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

Basi di Dati e Conoscenza

Progetto A.A. 2020/2021

Sistema di gestione di una pizzeria

0272911

Stefano Belli

**Indice**

1. Descrizione del Minimondo 2

2. Analisi dei Requisiti 3

3. Progettazione concettuale 7

4. Progettazione logica 15

5. Progettazione fisica 24

Appendice: Implementazione 82

# Descrizione del Minimondo

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27 | 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. |

# 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

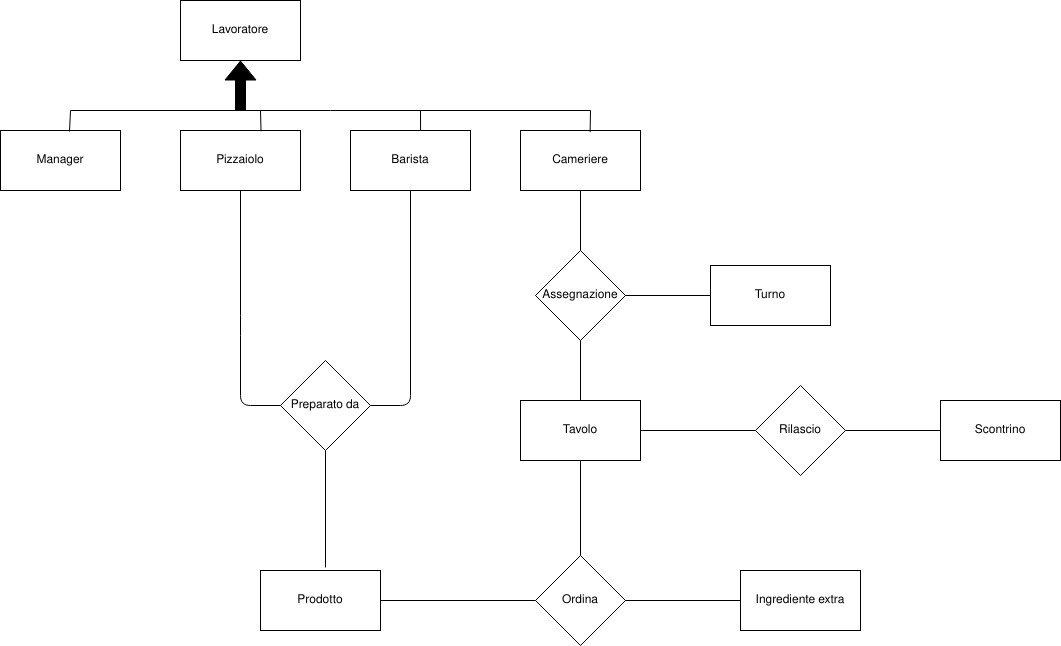
|  |
| --- |
| **Frasi di carattere generale** |
| In tale pizzeria è di interesse tenere traccia dei tavoli disponibili ed occupati, dei camerieri assegnati ai tavoli, dei pizzaioli che preparano le pizze, del barista, del manager.  Ciascuno dei lavoratori della pizzeria ha differenti mansioni e può 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. |

# Progettazione concettuale

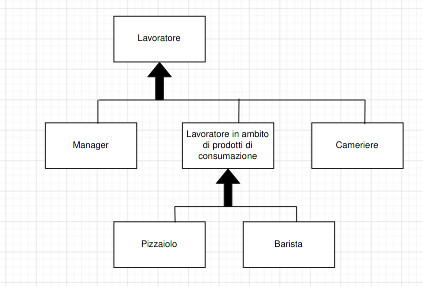
## Costruzione dello schema E-R

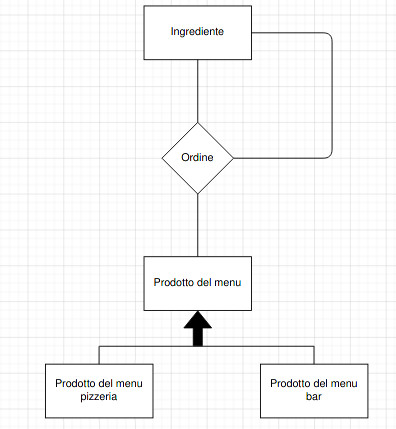
## Per la progettazione concettuale è stato seguito un approccio **top-down**:

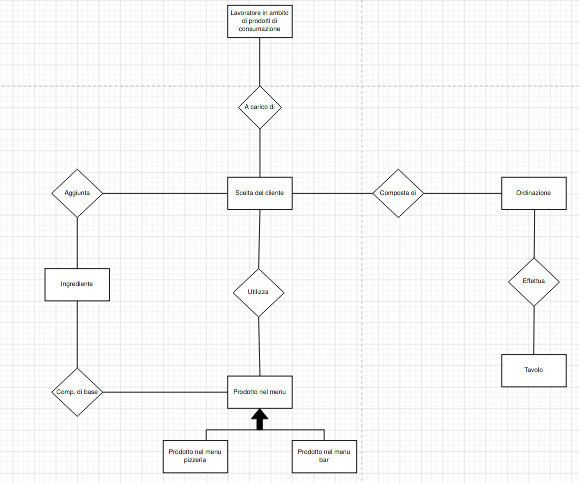
1. Identificazione dei concetti di base



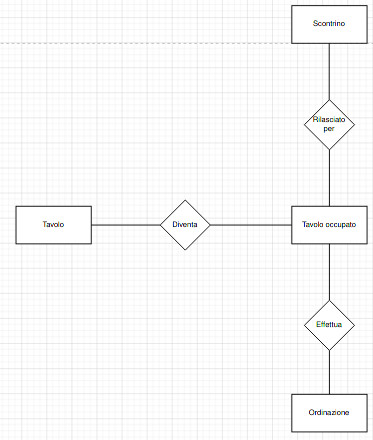
1. Raffinazione dei concetti

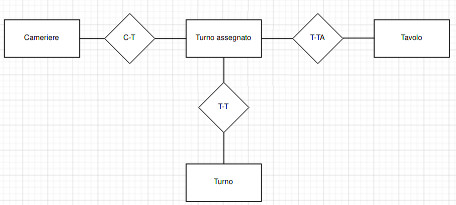






1. Applicazione di design pattern (reificazione relazione ternaria e instance-of)





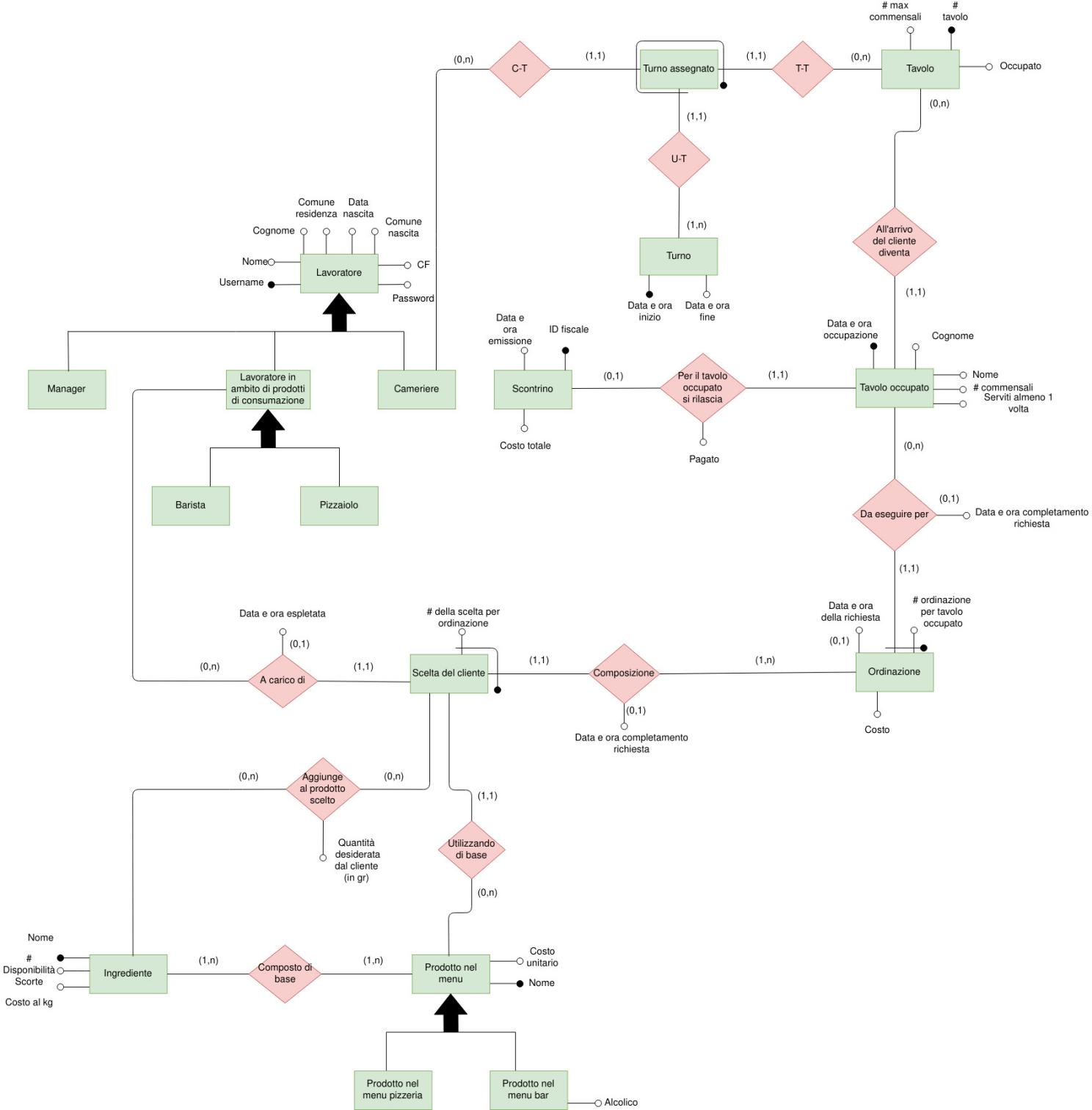
1. Merge dei pezzi risultanti

Si uniscono i pezzi dello schema

1. 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:
  + ,
* RD2: Il costo di una ordinazione si ricava dalla seguente formula:
* RD3: Il costo totale si ricava dalla seguente formula:

*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 |

# Progettazione logica

## Volume dei dati

|  |  |  |
| --- | --- | --- |
| **Concetto nello schema** | **Tipo[[1]](#footnote-0)** | **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 pagato | 20/giorno |
| 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  18 Tavolo  10 C-T  10 UtenteLavoratore | L  L  L  L  L  L  L | (3 + 3 + 3 + 3 + 18 + 10 + 10) \* 1 = 50 accessi / giorno |
| 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  1 TurnoAssegnato  1 UtenteLavoratore  2 Tavolo | L  L  L  L  L  L  L | (1 + 1 + 1 + 1 + 1 + 1 + 2) \* 10 = 80 accessi / giorno |
| 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)

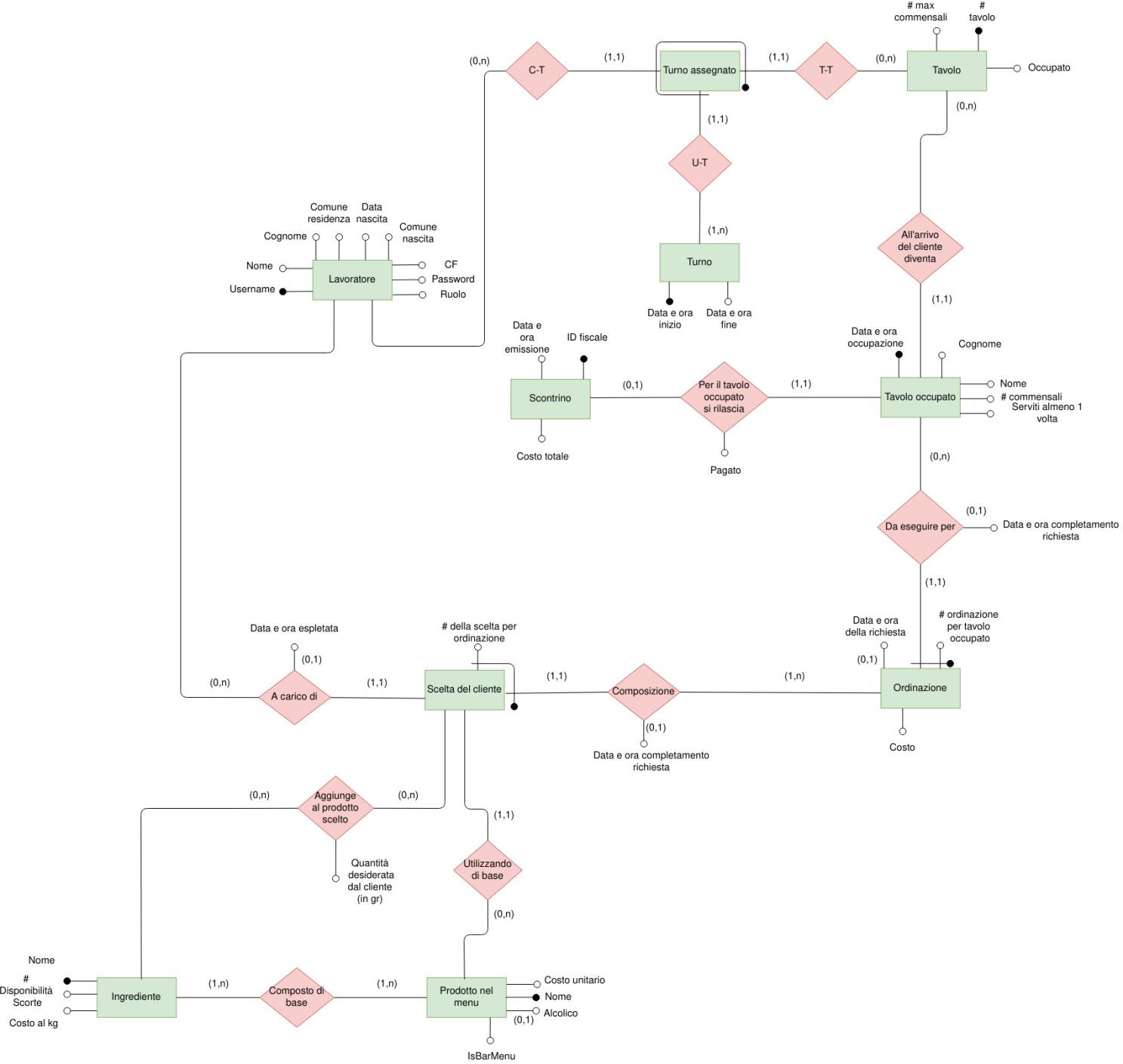
Data la complessità delle operazioni da effettuare per ricavare gli attributi (e dal volume dei dati), è risultato opportuno fornire il dato ridondante già “pronto”.

*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:*

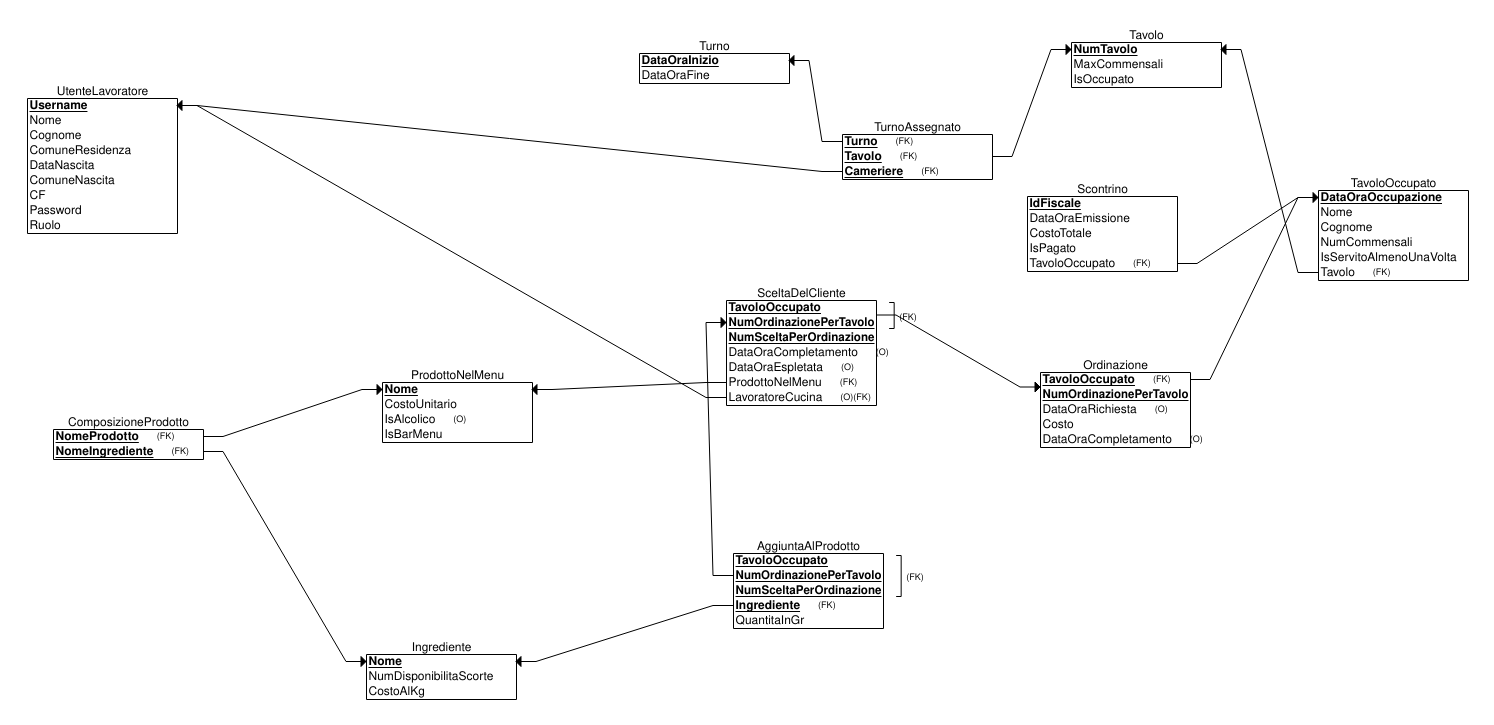
* Lavoratore: Username - è sufficiente per identificare ogni singolo lavoratore col proprio username
* Turno Assegnato: identificazione esterna (Username, Data e ora inizio, # tavolo) - attributi minimi necessari all’identificazione di un particolare turno assegnato a un particolare lavoratore per un 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

# 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]](#footnote-1)** |
| 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]](#footnote-2):** |
| 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 |

|  |  |
| --- | --- |
| **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 cameriereUsername varchar(10))

begin

declare exit handler for sqlexception

begin

rollback;

resignal;

end;

start transaction;

insert into

TurnoAssegnato(Tavolo, Turno, Cameriere)

values

(numTavolo, dataOraInizioTurno, cameriereUsername);

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

1. Indicare con E le entità, con R le relazioni [↑](#footnote-ref-0)
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. [↑](#footnote-ref-1)
3. IDX = index, UQ = unique, FT = full text, PR = primary. [↑](#footnote-ref-2)