Basi di Dati

Progetto A.A. 2023/2024

BASI DI DATI:

PIZZERIA

0280050

Giada Pica

Indice

1. Descrizione del Minimondo

Si vuole progettare il backend di un sistema informativo per la gestione dell'operatività di una pizzeria. In tale pizzeria è di interesse tenere traccia dei tavoli disponibili ed assegnati, dei camerieri associati ai tavoli, dei pizzaioli che preparano le pizze, del barista, del manager. Ciascuno dei lavoratori della pizzeria ha differenti mansioni e può effettuare operazioni differenti all'interno del sistema.
All'ingresso di un cliente, il manager lo riceve e lo registra, segnando nome, cognome e numero di commensali, assegnando un tavolo disponibile in grado di ospitarli tutti.
Un cameriere ha sempre la possibilità di visualizzare quali tavoli a lui assegnati sono occupati e quali sono stati serviti. Al momento di prendere l'ordine, il cameriere registra la comanda. Parte delle ordinazioni sono espletate dal barista, parte dal pizzaiolo.
Barista, pizzaiolo hanno sempre la possibilità di visualizzare cosa debbono preparare, in ordine di ricezione della comanda. Quando hanno preparato una bevanda o una pizza e la indicano come pronta, il cameriere può visualizzare cosa è pronto (in relazione agli ordini) e sapere cosa deve consegnare a quale tavolo.
Il menu è unico e definito dal manager, con i rispettivi prezzi.
Il manager ha la possibilità di stampare lo scontrino di un ordine. Inoltre, per motivi statistici, ha la possibilità di visualizzare le entrate giornaliere e/o mensili.

2. Analisi dei Requisiti

Identificazione dei termini ambigui e correzioni possibili

Linea	Termine	Nuovo termine	Motivo correzione
11	Parte	Ordini delle bevande	Gli ordini delle bevande sono espletati dal barista
12	Parte	Ordini delle pizze	Gli ordini delle pizze sono espletati dal pizzaiolo
15	Cosa	Comanda	Il cameriere può visualizzare quando una comanda è pronta da consegnare
18	Ordine	Comanda	Il manager ha la possibilità di stampare lo scontrino di una comanda

Specifica disambiguata

Si vuole progettare il backend di un sistema informativo per la gestione dell'operatività di una pizzeria. In tale pizzeria è di interesse tenere traccia dei tavoli disponibili ed assegnati, dei camerieri associati ai tavoli, dei pizzaioli che preparano le pizze, del barista, del manager. Ciascuno dei lavoratori della pizzeria ha differenti mansioni e può effettuare operazioni differenti all'interno del sistema.
All'ingresso di un cliente, il manager lo riceve e lo registra, segnando nome, cognome e numero di commensali, assegnando un tavolo disponibile in grado di ospitarli tutti.
Un cameriere ha sempre la possibilità di visualizzare quali tavoli a lui assegnati sono occupati e quali sono stati serviti. Al momento di prendere l'ordine, il cameriere registra la comanda. Gli ordini delle bevande sono espletati dal barista, gli ordini delle pizze 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 e la indicano come pronta, il cameriere può visualizzare le comande pronte (in relazione agli ordini) e sapere cosa deve consegnare a quale tavolo.
Il menu è unico e definito dal manager, con i rispettivi prezzi.
Il manager ha la possibilità di stampare lo scontrino di una comanda. Inoltre, per motivi statistici, ha la possibilità di visualizzare le entrate giornaliere e/o mensili.

Glossario dei Termini

Termine	Descrizione	Sinonimi	Collegamenti
Manager	Lavoratore della pizzeria che coordina le occupazioni dei tavoli e dell'accoglienza clienti		Cameriere, Cliente, Tavolo
Cameriere	Lavoratore della pizzeria che si occupa di servire i tavoli		Tavolo
Cliente	Persona registrata a cui è assegnato un tavolo		Tavolo
Tavolo	Tavolo della pizzeria che può essere occupato o meno che espleta la comanda		Cameriere, Comanda, Cliente
Comanda	Lista di pizze e/o bevande ordinate da un tavolo	Ordine	Tavolo, Scontrino, Menù
Scontrino	Resoconto finale della comanda		Comanda
Pizza	Prodotto del menù preparato dal pizzaiolo		Menù, Comanda
Bevanda	Prodotto del menù preparato dal barista		Menù, Comanda
Menù	L'insieme dei prodotti venduti dalla pizzeria		Pizze, Bevande

Raggruppamento dei requisiti in insiemi omogenei

Frasi relative a Manager
All'ingresso di un cliente, il manager lo riceve e lo registra.
Il menu è unico e definito dal manager.
Il manager ha la possibilità di stampare lo scontrino di una comanda. Inoltre, per motivi statistici, ha la possibilità di visualizzare le entrate giornaliere e/o mensili

Frasi relative a Cameriere
In tale pizzeria è di interesse tenere traccia dei tavoli disponibili ed assegnati, dei camerieri associati ai tavoli.
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.
il cameriere può visualizzare le comande pronte (in relazione agli ordini) e sapere cosa deve consegnare a quale tavolo.

Frasi relative a Cliente
All’ingresso di un cliente, il manager lo riceve e lo registra, segnando nome, cognome e numero di commensali.

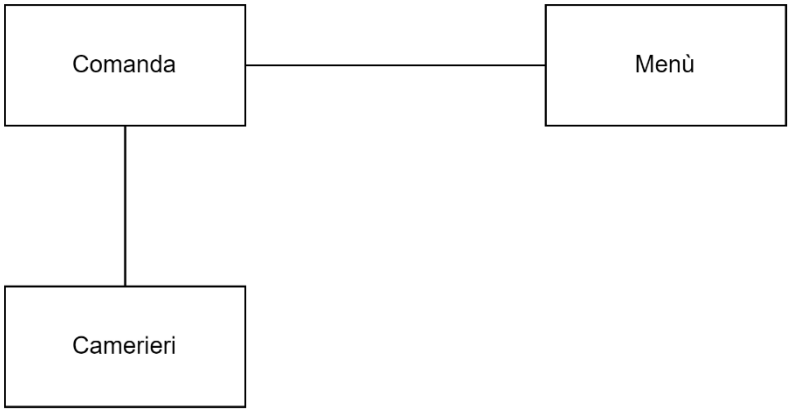
Frasi relative a Tavolo
All’ingresso di un cliente, il manager lo riceve e lo registra, segnando nome, cognome e numero di commensali, assegnando un tavolo disponibile in grado di ospitarli tutti.
Un cameriere ha sempre la possibilità di visualizzare quali tavoli a lui assegnati sono occupati e quali sono stati serviti.

Frasi relative a Comanda
Al momento di prendere l’ordine, il cameriere registra la comanda.
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 e la indicano come pronta, il cameriere può visualizzare le comande pronte.
Il manager ha la possibilità di stampare lo scontrino di una comanda.

3. Progettazione concettuale

Costruzione dello schema E-R

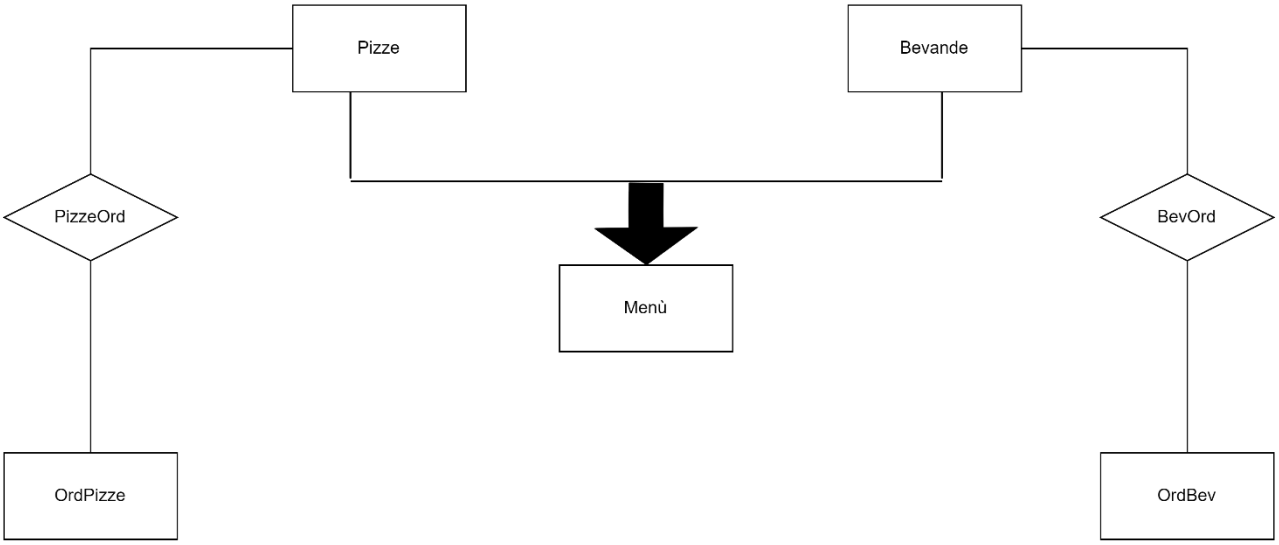
Per la creazione del modello concettuale ho utilizzato una strategia mista definendomi dei macro concetti da cui partire stabilendo una visione chiara delle basi del sistema e sviluppare perciò gradualmente i dettagli man mano che si sale nel livello di complessità. L'obiettivo è creare un modello ER ben strutturato e coerente che rispecchi le dinamiche e i requisiti della pizzeria.



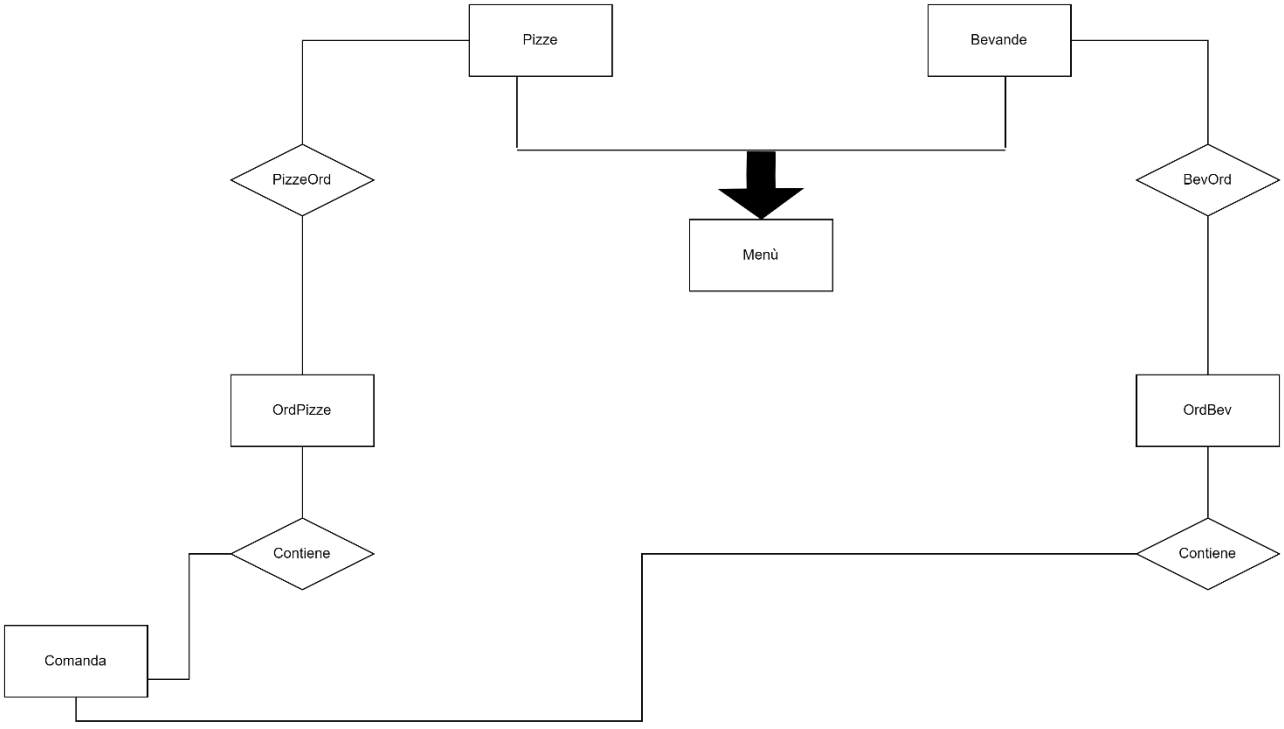
I tre macro concetti sono: il menù, ovvero l'insieme dei prodotti offerti dalla pizzeria e decisi dal manager, la comanda ovvero l'ordine effettuato da un tavolo relativo ai prodotti del menù e infine i camerieri che solo coloro che gestiscono il tavolo a cui è relativa la comanda e consegnano le portate del menù.

Per ogni macro concetto ho voluto analizzarlo in maniera singola affiancando la specifica, in modo da avere una visione globale di ogni entità.

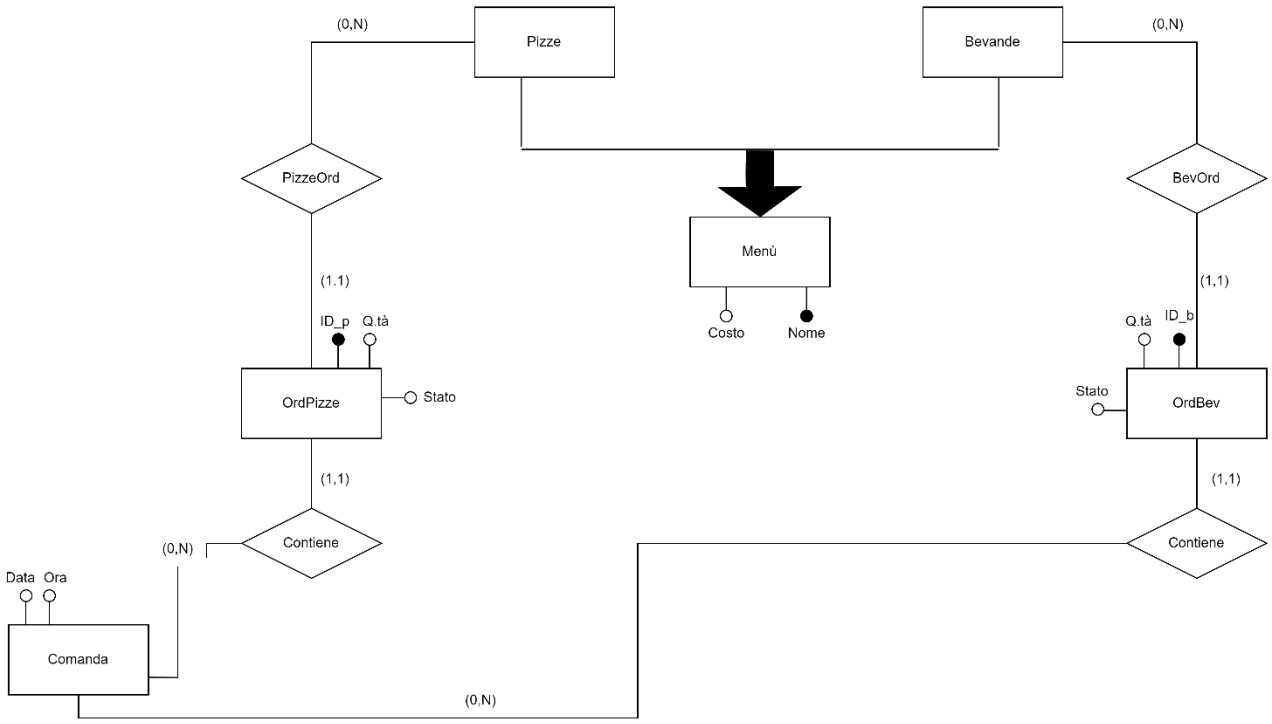
Il primo concetto analizzato è il menù che si suddivide in pizze e bevande, seguito poi dalla necessità di suddividere gli ordini di una comanda in ordini delle pizze e ordini delle bevande in modo che ognuno dei due possa essere separatamente preparato da pizzaiolo o barista.



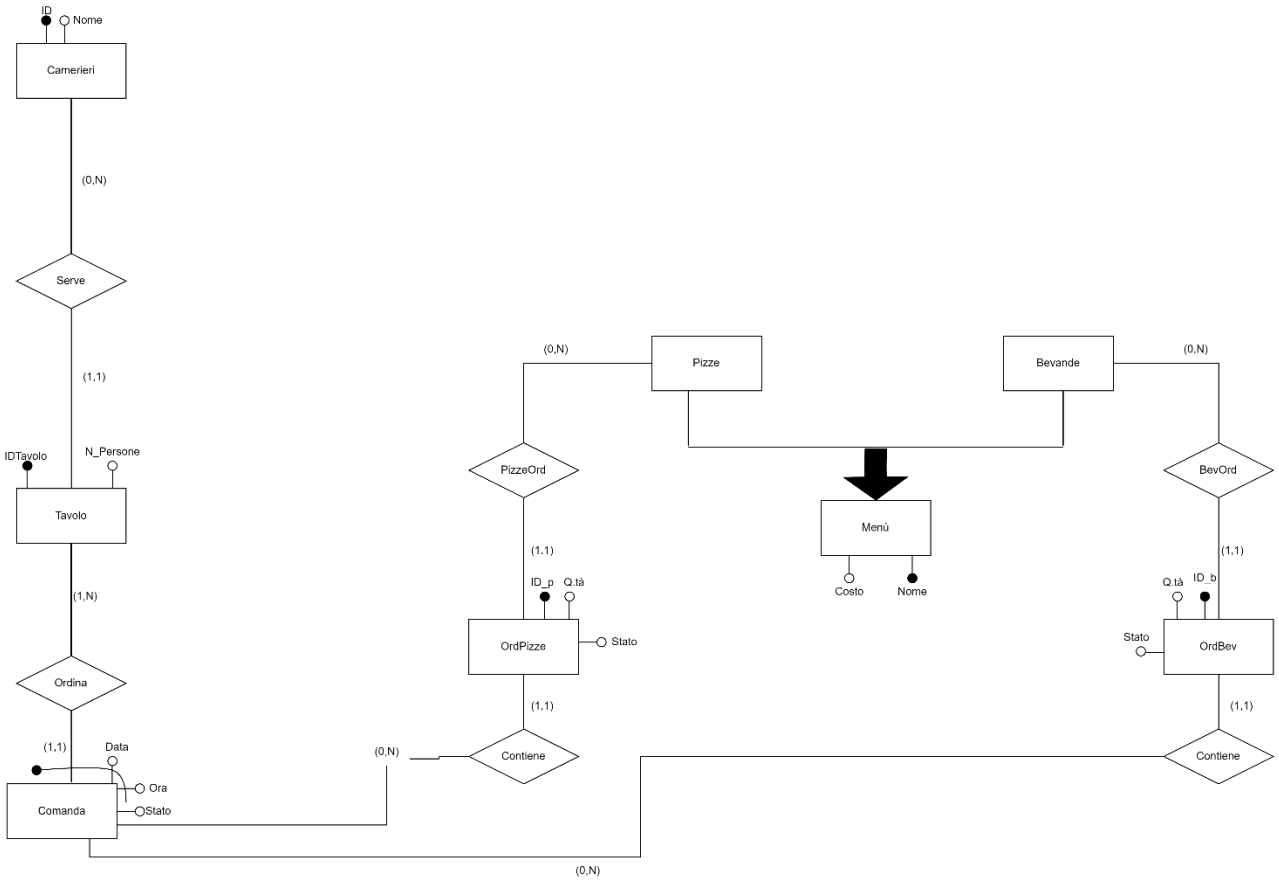
Ovviamente le due entità relative agli ordini delle pizze e delle bevande sono strettamente collegate alla comanda, sono andata perciò a collegare la comanda con i singoli ordini del tavolo.



Con un’ulteriore raffinazione ho aggiunto le cardinalità e gli identificatori di alcune delle entità:



Prima di passare allo schema ER completo va analizzato l'ultimo macro concetto, i camerieri che gestiscono le comande di un tavolo, li identifico con un codice univoco e inserisco anche il nome.
I camerieri sono gli unici lavoratori ad essere esplicitamente indicati in questa fase poiché la loro importanza è rilevante dato il collegamento che hanno con i tavoli e la relativa segnalazione di comanda pronta da consegnare.
Vado inoltre a rappresentare come chiave dell'entità Comanda la data e l'ora, che non sono univoche, insieme al tavolo che l'ha effettuata.



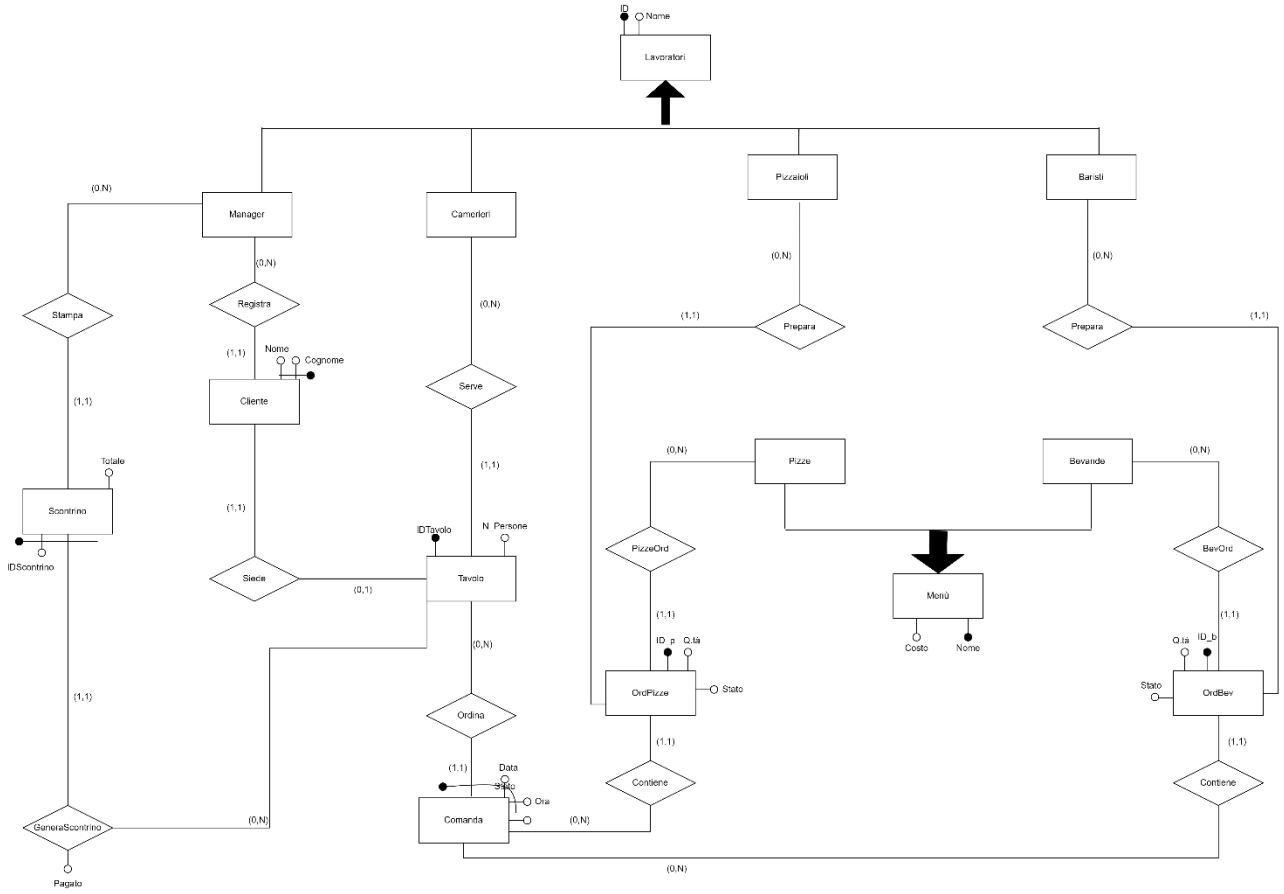
I tre macro-concetti sono adesso collegati fra di loro in maniera coerente all'interpretazione della specifica.

Integrazione finale

Avendo effettuato un'integrazione passo passo in questa sezione non c'è un'associazione dei macro concetti ma una raffinazione ulteriore delle necessità per la specifica.

Infatti una volta finito il collegamento fra i macro-concetti ho inserito le ultime due entità di cui avevo necessità per la richiesta, ovvero i clienti e lo scontrino.

Inoltre a fini concettuali vado ad aggiungere tutti i lavoratori presenti all'interno del minimondo in modo da dare un quadro generale di ciò che i singoli possono fare.



Regole aziendali

RA1: Un tavolo può essere assegnato a un cliente se risulta libero, e se soddisfa N_pers <= N_Persone, ovvero le persone che richiede di far sedere il cliente sia possibile accoglierle tutte nello stesso tavolo.

RA2: Un tavolo torna nello stato libero se lo scontrino è stato rilasciato e risulta pagato.

RA3: Ogni cameriere vede solo i tavoli a lui assegnati.

RA4: Le pizze ordinate e le bevande sono visibili solo a pizzaiolo e barista rispettivamente.

RA5: Una comanda contiene sia pizze che bevande (l'una o l'altra può essere null) e quando verrà evasa prima di essere consegnata devono essere pronte tutte le pizze e bevande contenute, l'ordinazione può essere effettuata per una sola tipologia di pizze/bevande alla volta ma con quantità >0.

Dizionario dei dati

Entità	Descrizione	Attributi	Identificatori
Scontrino	L'entità rappresenta un documento che contiene il totale speso per una specifica	IDScontrino, Totale, isPagato	IDScontrino, IDTavolo

	comanda emessa in una data e ora specifiche.		
Comanda	Descrizione degli ordini effettuati da un Tavolo	Data_Ora, Tavolo_, Stato	Data_Ora, Tavolo_
Tavolo	L'entità rappresenta ogni singolo tavolo presente nella pizzeria.	IDTavolo, N_posti, Occupazione, Cameriere	IDTavolo
Camerieri	Lavoratore della pizzeria che gestisce le comande dei tavoli.	ID, Nome	ID
Pizza	Prodotto del menù.	Nome, Prezzo	Nome
Bevanda	Prodotto del menù.	Nome_bevanda, costo	Nome_bevanda
Cliente	Persona che entra nella pizzeria e si siede a un Tavolo	Nome, Cognome, N_pers	Nome, Cognome
OrdPizze	Pizze effettivamente ordinate.	Q_tà, Pizza_pronta, comanda_Data_ora, comanda_Tavolo_, Pizze_Nome	Q_tà, comanda_Tavolo_, Pizze_Nome
OrdBev	Bevande effettivamente ordinate.	Quantità, Bevanda_pronta, comanda_Data_ora, comanda_Tavolo_, Bevande_Nome_bevanda	Quantità, comanda_Tavolo_, Bevande_Nome_bevanda

4. Progettazione logica

Volume dei dati

Supponendo un totale di 60-80 clienti al giorno, contando una media di 4 persone per tavolo ogni comanda conterrà all'incirca 4 pizze e 4 bevande. Ogni tavolo fa di media 2 comande e per ogni tavolo viene emesso un solo scontrino.

Ogni cliente è strettamente legato al tavolo e la registrazione è perciò valida solo per la giornata, così come la comanda e gli ordini dei singoli prodotti.

Gli scontrini verranno memorizzati per un anno.

Il menù e i camerieri non hanno una data di eliminazione poiché si presume non varino.

Concetto nello schema	Tipo[1]	Volume atteso
Pizze	E	15-20
Bevande	E	15-20
Menù	E	30-40
Clienti	E	60-80
Tavoli	E	100-120
Camerieri	E	5-6
Scontrini	E	21000-28000
Comande	E	60-80
OrdPizze	E	960-1280
OrdBev	E	960-1280
Siede	R	60-80
GeneraScontrino	R	100-120
Ordina	R	100-120
ContieneP	R	960-1280
ContieneB	R	960-1280
Serve	R	100-120
PizzaOrd	R	240-320
BevOrd	R	240-320

Tavola delle operazioni

Cod.	Descrizione	Frequenza attesa
1	Aggiunta prodotto nel menù	10/mese
2	Rimozione di prodotti dal menù	12/anno
3	Aggiunta di un tavolo	10/anno
4	Contrassegna scontrino come pagato	100/giorno
5	Assegnazione tavolo a un cliente	60/giorno

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

6	Visualizza tavoli disponibili	50/giorno
7	Visualizza tavoli che devono pagare	50/giorno
8	Visualizza statistiche giornaliere	1/giorno
9	Visualizza statistiche mensili	1/mese
10	Aggiunta nuovo lavoratore	4/anno
11	Assegnazione tavolo a cameriere (fine manager)	5/giorno
12	Prendi comanda (camerieri)	30/giorno
13	Visualizza tavoli assegnati	5/giorno
14	Visualizza comanda pronta (fine camerieri)	30/giorno
15	Visualizza prodotti da preparare (pizzaioli e baristi)	300/giorno
16	Segna prodotti come pronti (fine pizzaioli e baristi)	300/giorno
17	Login	10/giorno

Costo delle operazioni

Op1: Accesso in scrittura -> 2*10 =20 accessi/mese

Op2: Accesso in scrittura -> 2*12 =24 accessi/anno

Op3: Accesso in scrittura -> 2*10 =20 accessi/ anno

Op4: Accesso in scrittura -> 2*100 =200accessi/giorno

Op5: Accesso in lettura su Tavolo, scrittura su Cliente, scrittura su Tavolo

-> (100+2+2)*60 =6240accessi/giorno

Op6: Accesso in lettura -> 1*50= 50 accessi/giorno

Op7: Accesso in lettura -> 1*50= 50 accessi/giorno

Op8: Accesso in lettura -> 1*100*1=100 accessi/giorno

Op9: Accesso in lettura -> 1*20*100*1=2000 accesi/mese

Op10: Accesso in scrittura -> 2*4= 8 accessi/anno

Op11: Accesso in lettura su Cameriere, accesso in scrittura su Cameriere

-> (2+2)*5= 20 accessi/giorno

Op12: Accesso in lettura su Tavolo, accesso in scrittura su Comanda, accesso in scrittura su OrdPizze, accesso in scrittura su OrdBev

-> (1+2+2+2)*30 = 210 accessi/giorno

Op13: Accesso in lettura su Camerieri, accesso in lettura su Tavolo

-> (1+1)*5= 10 accessi/giorno

Op14: Accesso in lettura su Comanda, accesso in scrittura su Comanda, accesso in lettura su Tavolo

-> (1+2+1)*30=90 accessi/giorno

Op15: Accesso in lettura OrdPizze (oppure OrdBev), accesso in lettura Pizze (oppure Bevande)

-> (1+1)*300*2= 1200 accessi/giorno

Op16: Accesso in scrittura su OrdPizze (oppure OrdBev), accesso in scrittura su comanda

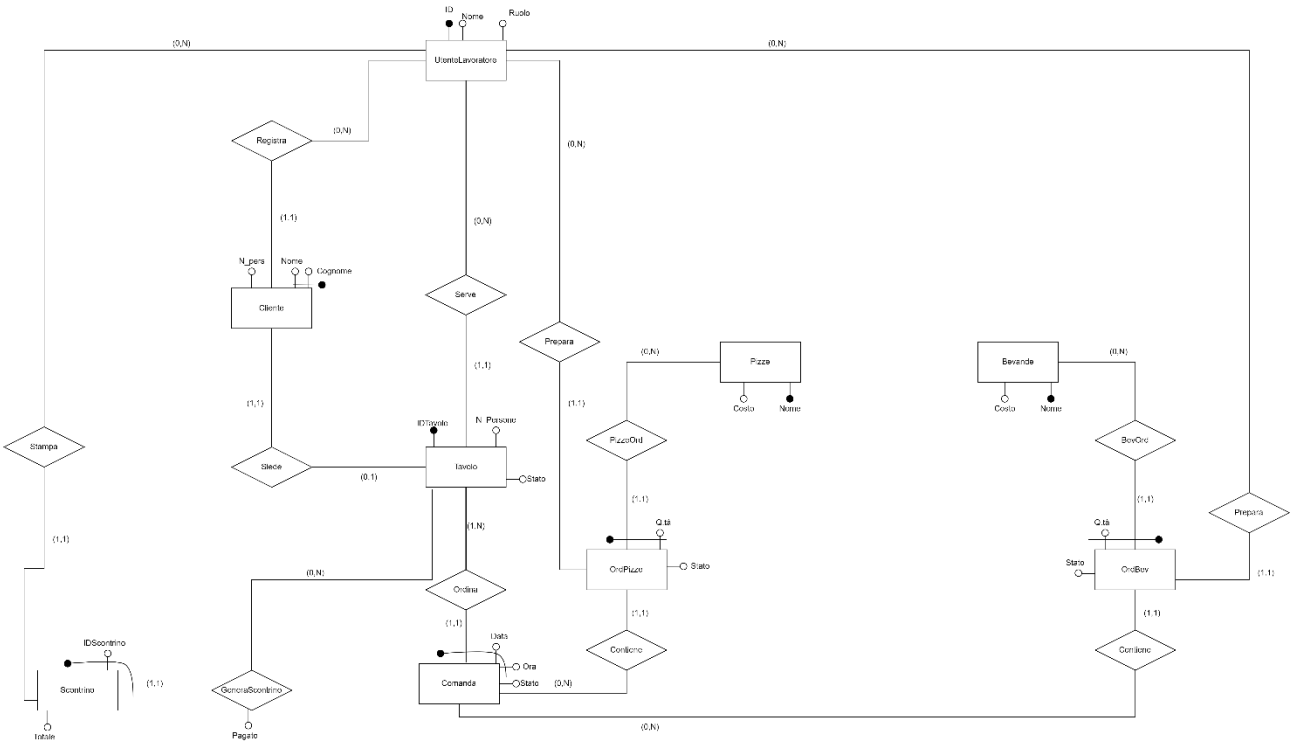
-> (2*2+2)*300 =1800 accessi/giorno

Op17: Accesso in lettura -> 1*10= 10 accessi/giorno

Ristrutturazione dello schema E-R

Nello schema concettuale è presente la ridondanza sulle entità di OrdPizze e OrdBev che sono del tutto simili ma che riferiscono a elementi del menù diversi, questa ridondanza non è utile eliminarla dato che è ottimale per le prestazioni dell'applicazione, inoltre la sua eliminazione andrebbe a creare disagio poiché in questa fase di ristrutturazione vado ad eliminare le generalizzazioni e quella del menù la elimino con l'accorpamento dell'entità padre nelle entità figlie dato che ciò mi risparmia un numero di accessi in memoria e conviene poiché quest'ultimi sono fatti da operazioni e soggetti distinti.

Inoltre vado ad eliminare la generalizzazione dei Lavoratori accorpando tutte le entità figlie in un unico UtenteLavoratore a cui aggiungo un attributo ruolo.



Trasformazione di attributi e identificatori

Nelle entità Tavolo, Comanda, OrdPizze e OrdBev è presente un attributo chiamato Stato, lo vado a modificare rinominandolo rispettivamente: occupazione, completata, pizza_pronta, bevanda_pronta.
Tra le entità tavolo e cliente i nomi sono già diversi ma vado a specificare che in clienti è N_commensali e in tavolo N_posti.
Inoltre l'attributo Nome è presente sia per il cameriere che per il cliente, le pizze e le bevande, diventa rispettivamente NomeCam, NomeClient, NomePizza, NomeBevanda.
Infine l'attributo costo su Pizze e Bevande, in Pizze diventa prezzo.

Traduzione di entità e associazioni

- **Camerieri** (ID,Nome)
- **Tavolo** (idTavolo, N_posti, Occupazione, Cameriere)

Cameriere vincolo di integrità referenziale su Cameriere

- **Cliente** (Nome, Cognome, N_persone, Tavolo)

Tavolo vincolo di integrità su Tavolo

- **Comanda** (Data_ora, Tavolo_, Stato)

  Tavolo vincolo di integrità referenziale su comanda

- **Scontrino** (idScontrino, Totale, Tavolo, isPagato, DataEmissione)

  Tavolo vincolo di integrità referenziale su Tavolo

- **OrdPizze** (Q_tà, Pizza_pronta, comanda_Data_ora, comanda_Tavolo_, Pizza_Nome)

Comanda_Data_ora, comanda_Tavolo_ vincolo di integrità referenziale su Comanda

- **OrdBev**(Quantità,Bevanda_pronta,comanda_data_ora,comanda_Tavolo_,Bevande_Nome_bevanda)

Comanda_Data_ora, comanda_Tavolo_ vincolo di integrità referenziale su Comanda

- **Pizze** (Nome, Prezzo)
- **Bevande** (Nome_bevanda, Costo)

Normalizzazione del modello relazionale

Non c'è stato bisogno di normalizzare il modello relazionale poiché è già in 3NF.

5. Progettazione fisica

Utenti e privilegi

Nell'applicazione sono previsti 4 tipi di utenti: Manager, Cameriere, Pizzaiolo e Barista che hanno ognuno username e password tramite il quale si identificano.

Strutture di memorizzazione

Tabella <Cameriere>		
Colonna	**Tipo di dato**	**Attributi[2]**
ID	INT	PK, NN, AI
Nome	Varchar(45)	NN

Tabella <Tavolo>		
Colonna	**Tipo di dato**	**Attributi[3]**
idTavolo	INT	PK, NN, AI
N_posti	INT	NN
Occupazione	Char(1)	
Cameriere	INT	

Tabella <Cliente>		
Colonna	**Tipo di dato**	**Attributi[4]**
Nome	Varchar(45)	PK, NN
Cognome	Varchar(45)	PK, NN
N_Commensali	INT	
Tavolo	INT	NN,AI

[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.

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

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

Tabella <Comanda>		
Colonna	**Tipo di dato**	**Attributi[5]**
Data_Ora	Datetime(1)	PK, NN
Tavolo	INT	NN, AI
Stato	Char(1)	

Tabella <Bevande>		
Colonna	**Tipo di dato**	**Attributi[6]**
Nome_bevanda	Varchar(45)	PK, NN
Costo	Float	

Tabella <Pizza>		
Colonna	**Tipo di dato**	**Attributi[7]**
Nome	Varchar(45)	PK, NN
Prezzo	Float	

Tabella <OrdBev>		
Colonna	**Tipo di dato**	**Attributi[8]**
Quantità	INT	PK, NN
Bevanda_pronta	Char(1)	
Comanda_Data_ora	Datetime(1)	NN
Comanda_Tavolo_	INT	NN
Bevanda_Nome_bevanda	Varchar(45)	PK, NN

Tabella <OrdPizza>		
Colonna	**Tipo di dato**	**Attributi[9]**
Q_tà	INT	PK, NN
Pizza_pronta	Char(1)	
Comanda_Data_ora	Datetime(1)	NN
Comanda_Tavolo_	INT	NN
Pizze_Nome_	Varchar(45)	PK, NN

Tabella <Scontrino>		
Colonna	**Tipo di dato**	**Attributi[10]**

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

idScontrino	INT	PK, NN, AI
Totale	Float	INT
Tavolo	INT	PK,NN
isPagato	Char(1)	
DataEmissione	Date	

Indici

Tabella <Tavolo >	
Indice <Primary>	**Tipo[11]:**
idTavolo	<PR>
Indice <Cameriere_idx>	**Tipo[12]:**
Cameriere	<IDX>

Tabella <Scontrino >	
Indice <Primary>	**Tipo[13]:**
idScontrino	<PR>
Indice<fk_Scontrino_comanda1_idx>	
Tavolo	<IDX>

Tabella <Pizze >	
Indice <Primary>	**Tipo[14]:**
Nome	<Pr>

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

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

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

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

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

Tabella <OrdPizze >	
Indice <Primary>	**Tipo[15]:**
Q_tà, comanda_Tavolo_, Pizze_Nome	<PR>
Indice <fk_OrdPizze_comanda_idx >	
Comanda_Data_ora, comanda_Tavolo_	<IDX>
Indice <fk_OrdPizze_Pizze1_idx >	
Pizze_Nome	<IDX>

Spiegazione indici valida per OrdPizze e OrdBev:

In queste tabelle che si riferiscono ai singoli ordini è stata necessaria l'indicizzazione poiché risulta molto utile (viste le query che coinvolgono la ricerca in base al tavolo e alla data_ora del tavolo) per la ricerca e l'ordinamento che verranno eseguiti in maniera più veloce e ottimizzata.

L'indicizzazione in base al nome è utile a migliorare l'efficienza dell'unione fra le tabelle oltre a permettere una ricerca delle pizze e bevande in maniera più rapida in base al nome.

Tabella <OrdBev >	
Indice <Primary>	**Tipo[16]:**
Q_tà, comanda_Tavolo_, Bevande_Nome_bevanda	<PR>
Indice <fk_OrdBev_comanda_idx >	
Comanda_Data_ora, comanda_Tavolo_	<IDX>
Indice <fk_OrdBev_Pizze1_idx >	
Bevande_Nome_bevanda	<IDX>

Tabella <Comanda >	
Indice <Primary>	**Tipo[17]:**
Data_ora, Tavolo_	<PR>
Indice <Tavolo_idx>	
Tavolo_	<IDX>

Tabella <Cliente >	
Indice <Primary>	**Tipo[18]:**
Nome, Cognome	<PR>
Indice <Tavolo_idx>	
Tavolo	<IDX>

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

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

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

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

Tabella <Cameriere >	
Indice <Primary>	**Tipo[19]:**
ID	<PR>

Tabella <Bevande >	
Indice <Primary>	**Tipo[20]:**
Nome_bevanda	<PR>

Trigger

- Trigger utilizzato per permettere l'inserimento di un nuovo scontrino solo se le comande a cui è associato il tavolo

```
CREATE DEFINER=`BasiGiada`@`%` TRIGGER `scontrino_BEFORE_INSERT` BEFORE INSERT ON `scontrino` FOR EACH ROW BEGIN
  DECLARE num_comande INT;

  -- Conta il numero di comande con stato diverso da 1 associate al tavolo dello scontrino
  SELECT COUNT(*)
  INTO num_comande
  FROM Comanda
  WHERE Tavolo_ = NEW.Tavolo AND Stato <> 1;

  -- Se ci sono comande con stato diverso da 1, genera un errore
  IF num_comande > 0 THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'Ci sono comande che non sono state espletate associate al tavolo dello scontrino.';
  END IF;
END
```

- Trigger che a seguito dell'aggiornamento di uno scontrino come pagato aggiorna il tavolo a cui si riferisce come libero

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

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

```
CREATE DEFINER=`BasiGiada`@`%` TRIGGER `scontrino_AFTER_UPDATE` AFTER UPDATE ON `scontrino` FOR EACH ROW BEGIN
   DECLARE tavolo_id INT;
   SELECT Tavolo INTO tavolo_id FROM scontrino WHERE idScontrino = NEW.idScontrino;
   IF NEW.isPagato = 1 THEN
      UPDATE Tavolo SET Occupazione = 0 WHERE idTavolo = tavolo_id;
   END IF;
END
```

- Trigger che dopo l'aggiornamento di una comanda controlla che le pizze e le bevande ad essa collegate siano pronte (l'aggiornamento della comanda è inteso per il suo stato di completata)

```
CREATE DEFINER=`BasiGiada`@`%` TRIGGER `comanda_BEFORE_UPDATE` BEFORE UPDATE ON `comanda` FOR EACH ROW BEGIN
 DECLARE pizza_pronta INT;
  DECLARE bevanda_pronta INT;

  SELECT Pizza_pronta INTO pizza_pronta
  FROM OrdPizze
  WHERE comanda_Tavolo_ = NEW.Tavolo_ AND comanda_Data_ora = NEW.Data_ora;

  SELECT Bevanda_pronta INTO bevanda_pronta
  FROM OrdBev
  WHERE comanda_Tavolo_ = NEW.Tavolo_ AND comanda_Data_ora = NEW.Data_ora;

  IF pizza_pronta <> 1 OR bevanda_pronta <>1 THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'Pizza_pronta e/o Bevanda_pronta non sono impostate a 1';
  END IF;
END
```

- Trigger che controlla che ogni cliente prima di essere registrato stia richiedendo un tavolo per un numero di persone inferiori a quelle che il tavolo può ospitare

```
CREATE
```

```
DEFINER=`BasiGiada`@`%`
TRIGGER `basididati`.`cliente_BEFORE_INSERT`
BEFORE INSERT ON `basididati`.`cliente`
FOR EACH ROW
BEGIN
  DECLARE num_persone_cliente INT;
  DECLARE num_posti_tavolo INT;

  SELECT N_Persone INTO num_persone_cliente
  FROM cliente
  WHERE Nome = NEW.Nome AND Cognome = NEW.Cognome;

  SELECT N_posti INTO num_posti_tavolo
  FROM Tavolo
  WHERE idTavolo = NEW.Tavolo;

  IF num_persone_cliente > num_posti_tavolo THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'Il numero di persone del cliente supera il numero di posti del tavolo';
  END IF;
END
```

- Trigger che a seguito dell'inserimento di un nuovo cliente aggiorna lo stato del tavolo ad egli associato a "occupato"

```
CREATE
DEFINER=`BasiGiada`@`%`
TRIGGER `basididati`.`cliente_AFTER_INSERT`
AFTER INSERT ON `basididati`.`cliente`
FOR EACH ROW
BEGIN
  UPDATE Tavolo
  SET Occupazione = 1
  WHERE idTavolo = NEW.Tavolo;
END
```

- Questo trigger evita che all'inserimento di una nuova comanda possano essere inserite pizze che non sono nel menù

```
CREATE DEFINER=`BasiGiada`@`%` TRIGGER `ordpizze_before_insert_trigger` BEFORE INSERT ON `ordpizze` FOR EACH ROW BEGIN
  -- Verifica se il Pizza_Nome esiste nella tabella Pizze
  IF NOT EXISTS (SELECT 1 FROM Pizze WHERE Nome = NEW.Pizze_Nome) THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'Il nome della Pizza specificato non esiste nella tabella Pizze.';
  END IF;
END
```

- Questo trigger evita che all'inserimento di una nuova comanda si possa inserire una bevanda che non è nel menù

```
CREATE DEFINER=`BasiGiada`@`%` TRIGGER `ordbev_BEFORE_INSERT` BEFORE INSERT ON `ordbev` FOR EACH ROW BEGIN
IF NOT EXISTS (SELECT 1 FROM Bevande WHERE Nome_bevanda = NEW.Bevande_Nome_bevanda) THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'Il Bevande_Nome_bevanda specificato non esiste nella tabella Bevande.';
  END IF;
END
```

- Questo trigger imposta un tavolo come libero dopo che lo scontrino relativo ad esso è stato pagato

```
CREATE DEFINER = CURRENT_USER TRIGGER `basididati`.`scontrino_AFTER_UPDATE` AFTER UPDATE ON `scontrino` FOR EACH ROW
BEGIN
  DECLARE tavolo_id INT;

  -- Ottieni l'ID del tavolo relativo allo scontrino
  SELECT Tavolo INTO tavolo_id FROM scontrino WHERE idScontrino = NEW.idScontrino;

  -- Aggiorna l'occupazione del tavolo se lo stato di pagamento dello scontrino è stato impostato a 1
  IF NEW.isPagato = 1 THEN
    UPDATE Tavolo SET Occupazione = 0 WHERE idTavolo = tavolo_id;
  END IF;
END;
```

- Questo trigger dopo l'aggiornamento di un'ordinazione delle pizze va a controllare se le bevande inserite per la stessa ordinazione sono pronte così da segnare che la comanda relativa è pronta.

```
CREATE DEFINER=`BasiGiada`@`%` TRIGGER `ordpizze_AFTER_UPDATE` AFTER UPDATE ON `ordpizze` FOR EACH ROW BEGIN
  IF NEW.Pizza_pronta = 1 THEN
    -- Verifica se esiste un record in ordpizze con Pizza_pronta = 1 per lo stesso tavolo
    IF EXISTS (
      SELECT 1
      FROM ordbev
      WHERE comanda_Tavolo_ = NEW.comanda_Tavolo_
        AND Bevanda_pronta = 1
    ) THEN
      -- Aggiorna lo stato della comanda a 1
      UPDATE Comanda
      SET Stato = 1
      WHERE Data_ora = NEW.comanda_Data_ora
        AND Tavolo_ = NEW.comanda_Tavolo_;
    END IF;
```

```
  END IF;
END
```

- Questo trigger dopo l'aggiornamento di un'ordinazione delle bevande va a controllare se le pizze inserite per la stessa ordinazione sono pronte così da segnare che la comanda relativa è pronta.

```
CREATE DEFINER=`BasiGiada`@`%` TRIGGER `ordbev_AFTER_UPDATE` AFTER UPDATE ON `ordbev` FOR EACH ROW BEGIN
  IF NEW.Bevanda_pronta = 1 THEN
    -- Verifica se esiste un record in ordpizze con Pizza_pronta = 1 per lo stesso tavolo
    IF EXISTS (
      SELECT 1
      FROM ordpizze
      WHERE comanda_Tavolo_ = NEW.comanda_Tavolo_
        AND Pizza_pronta = 1
    ) THEN
      -- Aggiorna lo stato della comanda a 1
      UPDATE Comanda
      SET Stato = 1
      WHERE Data_ora = NEW.comanda_Data_ora
        AND Tavolo_ = NEW.comanda_Tavolo_;
    END IF;
  END IF;
END
```

Eventi

- Le comande di giorni diversi a quello corrente verranno eliminate

```
CREATE EVENT eliminaComande
ON SCHEDULE
  EVERY 1 DAY
DO
  DELETE FROM Comanda WHERE Data_ora < NOW();
```

- I clienti di giorni diversi a quello corrente verranno eliminati

```
CREATE EVENT EliminaClienti
ON SCHEDULE
  EVERY 1 DAY
DO
  DELETE c
  FROM Cliente c
  INNER JOIN Comanda co ON c.Tavolo = co.Tavolo_
  WHERE co.Data_ora < CURDATE() - INTERVAL 1 DAY;
```

- Gli scontrini verranno tenuti per la contabilità per 10 anni poi verranno cancellati

```
CREATE EVENT EliminaScontriniVecchi
ON SCHEDULE
  EVERY 1 DAY
DO
  DELETE FROM Scontrino WHERE DataEmissione < CURDATE() - INTERVAL 10 YEAR;
```

Viste

Non è stato implementato l'uso delle viste.

Stored Procedures e transazioni

- Questa procedura permette al manager di modificare il menu bevande inserendo una nuova bevanda

```
CREATE DEFINER=`Manager`@`%` PROCEDURE `AddBevandaMenu`(IN nome_p VARCHAR(255), IN prezzo_p FLOAT)
BEGIN
      declare exit handler for sqlexception
  begin
    rollback;
    resignal;
  end;
```

```
SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;
start transaction;
INSERT into bevande(Nome_bevanda, Costo)
VALUES (nome_p, prezzo_p);
commit;
END
```

- Questa procedura permette al manager di modificare il menu pizze inserendo una nuova pizza

```
CREATE DEFINER=`Manager`@`%` PROCEDURE `AddPizzaMenu`(IN nome_p VARCHAR(255), IN prezzo_p FLOAT)
BEGIN
        declare exit handler for sqlexception
   begin
      rollback;
      resignal;
   end;
SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;
start transaction;
INSERT into Pizze(Nome, Prezzo)
VALUES (nome_p, prezzo_p);
commit;
END
```

- Questa procedura permette al manager di aggiungere un nuovo tavolo fra quelli disponibili

```
CREATE DEFINER=`Manager`@`%` PROCEDURE `aggiungiNuovoTavolo`(IN numeroPosti INT, IN CamerieреID INT)
BEGIN
DECLARE nuovo_idTavolo INT;
        declare exit handler for sqlexception
   begin
      rollback;
      resignal;
   end;
```

```
        set transaction isolation level read committed;
    start transaction;
#evito la lettura sporca dell id se c'e un inserimento concorrente
INSERT INTO Tavolo(N_posti) VALUES (numeroPosti);
SET nuovo_idTavolo = LAST_INSERT_ID();
Call AssegnaCamerieraATavolo(nuovo_idTavolo,CamerieraID);
commit;
END
```

- Questa procedura permette al manager di assegnare un cameriere a un tavolo

```
CREATE DEFINER=`Manager`@`%` PROCEDURE `AssegnaCamerieraATavolo`(
    IN p_IDTavolo INT,
    IN p_IDCameriere INT
)
BEGIN
    -- Verifica se il cameriere esiste
    DECLARE cameriere_esiste INT;

    SELECT COUNT(*) INTO cameriere_esiste
    FROM Camerieri
    WHERE ID = p_IDCameriere;

    IF cameriere_esiste = 0 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Cameriere non valido.';
    ELSE
        -- Assegna il cameriere al tavolo
        UPDATE Tavolo
        SET Cameriere = p_IDCameriere
        WHERE idTavolo = p_IDTavolo;

        SELECT 'Cameriere assegnato con successo.' AS Messaggio;
    END IF;
```

END

- Questa procedura permette di avere una media delle entrate giornaliere, generabile dal manager

```
CREATE DEFINER=`Manager`@`%` PROCEDURE `CalcolaMediaEntrateGiornaliere`(IN p_Data DATE)
BEGIN
   -- Dichiarazione delle variabili
   DECLARE numero_scontrini INT;
   DECLARE totale_incassi DECIMAL(10, 2);
   DECLARE media_incassi DECIMAL(10, 2);
        declare exit handler for sqlexception
   begin
      rollback;
      resignal;
   end;

        set transaction isolation level repeatable read;
   start transaction;
   -- Calcola le statistiche giornaliere
   SELECT
      COUNT(idScontrino) AS NumeroScontrini,
      SUM(Totale) AS TotaleIncassi
   INTO numero_scontrini, totale_incassi
   FROM Scontrino
   WHERE DataEmissione = p_Data;

   -- Calcola la media degli incassi
   IF numero_scontrini > 0 THEN
      SET media_incassi = totale_incassi / numero_scontrini;
   ELSE
      SET media_incassi = 0;
```

```
    END IF;

    -- Restituisci la media degli incassi
    SELECT media_incassi AS MediaIncassi;
    commit;
END
```

- Questa procedura permette di avere una media delle entrate mensili, generabile dal manager

```
CREATE DEFINER=`Manager`@`%` PROCEDURE `CalcolaMediaEntrateMensili`(IN p_NumeroMese INT)
BEGIN
    -- Dichiarazione delle variabili
    DECLARE numero_scontrini INT;
    DECLARE totale_incassi DECIMAL(10, 2);
    DECLARE media_incassi DECIMAL(10, 2);
        declare exit handler for sqlexception
    begin
        rollback;
        resignal;
    end;

        set transaction isolation level repeatable read;
    start transaction;
    -- Calcola le statistiche mensili
    SELECT
        COUNT(idScontrino) AS NumeroScontrini,
        SUM(Totale) AS TotaleIncassi
    INTO numero_scontrini, totale_incassi
    FROM Scontrino
    WHERE MONTH(DataEmissione) = p_NumeroMese;

    -- Calcola la media degli incassi
    IF numero_scontrini > 0 THEN
        SET media_incassi = totale_incassi / numero_scontrini;
```

```
    ELSE
        SET media_incassi = 0;
    END IF;

    -- Restituisci la media degli incassi
    SELECT media_incassi AS MediaIncassi;
    commit;
END
```

- Questa procedura permette al manager di cancellare un cliente precedentemente registrato

```
CREATE DEFINER=`Manager`@`%` PROCEDURE `CancellaCliente`(IN nome_cliente varchar(255), IN cognome_cliente varchar(255))
BEGIN
DELETE FROM Cliente WHERE Nome= nome_cliente AND Cognome= cognome_cliente;
END
```

- Questa procedura permette al manager di modificare il menù eliminando una bevanda

```
CREATE DEFINER=`Manager`@`%` PROCEDURE `EliminaBevandaMenu`(IN p_NomeBevanda VARCHAR(255))
BEGIN
declare exit handler for sqlexception
    begin
        rollback;
        resignal;
    end;
SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
start transaction;
            -- altre transazioni possono modificare i dati mentre sta eseguendo ma l operazione
    -- non e' cosi importante da richiedere un livello piu alto
    -- Elimina la bevanda dalla tabella Bevande
    DELETE FROM Bevande
    WHERE Nome_bevanda = p_NomeBevanda;
```

```
   commit;
END
```

- Questa procedura permette al manager di modificare il menù eliminando una pizza

```
CREATE DEFINER=`Manager`@`%` PROCEDURE `EliminaPizzaMenu`(IN p_NomePizza VARCHAR(255))
BEGIN
declare exit handler for sqlexception
   begin
      rollback;
      resignal;
   end;
SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
start transaction;
         -- altre transazioni possono modificare i dati mentre sta eseguendo ma l operazione
   -- non e' cosi importante da richiedere un livello piu alto
   -- Elimina la pizza dalla tabella Pizze
   DELETE FROM Pizze
   WHERE Nome = p_NomePizza;
   commit;
END
```

- Questa procedura complessa permette a un maager di generare lo scontrino per un determinato tavolo e fornisce un output che è il totale da pagare per il tavolo

```
CREATE DEFINER=`Manager`@`%` PROCEDURE `GeneraScontrino`(IN p_idTavolo INT, OUT p_Totale DECIMAL(10, 2))
BEGIN
  -- Dichiarazione delle variabili per il totale delle pizze e bevande
  DECLARE totale_pizze DECIMAL(10, 2) DEFAULT 0;
  DECLARE totale_bevande DECIMAL(10, 2) DEFAULT 0;
declare exit handler for sqlexception
    begin
        rollback;
```

```
        resignal;
      end;
    SET TRANSACTION ISOLATION LEVEL REPEATABLE READ ;
    start transaction;
       -- altre transazioni possono modificare i dati mentre sta eseguendo ma l operazione
      -- non e' cosi importante da richiedere un livello piu alto
      -- Elimina la bevanda dalla tabella Bevande
     -- Calcola il totale delle pizze
     SELECT SUM(OrdPizze.Q_tà * Pizze.Prezzo)
     INTO totale_pizze
     FROM OrdPizze
     JOIN Pizze ON OrdPizze.Pizze_Nome = Pizze.Nome
     WHERE OrdPizze.comanda_Tavolo_ = p_idTavolo;

     -- Calcola il totale delle bevande
     SELECT SUM(OrdBev.Quantità * Bevande.Costo)
     INTO totale_bevande
     FROM OrdBev
     JOIN Bevande ON OrdBev.Bevande_Nome_bevanda = Bevande.Nome_bevanda
     WHERE OrdBev.comanda_Tavolo_ = p_idTavolo;

     -- Calcola il totale complessivo
     SET p_Totale = totale_pizze + totale_bevande;
     INSERT INTO Scontrino (Totale, Tavolo, isPagato, DataEmissione)
     VALUES (p_Totale, p_idTavolo, 0, current_date());
     commit;
    END
```

- Questa procedura permette di visualizzare tutto il menù bevande

```
CREATE DEFINER=`Manager`@`%` PROCEDURE `GetBevandeNelMenu`()
BEGIN
declare exit handler for sqlexception
   begin
```

```
      rollback;
      resignal;
   end;
SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
start transaction;
         -- altre transazioni possono modificare i dati mentre sta eseguendo ma l operazione
   -- non e' cosi importante da richiedere un livello piu alto
   SELECT Nome_bevanda, Costo
   FROM Bevande;
   commit;
END
```

- Questa procedura permette a un cameriere di visualizzare le comande che sono pronte da consegnare

```
CREATE DEFINER=`Cameriere`@`%` PROCEDURE `GetComandeCameriere`(
   IN p_CamerieреID INT
)
BEGIN
declare exit handler for sqlexception
   begin
      rollback;
      resignal;
   end;
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ ;
start transaction;
   SELECT C.*
   FROM Comanda C
   JOIN Tavolo T ON C.Tavolo_ = T.idTavolo
   JOIN Camerieri CM ON T.Cameriere = CM.ID
   WHERE CM.ID = p_CamerieреID AND C.Stato = 1;
   commit;
END
```

- Questa procedura permette di visualizzare tutto il menù pizze

```
CREATE DEFINER=`Manager`@`%` PROCEDURE `GetPizzeNelMenu`()
BEGIN
declare exit handler for sqlexception
   begin
      rollback;
      resignal;
   end;
SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
start transaction;
   SELECT Nome, Prezzo
   FROM Pizze;
   commit;
END
```

- Questa procedura permette a un cameriere di prendere una nuova ordinazione per un determinato tavolo, la comanda verrà registrata con la data e l'ora in modo da permettere più ordinazioni per uno stesso tavolo

```
CREATE DEFINER=`Cameriere`@`%` PROCEDURE `InserisciNuovaComanda`(
   IN p_IDTavolo INT,
   IN p_Pizze_Nomi VARCHAR(255),
   IN p_QuantitaPizze INT,
   IN p_Bevande_Nomi VARCHAR(255),
   IN p_QuantitaBevande INT
)
BEGIN
declare exit handler for sqlexception
   begin
      rollback;
      resignal;
   end;
SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
start transaction;
```

```
    -- Assegna il timestamp corrente a una variabile
    SET @current_timestamp := CURRENT_TIMESTAMP;

    -- Inserimento della nuova comanda con la data e ora correnti
    INSERT INTO Comanda (Data_Ora, Tavolo_, Stato)
    VALUES (@current_timestamp, p_IDTavolo, 0);  -- Stato iniziale della comanda

    -- Recupero l'ID della nuova comanda
    SET @idComanda = CONCAT(@current_timestamp, '_', p_IDTavolo);
    SET @comanda_Tavolo_ = SUBSTRING_INDEX(@idComanda, '_', -1);

    -- Inserimento delle pizze nella tabella OrdPizze con la data e ora correnti
    INSERT INTO OrdPizze (Q_tà, Pizza_pronta, comanda_Data_ora, comanda_Tavolo_,
  Pizze_Nome)
    VALUES (p_QuantitaPizze, 0, @current_timestamp, @comanda_Tavolo_, p_Pizze_Nomi);

    -- Inserimento delle bevande nella tabella OrdBev con la data e ora correnti
    INSERT INTO OrdBev (Quantità, Bevanda_pronta, comanda_Data_ora, comanda_Tavolo_,
  Bevande_Nome_bevanda)
    VALUES (p_QuantitaBevande, 0, @current_timestamp, @comanda_Tavolo_,
  p_Bevande_Nomi);

  commit;
  END
```

- Questa procedura complessa permette al manager di registrare e al contempo assegnare un tavolo a un nuovo cliente

```
CREATE DEFINER=`Manager`@`%` PROCEDURE `NuovoCliente`(
  IN p_Nome VARCHAR(255),
  IN p_Cognome VARCHAR(255),
  IN p_N_persone INT,
  OUT idTavoloAssegnato INT
)
BEGIN
```

```
        DECLARE
    tavolo_scelto INT;

    -- Trova il primo tavolo disponibile con posti sufficienti, ordinato per posti disponibili
    SELECT idTavolo
    INTO tavolo_scelto
    FROM tavolo
    WHERE N_posti >= p_N_persone AND Occupazione = 0
    ORDER BY N_posti ASC
    LIMIT 1;

    -- Se trova un tavolo disponibile, crea il nuovo cliente e assegna il tavolo
    IF tavolo_scelto IS NOT NULL THEN
        -- Inserisce il nuovo cliente
        INSERT INTO cliente (Nome, Cognome, N_persone, Tavolo)
        VALUES (p_Nome, p_Cognome, p_N_persone, tavolo_scelto);

        -- Imposta l'Occupazione del tavolo a 1
        UPDATE tavolo
        SET Occupazione = 1
        WHERE idTavolo = tavolo_scelto;

      SET idTavoloAssegnato = tavolo_scelto;
    END IF;

END
```

- Procedura che permette al barista di segnare le bevande relative a un ordine (tutte le bevande dell'ordine devono essere pronte) come pronte

```
CREATE DEFINER=`Barista`@`%` PROCEDURE `segnaBevandapronta`(IN id_Tavolo INT, IN nomeBevandaP VARCHAR(255), IN quant INT)
BEGIN
        #tutte le bevande di un tavolo devono essere prodotte nello stesso momento
```

```
   declare exit handler for sqlexception
   begin
      rollback;
      resignal;
   end;
SET TRANSACTION ISOLATION LEVEL READ COMMITTED ;
start transaction;
 UPDATE OrdBev
   SET Bevanda_pronta = 1
   WHERE comanda_Tavolo_ = id_Tavolo AND Bevande_Nome_bevanda = nomeBevandaP AND
Quantità =quant;
   commit;

END
```

- Procedura che permette al pizzaiolo di segnare le pizze relative a un ordine (tutte le pizze dell'ordine devono essere pronte) come pronte

```
CREATE DEFINER=`Pizzaiolo`@`%` PROCEDURE `segnaPizzapronta`(IN id_Tavolo INT, IN
nomePizzaP VARCHAR(255), IN quantità INT)
BEGIN
     #tutte le pizze devono essere prodotte nello stesso momento
       declare exit handler for sqlexception
   begin
      rollback;
      resignal;
   end;
SET TRANSACTION ISOLATION LEVEL READ COMMITTED ;
start transaction;
 UPDATE OrdPizze
   SET Pizza_pronta = 1
   WHERE comanda_Tavolo_ = id_Tavolo AND Pizze_Nome = nomePizzaP AND Q_tà
=quantità;
commit;
END
```

- Procedura che permette al manager di segnare come pagato uno scontrino che era stato precedentemente generato

```
CREATE DEFINER=`Manager`@`%` PROCEDURE `segnaTavoloPagato`(IN idTavoloP INT)
BEGIN
   declare exit handler for sqlexception
   begin
      rollback;
      resignal;
   end;
SET TRANSACTION ISOLATION LEVEL READ COMMITTED ;
start transaction;
UPDATE Scontrino
SET isPagato =1
WHERE Tavolo=idTavoloP;

UPDATE Tavolo
SET Occupazione = 0
WHERE idTavolo = idTavoloP;
commit;
END
```

- Procedura che permette al manager di visualizzare per quali tavoli è stato emesso uno scontrino ma non è ancora stato pagato

```
CREATE DEFINER=`BasiGiada`@`%` PROCEDURE `TavoliCheDevonoPagare`()
BEGIN
SELECT *
FROM Scontrino
WHERE isPagato =0;
END
```

- Procedura che permette al manager di visualizzare i tavoli liberi

```
CREATE DEFINER=`Manager`@`%` PROCEDURE `TavoliDisponibili`( )
```

```
BEGIN
   declare exit handler for sqlexception
   begin
      rollback;
      resignal;
   end;
SET TRANSACTION ISOLATION LEVEL READ COMMITTED ;
start transaction;
SELECT *
   FROM tavolo
   WHERE Occupazione = 0;
   commit;
END
```

- Procedura eseguita dal cameriere che può vedere quali tavoli gli sono stati assegnati

```
CREATE DEFINER=`Cameriere`@`%` PROCEDURE `visualizzaTavoliAssegnati`(IN CamerieреID INT)
BEGIN
   declare exit handler for sqlexception
   begin
      rollback;
      resignal;
   end;
SET TRANSACTION ISOLATION LEVEL READ COMMITTED ;
start transaction;
SELECT idTavolo
FROM Tavolo
WHERE Cameriere = CamerieреID;
commit;
END
```

- Questa procedura permette al barista di visualizzare tutti gli ordini ancora da preparare in ordine di arrivo

```
CREATE DEFINER=`Barista`@`%` PROCEDURE `VisualizzaBevandeDaPreparare`()
BEGIN
   -- Seleziona il nome e la quantità delle bevande da preparare ordinato per comanda_data_ora
   SELECT Bevande_Nome_bevanda, Quantità
   FROM Ordbev
   WHERE Bevanda_pronta = 0
   ORDER BY comanda_data_ora;
END
```

- Questa procedura permette di visualizzare al pizzaiolo tutti gli ordini da preparare in ordine di arrivo

```
CREATE DEFINER=`Pizzaiolo`@`%` PROCEDURE `VisualizzaPizzeDaPreparare`()
BEGIN
   -- Seleziona il nome e la quantità delle pizze da preparare
   SELECT Pizze_Nome, Q_tà
   FROM Ordpizze
   WHERE Pizza_pronta = 0
    ORDER BY comanda_data_ora;
END
```

- LOGIN

```
CREATE DEFINER=`BasiGiada`@`%` PROCEDURE `login`(in var_username varchar(45), in var_pass varchar(45), out var_role INT)
BEGIN
        declare var_user_role varchar(45);

   select `ruolo` from `Utenti`
                where `username` = var_username
      and `password` = var_pass
      into var_user_role;

        -- See the corresponding enum in the client
```

```
        if var_user_role = 'Manager' then
                set var_role = 1;
        elseif var_user_role = 'Cameriere' then
                set var_role = 2;
        elseif var_user_role = 'Pizzaiolo' then
                set var_role = 3;
        elseif var_user_role = 'Barista' then
                set var_role = 4;
        else
                set var_role = 5;
        end if;
END
```