STATIC
#define JSMN_API static
#else
#define JSMN_API extern
#endif

/**
 * 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 jsmntok {
  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 jsmn_parser {
  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;

/**
 * Create JSON parser over an array of tokens
 */
```

```
JSMN_API void jsmn_init(jsmn_parser *parser);

/**
 * Run JSON parser. It parses a JSON data string into and array of tokens, each
 * describing
 * a single JSON object.
 */
JSMN_API int jsmn_parse(jsmn_parser *parser, const char *js, const size_t len,
                        jsmntok_t *tokens, const unsigned int num_tokens);

#ifndef JSMN_HEADER
/**
 * Allocates a fresh unused token from the token pool.
 */
static jsmntok_t *jsmn_alloc_token(jsmn_parser *parser, jsmntok_t *tokens,
                                   const 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, const jsmntype_t type,
                            const int start, const 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,
                                const size_t len, jsmntok_t *tokens,
                                const 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;
    default:
                /* to quiet a warning from gcc*/
      break;
```

```
    }
    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,
```

```
                        const size_t len, jsmntok_t *tokens,
                        const 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.
 */
JSMN_API int jsmn_parse(jsmn_parser *parser, const char *js, const size_t len,
                        jsmntok_t *tokens, const 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) {
        jsmntok_t *t = &tokens[parser->toksuper];
#ifdef JSMN_STRICT
        /* In strict mode an object or array can't become a key */
        if (t->type == JSMN_OBJECT) {
```

```
        return JSMN_ERROR_INVAL;
      }
#endif
      t->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) {
        const 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.
 */
JSMN_API void jsmn_init(jsmn_parser *parser) {
  parser->pos = 0;
  parser->toknext = 0;
  parser->toksuper = -1;
}

#endif /* JSMN_HEADER */
```

```
#ifdef __cplusplus
}
#endif

#endif /* JSMN_H */
#ifndef MACROS_H
#define MACROS_H

#define LOGIN_JSON_FILE "login.json"
#define PIZZAIOLO_JSON_FILE "pizzaiolo.json"
#define BARMAN_JSON_FILE "barman.json"
#define CAMERIERE_JSON_FILE "cameriere.json"
#define MANAGER_JSON_FILE "manager.json"

#endif
#ifndef MENU_ENTRIES_H
#define MENU_ENTRIES_H

#include "op.h"
#include "myutils.h"

#define ENTRIES_LEN_MANAGER 27
#define ENTRIES_LEN_BARMAN_E_PIZZAIOLO 5
#define ENTRIES_LEN_CAMERIERE 8

menu_entry entries_manager[ENTRIES_LEN_MANAGER] =
{
   {
      "Crea nuovo utente",
      manager_crea_nuovo_utente
   },
   {
      "Ripristina password utente esistente",
```

```
    manager_ripristina_password_utente_esistente
  },
  {
    "Aggiungi un nuovo tavolo",
    manager_aggiungi_nuovo_tavolo
  },
  {
    "Aggiungi un nuovo ingrediente",
    manager_aggiungi_nuovo_ingrediente
  },
  {
    "Incrementa disponibilita ingrediente",
    manager_inc_disp_ingrediente
  },
  {
    "Aggiungi prodotto del menu",
    manager_aggiungi_prodotto_del_menu
  },
  {
    "Associa prodotto e ingrediente",
    manager_associa_prodotto_e_ingrediente
  },
  {
    "Aggiungi nuovo turno",
    manager_crea_turno
  },
  {
    "Rimuovi un prodotto dal menu",
    manager_rimuovi_prodotto_del_menu
  },
  {
    "Rimuovi un ingrediente",
    manager_rimuovi_ingrediente
  },
```

```
{
  "Rimuovi una associazione prodotto e ingrediente",
  manager_rimuovi_prodotto_e_ingrediente
},
{
  "Visualizza turni",
  manager_visualizza_turni
},
{
  "Visualizza utenti registrati",
  manager_visualizza_utenti
},
{
  "Visualizza situazione tavoli",
  manager_visualizza_tavoli
},
{
  "Assegna un turno a cameriere",
  manager_assegna_turno
},
{
  "Visualizza turno attuale",
  manager_visualizza_turno_attuale
},
{
  "Visualizza turni assegnati",
  manager_visualizza_turni_assegnati
},
{
  "Visualizza menu",
  manager_visualizza_menu
},
{
  "Visualizza disponibilita ingredienti",
```

```
    manager_visualizza_situazione_ingredienti
  },
  {
    "Visualizza composizione dei prodotti nel menu",
    manager_visualizza_assoc_prodotti_ingredienti
  },
  {
    "Visualizza entrate mensili",
    manager_visualizza_entrate_mensili
  },
  {
    "Visualizza entrate giornaliere",
    manager_visualizza_entrate_giornaliere
  },
  {
    "Visualizza scontrini stampati ma non pagati",
    manager_visualizza_scontrini_non_pagati
  },
  {
    "Contrassegna scontrino come pagato",
    manager_contrassegna_scontrino_pagato
  },
  {
    "Assegna tavolo a cliente",
    manager_assegna_tavolo_a_cliente
  },
  {
    "Visualizza posti per i quali è possibile stampare lo scontrino",
    manager_visualizza_tavoli_poss_stampare_scontrino
  },
  {
    "Stampa scontrino",
    manager_stampa_scontrino_tavolo_occupato
  }
```

```
};

menu_entry entries_barman_e_pizzaiolo[ENTRIES_LEN_BARMAN_E_PIZZAIOLO] =
{
  {
    "Visualizza scelte ancora da prendere in carico",
    lavoratore_cucina_visualizza_scelte_da_preparare
  },
  {
    "Prendi in carico scelta da preparare",
    lavoratore_cucina_prendi_in_carico_scelta_da_preparare
  },
  {
    "Visualizza scelte prese in carico da espletare",
    lavoratore_cucina_visualizza_scelte_presa_in_carico_da_espletare
  },
  {
    "Visualizza informazioni (ing extra e quantita) scelte prese in carico",
    lavoretore_cucina_visualizza_info_scelte_prese_in_carico
  },
  {
    "Espleta scelta",
    lavoratore_cucina_espleta_scelta
  }
};

menu_entry entries_cameriere[ENTRIES_LEN_CAMERIERE] =
{
  {
    "Visualizza TUTTI i tavoli assegnati",
    cameriere_visualizza_tavoli_assegnati
  },
  {
    "Visualizza situazione tavoli assegnati",
```

```
        cameriere_visualizza_situazione_tavoli
    },
    {
        "Prendi ordinazione da un tavolo occupato",
        cameriere_prendi_ordinazione
    },
    {
        "Chiudi ordinazione da un tavolo occupato",
        cameriere_chiudi_ordinazione
    },
    {
        "Prendi una scelta per ordinazione aperta",
        cameriere_prendi_scelta_per_ordinazione
    },
    {
        "Aggiungi ingrediente extra alla scelta",
        cameriere_aggiungi_ing_extra_per_scelta
    },
    {
        "Visualizza scelte espletate",
        cameriere_visualizza_scelte_espletate
    },
    {
        "Effettua consegna di una scelta espletata",
        cameriere_effettua_consegna
    }
};

#endif
#ifndef MYBOOL_H
#define MYBOOL_H

typedef enum {
        FALSE,
```

```
        TRUE
} mybool;

#define mybool_to_str(x) (x == FALSE ? "false" : "true")

#endif
#ifndef MYEZCVT_H
#define MYEZCVT_H

#include "mybool.h"

mybool cvt_str_to_int(const char* in, int* out);
mybool cvt_str_to_double(const char* in, double* out);
mybool cvt_str_yesno_to_mybool(const char* in, mybool* out);

#endif
#ifndef MYSQL_UTILS_H
#define MYSQL_UTILS_H

#include <mysql.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "global_config.h"
#include "mybool.h"

#define MYSQL_BASIC_PRINTERROR(msg) \
        (fprintf(stderr, "(%s) Error %u: %s\n", \
        msg, \
        mysql_errno(cfg.db_conn), \
        mysql_error(cfg.db_conn)))

//always failure
#define MYSQL_BASIC_PRINTERROR_EXIT(msg) \
```

```
        { \
                MYSQL_BASIC_PRINTERROR(msg); \
                close_and_exit(EXIT_FAILURE); \
        }

#define MYSQL_STMT_BASIC_PRINTERROR(msg, stmt) \
        (fprintf(stderr, "(%s) Error %u (%s): %s\n", \
                        msg, \
                        mysql_stmt_errno (stmt), \
                        mysql_stmt_sqlstate(stmt), \
                        mysql_stmt_error (stmt)))

// always failure
#define MYSQL_STMT_BASIC_PRINTERROR_EXIT(msg, stmt) \
  {                                 \
   MYSQL_STMT_BASIC_PRINTERROR(msg, stmt);        \
   close_and_exit(EXIT_FAILURE);       \
  }

#define RESET_MYSQL_BIND(name) \
        memset(name, 0, sizeof(name))

#define INIT_MYSQL_BIND(name, n) \
        MYSQL_BIND name[n]; \
        RESET_MYSQL_BIND(name)

#define INIT_MYSQL_TIME_ONLYDATE(name, d, m, y) \
        MYSQL_TIME name; \
        memset(&name, 0, sizeof(name)); \
        name.day = d; \
        name.month = m; \
        name.year = y

#define INIT_MYSQL_TIME_DATETIME(name, d, m, y, h, min, s) \
```

```
        INIT_MYSQL_TIME_ONLYDATE(name, d, m, y); \
        name.hour = h; \
        name.minute = min; \
        name.second = s

#define __basic_set_inout_maybe_null_param_type(par, idx, ptr, buflen, buftype, isnullptr) \
        par[idx].buffer = ptr; \
        par[idx].buffer_type = buftype; \
        par[idx].buffer_length = buflen; \
        par[idx].is_null = isnullptr

#define __basic_set_inout_param_type(par, idx, ptr, buflen, buftype) \
        __basic_set_inout_maybe_null_param_type(par, idx, ptr, buflen, buftype, NULL)

#define __basic_set_inout_param_int_or_date(idx, buf, subtype, params, isnullptr) \
        __basic_set_inout_maybe_null_param_type(params, idx, buf, sizeof(*buf), subtype, isnullptr)

#define __basic_mysql_stmt_bind(fun, stmt, params) \
        if (fun(stmt, params)) { \
                MYSQL_STMT_BASIC_PRINTERROR_EXIT(#fun, stmt); \
        }

MYSQL_STMT *init_and_prepare_stmt(const char* query);
mybool execute_stmt(MYSQL_STMT* stmt);
void close_everything_stmt(MYSQL_STMT* stmt);

#define next_result_stmt(stmt) \
   if (mysql_stmt_next_result(stmt) > 0) { \
                MYSQL_STMT_BASIC_PRINTERROR_EXIT("mysql_stmt_next_result", stmt); \
        }

/* inout params */

#define set_in_param_string(idx, buf, params) \
```

```
        __basic_set_inout_param_type(params, idx, buf, strlen(buf),
MYSQL_TYPE_VAR_STRING)

#define set_out_param_string(idx, buf, params) \
        __basic_set_inout_param_type(params, idx, buf, sizeof(buf),
MYSQL_TYPE_VAR_STRING)

#define set_out_param_maybe_null_string(idx, buf, isnullptr, params) \
        __basic_set_inout_maybe_null_param_type(params, idx, buf, sizeof(buf), \

        MYSQL_TYPE_VAR_STRING, isnullptr)

#define set_inout_param_int(idx, buf, params) \
        __basic_set_inout_param_int_or_date(idx, buf, MYSQL_TYPE_LONG, params, NULL)

#define set_inout_param_tinyint(idx, buf, params) \
        __basic_set_inout_param_int_or_date(idx, buf, MYSQL_TYPE_TINY, params, NULL)

#define set_inout_param_smallint(idx, buf, params) \
        __basic_set_inout_param_int_or_date(idx, buf, MYSQL_TYPE_SHORT, params, NULL)

#define set_out_param_maybe_null_int(idx, buf, isnullptr, params) \
        __basic_set_inout_param_int_or_date(idx, buf, MYSQL_TYPE_LONG, params, isnullptr)

#define set_inout_param_double(idx, buf, params) \
        __basic_set_inout_param_type(params, idx, buf, 0, MYSQL_TYPE_DOUBLE)

#define set_inout_param_mybool(idx, buf, params) \
        set_inout_param_tinyint(idx, buf, params)

#define set_in_param_null(idx, params) \
        params[idx].is_null_value = TRUE

#define set_inout_param_date(idx, buf, params) \
```

```
        __basic_set_inout_param_int_or_date(idx, buf, MYSQL_TYPE_DATE, params, NULL)

#define set_inout_param_datetime(idx, buf, params) \
        __basic_set_inout_param_int_or_date(idx, buf, MYSQL_TYPE_DATETIME, params,
NULL)

/* bind */

#define bind_param_stmt(stmt, params) \
        __basic_mysql_stmt_bind(mysql_stmt_bind_param, stmt, params)

#define bind_result_stmt(stmt, params) \
        __basic_mysql_stmt_bind(mysql_stmt_bind_result, stmt, params)

/* close */

#define close_only_stmt(stmt) \
        if (mysql_stmt_close(stmt)) { \
                MYSQL_BASIC_PRINTERROR_EXIT("mysql_stmt_close"); \
        }

/* fetch */

#define begin_fetch_stmt(stmt) \
        { \
                int stmt_fetch_error; \
                for(int i = 0; !(stmt_fetch_error = mysql_stmt_fetch(stmt)); ++i) {

#define end_fetch_stmt() \
                } \
                if (stmt_fetch_error != MYSQL_NO_DATA) { \
                MYSQL_STMT_BASIC_PRINTERROR_EXIT("mysql_stmt_fetch", stmt); \
      } \
        }
```

```
#endif
#ifndef MYUTILS_H
#define MYUTILS_H

#include "mybool.h"
#include "role.h"

typedef mybool (*entry_handler_fpt)();

typedef struct {
	char* entry;
	entry_handler_fpt handler;
} menu_entry;

typedef enum {
	STRING,
	MYBOOL,
	INTEGER,
	DOUBLE
} form_output_cvt_type;

typedef struct {
	char* field_name;
	int expected_min_len;
	int expected_max_len;
	void* output;
	form_output_cvt_type output_type;
	int output_size;
} form_field;

#define clear() (printf("\e[1;1H\e[2J"))

mybool show_menu();
```

```
mybool show_form(const form_field* fields, int fields_len);
mybool date_check(int day, int month, int year);
role str_to_role(char* role);
const char* role_to_str(role r);

#endif
#ifndef OP_H
#define OP_H

#include "myutils.h"

#define checked_execute_stmt(stmt) \
        if (!execute_stmt(stmt)) { \
                puts("operazione annullata."); \
                close_only_stmt(stmt); \
                return FALSE; \
        }

#define checked_show_form(fields, fields_len)   \
        if (show_form(fields, fields_len) == FALSE) { \
                puts("operazione annullata.");          \
                return FALSE;                           \
        }

#define checked_date_check(day, month, year)   \
        if (date_check(day, month, year) == FALSE) { \
                puts("data inserita non valida!");      \
                puts("operazione annullata.");          \
                return FALSE;                           \
        }

#define check_affected_stmt_rows(status, stmt, msg, ...) \
        if(mysql_stmt_affected_rows(stmt) == 0) { \
                printf(msg, __VA_ARGS__); \
```

```
                status = FALSE; \
        }

#define __basic_form_field(fields, idx, field_nm, expected_min, expected_max, \
                output_sz, output_buf, output_ty)               \
        fields[idx].field_name = field_nm;                              \
        fields[idx].expected_min_len = expected_min;                    \
        fields[idx].expected_max_len = expected_max;                    \
        fields[idx].output = output_buf;                            \
        fields[idx].output_type = output_ty;                            \
        fields[idx].output_size = output_sz;

#define string_form_field(f, i, n, mi, ma, os, ob) \
        __basic_form_field(f, i, n, mi, ma, os, ob, STRING)

#define int_form_field(f, i, n, mi, ma, ob) \
        __basic_form_field(f, i, n, mi, ma, 0, ob, INTEGER)

#define double_form_field(f, i, n, mi, ma, ob) \
        __basic_form_field(f, i, n, mi, ma, 0, ob, DOUBLE)

#define mybool_form_field(f, i, n, ob) \
        __basic_form_field(f, i, n, 1, 3, 0, ob, MYBOOL)

mybool manager_crea_nuovo_utente();
mybool manager_ripristina_password_utente_esistente();
mybool manager_aggiungi_nuovo_tavolo();
mybool manager_aggiungi_nuovo_ingrediente();
mybool manager_aggiungi_prodotto_del_menu();
mybool manager_associa_prodotto_e_ingrediente();
mybool manager_crea_turno();
mybool manager_rimuovi_prodotto_del_menu();
mybool manager_rimuovi_ingrediente();
mybool manager_rimuovi_prodotto_e_ingrediente();
```

```
mybool manager_assegna_turno();
mybool manager_visualizza_turni();
mybool manager_visualizza_utenti();
mybool manager_visualizza_tavoli();
mybool manager_visualizza_turni_assegnati();
mybool manager_visualizza_turno_attuale();
mybool manager_visualizza_menu();
mybool manager_visualizza_situazione_ingredienti();
mybool manager_visualizza_assoc_prodotti_ingredienti();
mybool manager_inc_disp_ingrediente();
mybool manager_visualizza_entrate_giornaliere();
mybool manager_visualizza_entrate_mensili();
mybool manager_visualizza_scontrini_non_pagati();
mybool manager_contrassegna_scontrino_pagato();
mybool manager_assegna_tavolo_a_cliente();
mybool manager_visualizza_tavoli_poss_stampare_scontrino();
mybool manager_stampa_scontrino_tavolo_occupato();

mybool cameriere_visualizza_situazione_tavoli();
mybool cameriere_prendi_ordinazione();
mybool cameriere_chiudi_ordinazione();
mybool cameriere_prendi_scelta_per_ordinazione();
mybool cameriere_aggiungi_ing_extra_per_scelta();
mybool cameriere_visualizza_scelte_espletate();
mybool cameriere_effettua_consegna();
mybool cameriere_visualizza_tavoli_assegnati();

mybool lavoratore_cucina_visualizza_scelte_da_preparare();
mybool lavoratore_cucina_prendi_in_carico_scelta_da_preparare();
mybool lavoratore_cucina_visualizza_scelte_presa_in_carico_da_espletare();
mybool lavoretore_cucina_visualizza_info_scelte_prese_in_carico();
mybool lavoratore_cucina_espleta_scelta();

#endif
```

```
#ifndef PARSE_DBMS_CONN_CONFIG_H
#define PARSE_DBMS_CONN_CONFIG_H

#include "goodmalloc.h"
#include "mybool.h"

typedef struct {
	char* db_username;
	char* db_password;
	char* db_hostname;
	char* db_name;
	int db_port;
} dbms_conn_config;

mybool parse_dbms_conn_config(const char* path, dbms_conn_config* conf);
#define free_dbms_conn_config(dbmscfg) \
	good_free(dbmscfg.db_name); \
	good_free(dbmscfg.db_hostname); \
	good_free(dbmscfg.db_password); \
	good_free(dbmscfg.db_username);

#endif

#ifndef ROLE_H
#define ROLE_H

typedef enum {
	ROLE_UNKNOWN,  // login fallito
	ROLE_MANAGER,
	ROLE_CAMERIERE,
	ROLE_PIZZAIOLO,
	ROLE_BARMAN
} role;
```

```
#endif
```