RELAZIONE PROGETTO BD – ILLENGO MIRCO 1045161

1. **Requisiti iniziali:**
    - Gli utenti devono poter registrarsi al sistema.
    - Gli utenti devono poter ricaricare il proprio borsellino per effettuare pagamenti.
    - Gli utenti devono poter utilizzare codici sconto per ottenere riduzioni sugli ordini.
    - Gli utenti devono poter effettuare ordini presso i ristoranti registrati.
    - Gli utenti devono poter visualizzare i ristoranti con informazioni dettagliate (nome, indirizzo, descrizione, costo di spedizione, immagine, valutazione).
    - Gli ordini devono includere le portate selezionate e le relative quantità.
    - Gli ordini evasi devono poter essere associati a rider per la consegna.
    - Gli ordini devono poter essere annullati o evasi con relativa tracciabilità.
    - Il sistema deve gestire chat tra utenti, rider e ristoranti.
    - Gli utenti devono poter lasciare recensioni per rider e ristoranti.

1.2. Glossario dei Termini

1. **Utente**
    - **Definizione**: Una persona che utilizza il sistema, che vuole ordinare del cibo da uno o più ristoranti
    - **Attributi**: nome, email, password, num_tel, indirizzo (via, civico), premium.
2. **Borsellino**
    - **Definizione**: Un'entità che rappresenta il borsellino, ovvero ciò che l'utente va a ricaricare quando deve pagare.
    - **Attributi**: metodo_pagamento, saldo, id_utente (chiave esterna riferita a `Utente`).
3. **Cod_sconto**
    - **Definizione**: Un'entità che rappresenta un codice sconto applicabile agli ordini.
    - **Attributi**: tipologia, codice, id_utente (chiave esterna riferita a `Utente`).
4. **Ristorante**
    - **Definizione**: Un'entità che rappresenta un ristorante registrato nel sistema.
    - **Attributi**: nome, indirizzo, descrizione, costo_spedizione, img, valutazione.
5. **Categorie**
    - **Definizione**: Un'entità che rappresenta tutte le possibili categorie di ristoranti.
    - **Attributi**: nome.
6. **Tipologia**
    - **Definizione**: Un'entità che rappresenta tutte le possibili categorie di un singolo ristorante.
    - **Attributi**: nome_rist (parte della chiave esterna riferita a `Ristorante`), ind_rist (altra parte della chiave esterna riferita a `Ristorante`), nome_cat (chiave esterna riferita a `Categoria`).
7. **Top_partner**
    - **Definizione**: Un'entità che rappresenta i ristoranti top partner del sistema.
    - **Attributi**: data, id_ristorante (chiave esterna riferita a `Ristorante`).
8. **Portate**
    - **Definizione**: Un'entità che rappresenta una portata del menu di un ristorante.
    - **Attributi**: titolo
9. **Piatto**
    - **Definizione**: Un'entità che rappresenta l'associazione tra una portata e un ristorante.
    - **Attributi**: img, prezzo, sconto, id_portata (chiave esterna riferita a `Portate`), id_ristorante (chiave esterna riferita a `Ristorante`).
10. **Ingredienti**
    - **Definizione**: Un'entità che rappresenta un ingrediente utilizzato nelle portate.
    - **Attributi**: ingrediente.

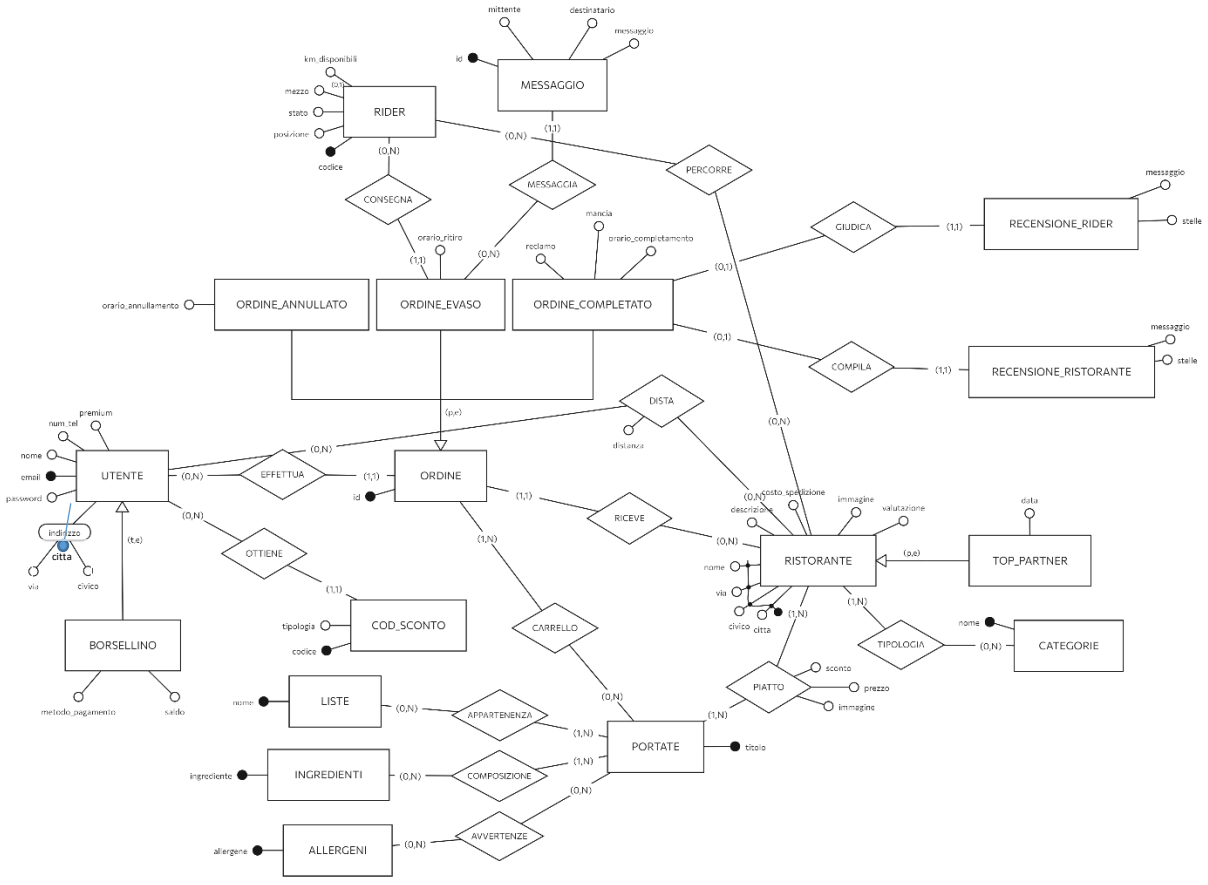
11. **Allergeni**
    - **Definizione**: Un'entità che rappresenta un allergene presente negli ingredienti.
    - **Attributi**: allergene.
12. **Composizione**
    - **Definizione**: Un'entità che rappresenta la composizione di una portata in termini di ingredienti.
    - **Attributi**: id_ingrediente (chiave esterna riferita a `Ingredienti`), id_portata (chiave esterna riferita a `Portate`).
13. **Avvertenze**
    - **Definizione**: Un'entità che rappresenta le avvertenze di allergeni presenti nelle portate.
    - **Attributi**: id_allergene (chiave esterna riferita a `Allergeni`), id_portata (chiave esterna riferita a `Portate`).
14. **Liste**
    - **Definizione**: Un'entità che rappresenta una lista creata dal ristorante.
    - **Attributi**: nome.
15. **Appartenenza**
    - **Definizione**: Un'entità che rappresenta l'associazione tra una lista e una portata.
    - **Attributi**: id, id_lista (chiave esterna riferita a `Liste`), id_portata (chiave esterna riferita a `Portate`).
16. **Ordini**
    - **Definizione**: Un'entità che rappresenta un ordine effettuato da un utente.
    - **Attributi**: id, id_ristorante (chiave esterna riferita a `Ristorante`), id_utente (chiave esterna riferita a `Utente`).
17. **Carrello**
    - **Definizione**: Un'entità che rappresenta le portate incluse in un ordine.
    - **Attributi**: id_ordine (chiave esterna riferita a `Ordini`), id_portate (chiave esterna riferita a `Portate`), quantità.
18. **Rider**
    - **Definizione**: Un'entità che rappresenta un rider che consegna gli ordini.
    - **Attributi**: codice, posizione, stato, mezzo, km_disp.
19. **Ordini_evasi**
    - **Definizione**: Un'entità che rappresenta gli ordini evasi e ritirati dal rider.
    - **Attributi**: id_ordine (chiave esterna riferita a `Ordini`), id_rider (chiave esterna riferita a `Rider`), orario_ritiro, orario_comp, reclamo.
20. **Ordini_annullati**
    - **Definizione**: Un'entità che rappresenta gli ordini annullati.
    - **Attributi**: id_ordine (chiave esterna riferita a `Ordini`), orario_annullamento.
21. **Chat**
    - **Definizione**: Un'entità che rappresenta una conversazione tra utenti del sistema, tra i quali l'utente, il rider ed il ristorante.
    - **Attributi**: id, id_ordine (chiave esterna riferita a `Ordini`), mittente, destinatario, messaggio.
22. **Ordini_completati**
    - **Definizione**: Un'entità che rappresenta gli ordini completati con successo.
    - **Attributi**: id_ordine_evaso (chiave esterna riferita a `Ordini_evasi`), mancia, orario_completamento, reclamo.
23. **Recensione_rider**
    - **Definizione**: Un'entità che rappresenta una recensione di un rider associata all'ordine fatto.
    - **Attributi**: id_ordine_evaso_completato (chiave esterna riferita a `Ordini_completati`), stelle, messaggio.
24. **Recensione_ristorante**
    - **Definizione**: Un'entità che rappresenta una recensione di un ristorante.

- **Attributi**: id_ordine_evaso_completato (chiave esterna riferita a `Ordini_completati`), stelle, messaggio.

1.3. Requisiti Rivisti

- **Gestione Utenti**: Registrazione, autenticazione, gestione del profilo, ricarica borsellino.
- **Gestione Ristoranti e Categorie**: Registrazione ristoranti, associazione categorie, visualizzazione ristoranti.
- **Gestione Ordini**: Creazione ordini, aggiunta portate al carrello, gestione codici sconto, tracciabilità ordini.
- **Gestione Consegne**: Assegnazione ordini ai rider, tracciamento consegne, gestione ordini evasi e annullati.
- **Comunicazioni**: Gestione chat tra utenti, rider e ristoranti.
- **Feedback**: Gestione recensioni per rider e ristoranti.

1.4. Schema E-R Principale + Business Rules



Regole Aziendali:

1. Ogni `Ristorante` può essere un `Top Partner` se rispetta i vincoli descritti nel testo del progetto.
2. Ogni `Utente` ha un indirizzo composto da tre parti, ovvero la via, il civico e la città, così come il `Ristorante`.

3. Ogni `Utente`, durante l'iscrizione, deve obbligatoriamente creare il suo borsellino inserendo il metodo di pagamento (di tipo enum che può valere ad esempio "bancomat", "carta di credito/debito", "Satispay") ed effettuando una prima ricarica di una cifra non inferiore a x (è a discrezione dello sviluppatore dell'app).
4. Ogni `Ordine` può finire o negli `Ordini evasi` oppure negli `Ordini annullati`. Una volta che è evaso, non può più essere annullato, e viceversa.
5. Sconto, prezzo e img di una portata sono inseriti all'interno della tabella di associazione `Piatto`, per rendere possibile ad ogni ristorante di avere la propria versione del piatto con possibilmente un'immagine, prezzo o sconto sul piatto diverso
6. Ogni `Rider` ha il suo mezzo, che può essere "monopattino", "bicicletta elettrica", "bicicletta normale". Se il mezzo è il monopattino allora l'attributo km_disponibili può anche valere NULL (il che vorrà dire che non ha km_disponibili), altrimenti deve obbligatoriamente essere NULL. Per quanto riguarda consegne dove la lunghezza in km è maggiore di 10, allora solo i rider muniti di bici elettrica possono effettuarla.
7. Ogni `Rider` ha uno stato, che è un attributo che può avere 3 valori: "Occupato", "Disponibile", "Fuori servizio".

2. Progettazione Logica

2.1. Tavola dei Volumi

Concetto	Tipo	Volume Stimato
Utenti	E	100,000
Borsellini	E	100,000
Codici Sconto	E	100,000
Ristoranti	E	10,000
Top partner	E	1,000
Categorie	E	200
Portate	E	100,000
Ordini	E	1,000,000
Rider	E	5,000
Ordini Evasi	E	800,000
Ordini Annullati	E	200,000
Messaggi	E	2,000,000
Ordini Completati	E	800,000
Recensioni Rider	E	800,000
Recensioni Rist	E	800,000
Liste	E	50
Ingredienti	E	1,000
Allergeni	E	100
Carrello	A	7,000,000
Dista	A	1,000,000,000
Percorre	A	50,000,000
Appartenenza	A	400,000

Concetto	Tipo	Volume Stimato
Composizione	A	1,000,000
Piatto	A	1,000,000
Tipologia	A	30,000
Avvertenze	A	200,000

Per quanto riguarda le associazioni, ho diviso il ragionamento in due parti:

- ASSOCIAZIONI STIMATE; tutte quelle associazioni il quale calcolo del volume è dato da una stima per una delle due entità collegate (es Carrello, ogni ordine ha in media 7 portate, quindi 7*n°ordini)

- ASSOCIAZIONI "PRECISE": le associazioni il cui volume, invece, è dato da tutte le possibili combinazioni (es, Dista contiene tutte le distanze per ogni utente per ogni ristorante, ergo N°utenti*N°ristoranti)

ASSOCIAZIONI STIMATE: Carrello, Appartenenza, Composizione, Piatto, Tipologia ed Avvertenze

ASSOCIAZIONI PRECISE: Dista, Percorre

2.2. Tavola delle Operazioni

Operazione	Frequenza	Descrizione
Registrazione Utente	Alta	Inserimento dati di un nuovo utente.
Elimina Utente	Bassa	Eliminazione dati di un utente già presente.
Ricarica Borsellino	Alta	Aggiunta di saldo al borsellino di un utente.
Applicazione Codice Sconto	Media	Applicazione di uno sconto ad un ordine.
Creazione Ordine	Alta	Inserimento di un nuovo ordine con associazione a utente e ristorante.
Aggiunta Portate al Carrello	Alta	Inserimento di portate al carrello di un ordine.
Assegnazione Ordine a Rider	Alta	Associazione di un ordine ad un rider per la consegna.
Gestione Chat	Alta	Invio e ricezione di messaggi tra utenti, rider e ristoranti.
Feedback su Rider e Ristorante	Media	Inserimento di recensioni per rider e ristoranti.

2.3. Ristrutturazione dello Schema E-R

2.3.1 Analisi delle Ridondanze:

Ho trovato alcune ridondanze nelle entità dello schema ER generalizzato. Le tabelle `Ordine_evasi` ed `Ordini_completati` dovrebbero essere fratelli e non padre e figlio, in quanto un'entità figlia (`Ordine_evasi`) non può diventare a sua volta padre. `Ordini_completati, Ordini_evasi ed Ordini_annullati` saranno dunque figli di `Ordini`.

Analisi della Ridondanza tra `Ordini_evasi` e `Ordini_completati`

Definizione delle Entità:

1. **Ordini_evasi:**
   - Contiene gli ordini che sono stati presi in carico e consegnati dal rider.
   - Attributi: `id_ordine`, `id_rider`, `orario_ritiro`, `orario_comp`.
2. **Ordini_completati:**
   - Contiene gli ordini che sono stati completati con successo e possono includere un reclamo.
   - Attributi: `id_ordine_evaso`, `id_recensione_rider`, `id_recensione_ristorante`, `mancia`, `reclamo`, `orario_completamento`.

Motivo per cui può essere ridondante:

1. **Duplicazione delle Informazioni:**
   - Se un ordine è completato, significa che è già stato evaso. Memorizzare l'ordine in entrambe le tabelle (`Ordini_evasi` e `Ordini_completati`) può duplicare informazioni come `id_ordine`, `id_rider`, `orario_ritiro`, `orario_comp`.
2. **Consistenza dei Dati:**
   - Mantenere due tabelle per rappresentare stati diversi dello stesso processo (evasione e completamento) richiede che i dati siano mantenuti consistenti tra di loro. Questo può introdurre complessità aggiuntiva e possibilità di incongruenze.
3. **Utilizzo di Spazio:**
   - Memorizzare informazioni duplicate richiede più spazio di archiviazione.

Esempio di Navigazione e Accesso:

Schema di Navigazione:

- **In Presenza di Ridondanza:**
  - Recuperare un ordine completato richiede una join tra `Ordini_evasi` e `Ordini_completati`.
- **In Assenza di Ridondanza:**
  - Tutte le informazioni sono presenti in una singola tabella, semplificando le query.

Tavola degli Accessi:

Operazione	In Presenza di Ridondanza	In Assenza di Ridondanza
Aggiunta di un Ordine Evasi	Inserimento in `Ordini_evasi` e successivamente in `Ordini_completati`	Inserimento in una singola tabella `Ordini` con stato
Modifica di un Ordine Evasi	Aggiornamento in `Ordini_evasi` e `Ordini_completati`	Aggiornamento in `Ordini`
Eliminazione di un Ordine Evasi	Rimozione da `Ordini_evasi` e `Ordini_completati`	Rimozione da `Ordini`
Ricerca di un Ordine Evasi	Join tra `Ordini_evasi` e `Ordini_completati`	Query su `Ordini` con condizione di stato

Confronto in Spazio e Tempo:

- **Presenza di Ridondanza:**
  - **Spazio:** Più spazio necessario per memorizzare le stesse informazioni in due tabelle diverse.
  - **Tempo:** Più tempo per eseguire join e mantenere la consistenza dei dati.
- **Assenza di Ridondanza:**
  - **Spazio:** Meno spazio necessario, le informazioni sono centralizzate.
  - **Tempo:** Operazioni più veloci e semplici da gestire senza necessità di join.

Scelta: Non Introdurre la Ridondanza

Motivazione:

- Centralizzare le informazioni in una singola tabella con un attributo che rappresenta lo stato dell'ordine (evaso o completato) semplifica la gestione e mantiene la consistenza dei dati. Riduce l'uso di spazio e velocizza le operazioni di query e aggiornamento.

Proposta di Ristrutturazione:

- Trasformare i figli di Ordine, ovvero Ordini_evasi, Ordini_completati e Ordini_annullati in entità connesse agli ordini tramite associazioni (1, 1)

2.3.2 Eliminazione delle Generalizzazioni

Non essendo presenti generalizzazioni, non ho dovuto cambiare niente

2.3.3 Partizionamento/Accorpamento di Entità e Associazioni

Le generalizzazioni presenti nello schema ER sono:

- Ordine – Ordini_evasi, Ordini_completati, Ordini_annullati (risolta come descritto precedentemente)

- Ristorante – Top_partner (accorpato il figlio al padre)

- Utente – Borsellino (accorpato il figlio al padre)

2.3.4 Eliminazione degli Attributi Composti e Multivalore

Gli unici due attributi composti presenti nel mio modello ER erano gli indirizzi del ristorante e dell'utente, composti da via, civico e città. Quello di Ristorante, però, si presentava già

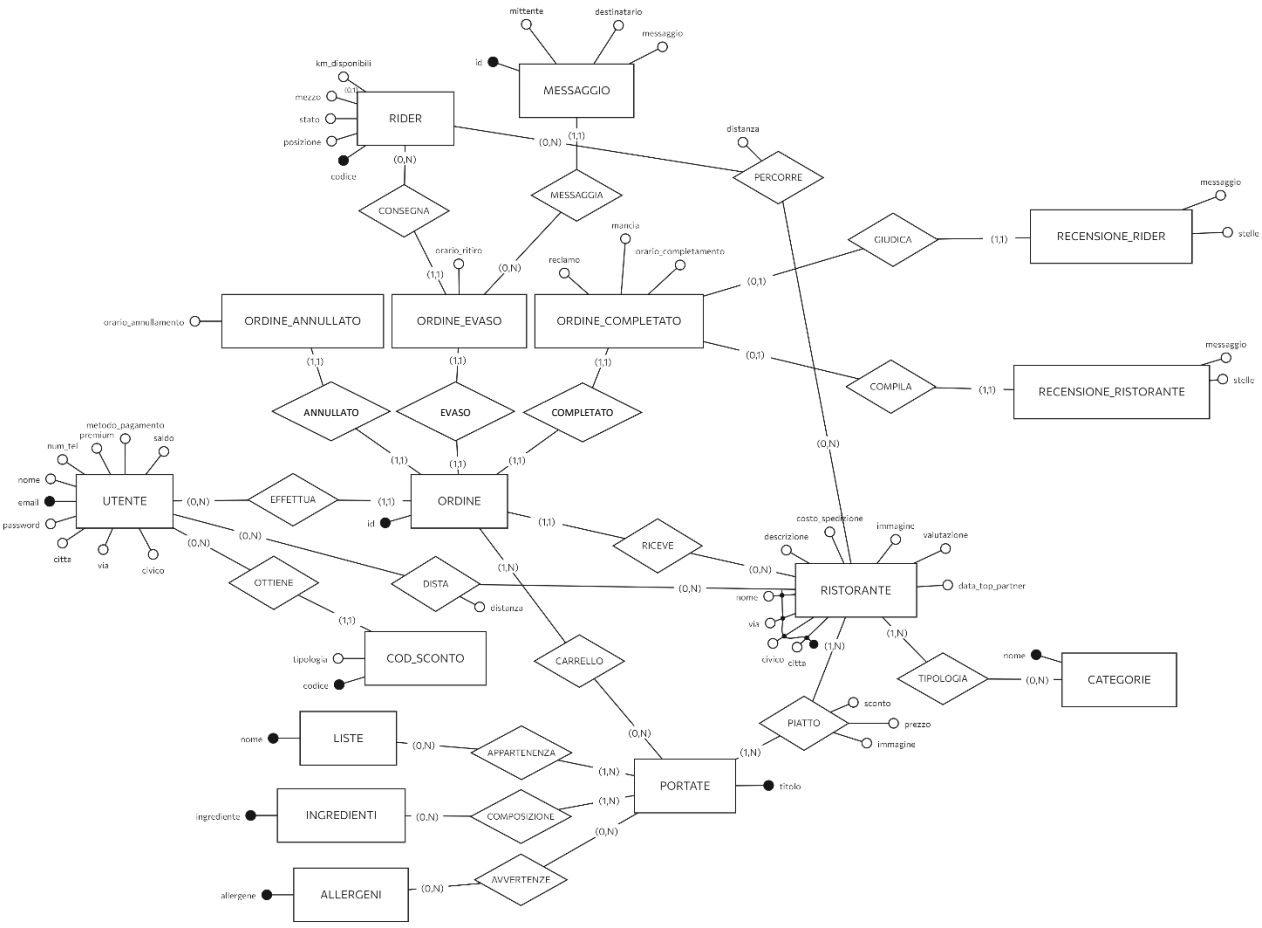
diviso in attributi semplici, per limitazioni implementative del software di sviluppo che ho utilizzato (non mi permetteva di impostare come chiave primaria un attributo composto)

Non erano presenti attributi multivalore.

2.3.5 Scelta degli Identificatori Principali:

Non è stato necessario aggiungere identificatori artificiali

2.4. Schema E-R ristrutturato + business rules



Regole Aziendali:

1. Ogni `Ristorante` può essere un `Top Partner` se rispetta i vincoli descritti nel testo del progetto. Se lo è, allora l'attributo data avrà la data in cui lo è diventato, se invece l'attributo data è NULL, significa che non lo è ancora diventato. Questo crea un campo possibilmente ridondante ma ottimizza il numero di attributi e tabelle
2. Ogni `Utente` non ha più un indirizzo composto da tre parti, ovvero la via, il civico e la città, ma è stato diviso nei 3 singoli attributi, così come il `Ristorante`.

3. Ogni `Utente`, durante l'iscrizione, deve obbligatoriamente creare il suo borsellino inserendo il metodo di pagamento (di tipo enum che può valere ad esempio "bancomat", "carta di credito/debito", "Satispay") ed effettuando una prima ricarica di una cifra non inferiore a x (è a discrezione dello sviluppatore dell'app).
4. Ogni `Ordine` può finire o negli `Ordini evasi` oppure negli `Ordini annullati`. Una volta che è evaso, non può più essere annullato, e viceversa. Dopo che è stato evasi, finirà sicuramente anche negli `Ordini completati`.
5. Sconto, prezzo e img di una portata sono inseriti all'interno della tabella di associazione `Piatto`, per rendere possibile ad ogni ristorante di avere la propria versione del piatto con possibilmente un'immagine, prezzo o sconto sul piatto diverso
6. Ogni `Rider` ha il suo mezzo, che può essere "monopattino", "bicicletta elettrica", "bicicletta normale". Se il mezzo è il monopattino allora l'attributo km_disponibili deve essere NOT NULL, altrimenti deve obbligatoriamente essere NULL. Per quanto riguarda consegne dove la lunghezza in km è maggiore di 10, allora solo i rider muniti di bici elettrica possono effettuarla.
7. Ogni `Rider` ha uno stato, che è un attributo che può avere 3 valori: "Occupato", "Disponibile", "Fuori servizio".

2.5. Schema relazionale con vincoli referenziali

- Utente; nome, email, password, num_tel, via, civico, citta, premium, saldo, metodo_pagamento.

- Cod_sconto; tipologia, codice, email_utente*

- Ristorante; nome, via, civico, citta, descrizione, costo_spedizione, immagine, valutazione, data_top_partner

- Tipologia; nome_ristorante*, via_ristorante*, civico_ristorante*, citta_ristorante*, nome_categoria*

- Categorie; nome

- Portate; titolo

- Piatto; immagine, prezzo, sconto, titolo_portata*, nome_ristorante*, via_ristorante*, civico_ristorante*, citta_ristorante*

- Ingredienti; ingrediente

- Allergeni; allergene

- Composizione; ingrediente_ingrediente*, titolo_portata*

- Avvertenze; allergene_allergene*, titolo_portata*

- Liste; nome

- Appartenenza; nome_lista*, titolo_portata*

- Ordine; id, nome_ristorante*, via_ristorante*, civico_ristorante*, citta_ristorante*, email_utente*

- Carrello; id_ordine*, titolo_portata*, quantità

- Rider; codice, posizione, stato, mezzo, km_disponibili

- Ordine_evaso; id_ordine*, codice_rider*, orario_ritiro

- Ordine_annullato; id_ordine*, orario_annullamento
- Messaggio; id, id_ordine_evasi*, mittente, destinatario, messaggio
- Ordine_completato; id_ordine *, mancia, orario_completamento, reclamo
- Recensione_rider; id_ordine_completato*, stelle, messaggio
- Recensione_ristorante; id_ordine_completato*, stelle, messaggio
- Dista; nome_ristorante*, via_ristorante*, civico_ristorante*, citta_ristorante*, email_utente*, distanza
- Percorre; nome_ristorante*, via_ristorante*, civico_ristorante*, citta_ristorante*, codice_rider*, distanza

LEGENDA:

***: CHIAVE ESTERNA _: CHIAVE PRIMARIA**

3. Implementazione

3.1. DDL di creazione del database

```
CREATE TABLE Utente (
   email VARCHAR(100) PRIMARY KEY,
   nome VARCHAR(100) NOT NULL,
   password VARCHAR(100) NOT NULL,
   num_tel VARCHAR(15),
   via VARCHAR(100) NOT NULL,
   civico VARCHAR(10),
   citta VARCHAR(100) NOT NULL,
   premium BOOLEAN NOT NULL,
   saldo FLOAT NOT NULL,
   metodo_pagamento VARCHAR(100) NOT NULL
);

CREATE TABLE Cod_sconto (
   codice VARCHAR(20) PRIMARY KEY,
   tipologia VARCHAR(50),
   email_utente VARCHAR(100),
   FOREIGN KEY (email_utente) REFERENCES Utente(email)
      ON DELETE CASCADE
```

```
    ON UPDATE CASCADE
);

CREATE TABLE Ristorante (
  nome VARCHAR(100),
  via VARCHAR(100),
  civico VARCHAR(10),
  citta VARCHAR(100),
  descrizione TEXT,
  costo_spedizione FLOAT,
  immagine VARCHAR(200) NOT NULL,
  valutazione FLOAT,
  data_top_partner DATE,
  PRIMARY KEY (nome, via, civico, citta)
);

CREATE TABLE Categorie (
  nome VARCHAR(100) PRIMARY KEY
);

CREATE TABLE Tipologia (
  nome_ristorante VARCHAR(100),
  via_ristorante VARCHAR(100),
  civico_ristorante VARCHAR(10),
  citta_ristorante VARCHAR(100),
  nome_categoria VARCHAR(100),
  PRIMARY KEY (nome_ristorante, via_ristorante, civico_ristorante, citta_ristorante, nome_categoria),
  FOREIGN KEY (nome_ristorante, via_ristorante, civico_ristorante, citta_ristorante) REFERENCES
Ristorante(nome, via, civico, citta)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  FOREIGN KEY (nome_categoria) REFERENCES Categorie(nome)
```

```
    ON DELETE CASCADE
    ON UPDATE CASCADE
);

CREATE TABLE Portate (
  titolo VARCHAR(100) PRIMARY KEY
);

CREATE TABLE Piatto (
  immagine VARCHAR(200),
  prezzo FLOAT NOT NULL,
  sconto DECIMAL(4, 2),
  titolo_portata VARCHAR(100),
  nome_ristorante VARCHAR(100),
  via_ristorante VARCHAR(100),
  civico_ristorante VARCHAR(10),
  citta_ristorante VARCHAR(100),
  PRIMARY KEY (titolo_portata, nome_ristorante, via_ristorante, civico_ristorante, citta_ristorante),
  FOREIGN KEY (titolo_portata) REFERENCES Portate(titolo)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  FOREIGN KEY (nome_ristorante, via_ristorante, civico_ristorante, citta_ristorante) REFERENCES
Ristorante(nome, via, civico, citta)
    ON DELETE CASCADE
    ON UPDATE CASCADE
);

CREATE TABLE Ingredienti (
  ingrediente VARCHAR(100) PRIMARY KEY
);

CREATE TABLE Allergeni (
```

```
    allergene VARCHAR(100) PRIMARY KEY
);

CREATE TABLE Composizione (
    ingrediente_ingrediente VARCHAR(100),
    titolo_portata VARCHAR(100),
    PRIMARY KEY (ingrediente_ingrediente, titolo_portata),
    FOREIGN KEY (ingrediente_ingrediente) REFERENCES Ingredienti(ingrediente)
        ON DELETE CASCADE
        ON UPDATE CASCADE,
    FOREIGN KEY (titolo_portata) REFERENCES Portate(titolo)
        ON DELETE CASCADE
        ON UPDATE CASCADE
);

CREATE TABLE Avvertenze (
    allergene_allergene VARCHAR(100),
    titolo_portata VARCHAR(100),
    PRIMARY KEY (allergene_allergene, titolo_portata),
    FOREIGN KEY (allergene_allergene) REFERENCES Allergeni(allergene)
        ON DELETE CASCADE
        ON UPDATE CASCADE,
    FOREIGN KEY (titolo_portata) REFERENCES Portate(titolo)
        ON DELETE CASCADE
        ON UPDATE CASCADE
);

CREATE TABLE Liste (
    nome VARCHAR(100) PRIMARY KEY
);

CREATE TABLE Appartenenza (
```

```
    nome_lista VARCHAR(100),
    titolo_portata VARCHAR(100),
    PRIMARY KEY (nome_lista, titolo_portata),
    FOREIGN KEY (nome_lista) REFERENCES Liste(nome)
        ON DELETE CASCADE
        ON UPDATE CASCADE,
    FOREIGN KEY (titolo_portata) REFERENCES Portate(titolo)
        ON DELETE CASCADE
        ON UPDATE CASCADE
);

CREATE TABLE Ordine (
    id INT PRIMARY KEY AUTO_INCREMENT,
    nome_ristorante VARCHAR(100),
    via_ristorante VARCHAR(100),
    civico_ristorante VARCHAR(10),
    citta_ristorante VARCHAR(100),
    email_utente VARCHAR(100),
    FOREIGN KEY (nome_ristorante, via_ristorante, civico_ristorante, citta_ristorante) REFERENCES
Ristorante(nome, via, civico, citta)
        ON DELETE CASCADE
        ON UPDATE CASCADE,
    FOREIGN KEY (email_utente) REFERENCES Utente(email)
        ON DELETE CASCADE
        ON UPDATE CASCADE
);

CREATE TABLE Carrello (
    id_ordine INT,
    titolo_portata VARCHAR(100),
    quantità INT NOT NULL,
    PRIMARY KEY (id_ordine, titolo_portata),
```

```
    FOREIGN KEY (id_ordine) REFERENCES Ordine(id)
        ON DELETE CASCADE
        ON UPDATE CASCADE,
    FOREIGN KEY (titolo_portata) REFERENCES Portate(titolo)
        ON DELETE CASCADE
        ON UPDATE CASCADE
);

CREATE TABLE Rider (
    codice VARCHAR(50) PRIMARY KEY,
    posizione VARCHAR(100) NOT NULL,
    stato ENUM('occupato', 'disponibile', 'fuori servizio') NOT NULL,
    mezzo ENUM('monopattino, 'bicicletta elettrica', 'bicicletta normale') NOT NULL,
    km_disponibili FLOAT CHECK (mezzo = 'monopattino')
);

CREATE TABLE Ordine_evaso (
    id_ordine INT,
    codice_rider VARCHAR(50) NOT NULL,
    orario_ritiro DATETIME NOT NULL,
    PRIMARY KEY (id_ordine),
    FOREIGN KEY (id_ordine) REFERENCES Ordine(id)
        ON DELETE CASCADE
        ON UPDATE CASCADE,
    FOREIGN KEY (codice_rider) REFERENCES Rider(codice)
        ON DELETE CASCADE
        ON UPDATE CASCADE
);

CREATE TABLE Ordine_annullato (
    id_ordine INT,
    orario_annullamento DATETIME NOT NULL,
```

```
    PRIMARY KEY (id_ordine),
    FOREIGN KEY (id_ordine) REFERENCES Ordine(id)
        ON DELETE CASCADE
        ON UPDATE CASCADE
);

CREATE TABLE Messaggio (
    id INT PRIMARY KEY AUTO_INCREMENT,
    id_ordine_evasi INT NOT NULL,
    mittente ENUM('rider', 'utente', 'ristorante') NOT NULL,
    destinatario ENUM('rider', 'utente', 'ristorante') NOT NULL,
    messaggio TEXT NOT NULL,
    FOREIGN KEY (id_ordine_evasi) REFERENCES Ordine_evaso(id_ordine)
        ON DELETE CASCADE
        ON UPDATE CASCADE
);

CREATE TABLE Ordine_completato (
    id_ordine INT,
    mancia FLOAT,
    orario_completamento DATETIME NOT NULL,
    reclamo TEXT,
    PRIMARY KEY (id_ordine),
    FOREIGN KEY (id_ordine) REFERENCES Ordine(id)
        ON DELETE CASCADE
        ON UPDATE CASCADE
);

CREATE TABLE Recensione_rider (
    id_ordine_completato INT,
    stelle INT NOT NULL,
    messaggio TEXT,
```

```
    PRIMARY KEY (id_ordine_completato),
    FOREIGN KEY (id_ordine_completato) REFERENCES Ordine_completato(id_ordine)
        ON DELETE CASCADE
        ON UPDATE CASCADE
);

CREATE TABLE Recensione_ristorante (
    id_ordine_completato INT,
    stelle INT NOT NULL,
    messaggio TEXT,
    PRIMARY KEY (id_ordine_completato),
    FOREIGN KEY (id_ordine_completato) REFERENCES Ordine_completato(id_ordine)
        ON DELETE CASCADE
        ON UPDATE CASCADE
);

CREATE TABLE Dista (
    nome_ristorante VARCHAR(100),
    via_ristorante VARCHAR(100),
    civico_ristorante VARCHAR(10),
    citta_ristorante VARCHAR(100),
    email_utente VARCHAR(100),
    distanza FLOAT NOT NULL,
    PRIMARY KEY (nome_ristorante, via_ristorante, civico_ristorante, citta_ristorante, email_utente),
    FOREIGN KEY (nome_ristorante, via_ristorante, civico_ristorante, citta_ristorante) REFERENCES
Ristorante(nome, via, civico, citta)
        ON DELETE CASCADE
        ON UPDATE CASCADE,
    FOREIGN KEY (email_utente) REFERENCES Utente(email)
        ON DELETE CASCADE
        ON UPDATE CASCADE
);
```

```
CREATE TABLE Percorre (
   nome_ristorante VARCHAR(100),
   via_ristorante VARCHAR(100),
   civico_ristorante VARCHAR(10),
   citta_ristorante VARCHAR(100),
   codice_rider VARCHAR(50),
   distanza FLOAT NOT NULL,
   PRIMARY KEY (nome_ristorante, via_ristorante, civico_ristorante, citta_ristorante, codice_rider),
   FOREIGN KEY (nome_ristorante, via_ristorante, civico_ristorante, citta_ristorante) REFERENCES
Ristorante(nome, via, civico, citta)
      ON DELETE CASCADE
      ON UPDATE CASCADE,
   FOREIGN KEY (codice_rider) REFERENCES Rider(codice)
      ON DELETE CASCADE
      ON UPDATE CASCADE
);


CREATE VIEW RidersPiuVeloci AS
SELECT
   Ordine_evaso.codice_rider,
   AVG(TIMESTAMPDIFF(MINUTE, Ordine_evaso.orario_ritiro, Ordine_completato.orario_completamento))
AS tempo_medio_consegna
FROM
   Ordine_evaso
JOIN
   Ordine_completato ON Ordine_evaso.id_ordine = Ordine_completato.id_ordine
GROUP BY
   Ordine_evaso.codice_rider
ORDER BY
   tempo_medio_consegna ASC;
```

```
CREATE VIEW CibiPiuPopolari AS
SELECT
   Carrello.titolo_portata,
   COUNT(Carrello.titolo_portata)*carrello.quantità AS numero_ordinazioni
FROM
   Carrello
GROUP BY
   Carrello.titolo_portata
ORDER BY
   numero_ordinazioni DESC;

CREATE VIEW RistorantiPiuRecensioniPositive AS
SELECT
   Ordine.nome_ristorante,
   Ordine.via_ristorante,
   Ordine.civico_ristorante,
   Ordine.citta_ristorante,
   COUNT(Recensione_ristorante.id_ordine_completato) AS numero_recensioni_positive
FROM
   Recensione_ristorante
JOIN
   Ordine_completato ON Recensione_ristorante.id_ordine_completato = Ordine_completato.id_ordine
JOIN
   Ordine ON Ordine_completato.id_ordine = Ordine.id
WHERE
   Recensione_ristorante.stelle >= 4
GROUP BY
   Ordine.nome_ristorante,
   Ordine.via_ristorante,
   Ordine.civico_ristorante,
   Ordine.citta_ristorante
ORDER BY
```

```
    numero_recensioni_positive DESC;


CREATE VIEW ClientiCheHannoSpesoDiPiu AS
SELECT
    Ordine.email_utente,
    SUM((Piatto.prezzo * Carrello.quantità) - (piatto.sconto * carrello.quantità)) AS totale_speso
FROM
    Ordine
JOIN
    Carrello ON Ordine.id = Carrello.id_ordine
JOIN
    Piatto ON Carrello.titolo_portata = Piatto.titolo_portata
        AND Ordine.nome_ristorante = Piatto.nome_ristorante
        AND Ordine.via_ristorante = Piatto.via_ristorante
        AND Ordine.civico_ristorante = Piatto.civico_ristorante
        AND Ordine.citta_ristorante = Piatto.citta_ristorante
GROUP BY
    Ordine.email_utente
ORDER BY
    totale_speso DESC;
```

3.2. DML di popolamento di tutte le tabelle del database

```
INSERT INTO Utente (email, nome, password, num_tel, via, civico, citta, premium, saldo,
metodo_pagamento)
VALUES
```

('mario.rossi@example.com', 'Mario Rossi', 'password123', '1234567890', 'Via Roma', '10', 'Roma', TRUE, 100.50, 'Carta di credito'),

('luigi.bianchi@example.com', 'Luigi Bianchi', 'securepassword', '0987654321', 'Via Milano', '20', 'Milano', FALSE, 50.00, 'PayPal');

INSERT INTO Cod_sconto (codice, tipologia, email_utente)

VALUES

('SCONTO10', '50%', 'mario.rossi@example.com'),

('WELCOME20', 'primo gratis', 'luigi.bianchi@example.com');

INSERT INTO Ristorante (nome, via, civico, citta, descrizione, costo_spedizione, immagine, valutazione, data_top_partner)

VALUES

('Ristorante A', 'Via Napoli', '15', 'Roma', 'Cucina Italiana', 5.00, 'img1.jpg', 4.5, '2023-01-01'),

('Ristorante B', 'Via Torino', '30', 'Milano', 'Cucina Vegetariana', 3.00, 'img2.jpg', 4.0, '2023-02-01');

INSERT INTO Categorie (nome)

VALUES

('Antipasti'),

('Primi Piatti'),

('Secondi Piatti'),

('Dolci');

INSERT INTO Tipologia (nome_ristorante, via_ristorante, civico_ristorante, citta_ristorante, nome_categoria)

VALUES

('Ristorante A', 'Via Napoli', '15', 'Roma', 'Antipasti'),

('Ristorante B', 'Via Torino', '30', 'Milano', 'Dolci');

INSERT INTO Portate (titolo)

VALUES

('Bruschette'),

('Carbonara'),

('Tiramisù');

INSERT INTO Piatto (immagine, prezzo, sconto, titolo_portata, nome_ristorante, via_ristorante, civico_ristorante, citta_ristorante)

VALUES

('img3.jpg', 8.50, 10.00, 'Bruschette', 'Ristorante A', 'Via Napoli', '15', 'Roma'),

('img4.jpg', 12.00, 0.00, 'Carbonara', 'Ristorante A', 'Via Napoli', '15', 'Roma'),

('img5.jpg', 6.00, 5.00, 'Tiramisù', 'Ristorante B', 'Via Torino', '30', 'Milano');

INSERT INTO Ingredienti (ingrediente)

VALUES

('Pomodori'),

('Pancetta'),

('Mascarpone');

INSERT INTO Allergeni (allergene)

VALUES

('Glutine'),

('Lattosio'),

('Uova');

INSERT INTO Composizione (ingrediente_ingrediente, titolo_portata)

VALUES

('Pomodori', 'Bruschette'),

('Pancetta', 'Carbonara'),

('Mascarpone', 'Tiramisù');

INSERT INTO Avvertenze (allergene_allergene, titolo_portata)

VALUES

('Glutine', 'Bruschette'),

('Uova', 'Carbonara'),

('Lattosio', 'Tiramisù');

```
INSERT INTO Liste (nome)
VALUES
('Lista A'),
('Lista B');

INSERT INTO Appartenenza (nome_lista, titolo_portata)
VALUES
('Lista A', 'Bruschette'),
('Lista B', 'Tiramisù');

INSERT INTO Ordine (nome_ristorante, via_ristorante, civico_ristorante, citta_ristorante, email_utente)
VALUES
('Ristorante A', 'Via Napoli', '15', 'Roma', 'mario.rossi@example.com'),
('Ristorante B', 'Via Torino', '30', 'Milano', 'luigi.bianchi@example.com');

INSERT INTO Carrello (id_ordine, titolo_portata, quantità)
VALUES
(1, 'Bruschette', 2),
(2, 'Tiramisù', 1);


INSERT INTO Rider (codice, posizione, stato, mezzo, km_disponibili)
VALUES
('R1', 'Centro', 'disponibile', 'bicicletta elettrica', NULL),
('R2', 'Nord', 'occupato', 'monopattino', 5.0);

INSERT INTO Ordine_evaso (id_ordine, codice_rider, orario_ritiro)
VALUES
(1, 'R1', '2024-07-01 10:00:00');

INSERT INTO Ordine_annullato (id_ordine, orario_annullamento)
```

```
VALUES
(2, '2024-07-01 11:00:00');

INSERT INTO Messaggio (id, id_ordine_evasi, mittente, destinatario, messaggio)
VALUES
(1, 1, 'utente', 'rider', 'Il mio ordine è pronto per il ritiro?');

INSERT INTO Ordine_completato (id_ordine, mancia, orario_completamento, reclamo)
VALUES
(1, 2.50, '2024-07-01 12:00:00', NULL);

INSERT INTO Recensione_rider (id_ordine_completato, stelle, messaggio)
VALUES
(1, 5, 'Ottimo servizio di consegna!');

INSERT INTO Recensione_ristorante (id_ordine_completato, stelle, messaggio)
VALUES
(1, 4, 'Il cibo era molto buono.');
INSERT INTO Dista (nome_ristorante, via_ristorante, civico_ristorante, citta_ristorante, email_utente, distanza)
VALUES
('Ristorante A', 'Via Napoli', '15', 'Milano', 'mario.rossi@example.com', 1.5),
('Ristorante B', 'Via Torino', '25', 'Roma', 'luigi.verdi@example.com', 3.0);

INSERT INTO Percorre (nome_ristorante, via_ristorante, civico_ristorante, citta_ristorante, codice_rider, distanza)
VALUES
('Ristorante A', 'Via Napoli', '15', 'Milano', 'R1', 2.0),
('Ristorante B', 'Via Torino', '25', 'Roma', 'R2', 4.0);
```

3.3. Qualche operazione di cancellazione e modifica

Prima di tutto testiamo la correttezza di uno dei percorsi più lunghi, ovvero quello tra utente e recensione ristorante (o recensione rider). In questa query andremo a selezionare l'id dell'utente che ha lasciato una recensione per un ristorante 'Il cibo era molto buono.'

```
SELECT Utente.nome
FROM Utente
JOIN Ordine ON Utente.email = Ordine.email_utente
JOIN Ordine_completato ON Ordine.id = Ordine_completato.id_ordine
JOIN Recensione_ristorante ON Ordine_completato.id_ordine =
Recensione_ristorante.id_ordine_completato
WHERE Recensione_ristorante.messaggio = 'Il cibo era molto buono.';
```

Il risultato della query appena fatta, se abbiamo popolato con le query sopra specificate il database, sarà:

nome

Mario Rossi

Dopodichè proviamo con la modifica, un esempio che dovrebbe fallire. Proverò ad inserire un ordine effettuato da un utente non registrato ad un ristorante non presente nel database

```
INSERT INTO Ordine (id, nome_ristorante, via_ristorante, civico_ristorante, citta_ristorante, email_utente)
VALUES (100, 'Ristorante Prova', 'Via Prova', '10', 'Città Prova', 'non_esistente@example.com');
```

La query infatti fallisce con messaggio di errore sui vincoli relazionali

Infine proviamo a testare una cancellazione considerando anche i vincoli ON DELETE cascade presenti nelle varie relazioni. Cancelliamo dunque il nostro 'Ristorante A' di 'Via Napoli' '15' a 'Roma'. Ciò che ci aspettiamo dalla seguente query è la corretta eliminazione del ristorante con successiva cancellazione anche di tutti gli ordini (quindi ordini evasi, annullati e completati), recensioni, piatti, tipologie (collegamento fra ristorante e categoria), carrelli ed eventuali messaggi nella chat legati agli ordini collegati al nostro ristorante. Procediamo dunque eseguendo questa query:

```
DELETE FROM Ristorante
WHERE nome = 'Ristorante A' AND via = 'Via Napoli' AND civico = '15' AND citta = 'Roma';
```

La query come ci aspettavamo esegue tutto ciò che abbiamo previsto. In questo modo abbiamo appurato che i vincoli di integrità e relazionali reggono ed il settaggio ON DELETE/ON UPDATE funziona.

Infine testiamo se la tabella dista e la tabella percorre funzionano come dovrebbero. Creiamo una query che calcoli la strada che dovrà percorrere il rider “R1” per ritirare il cibo dal ristorante “Ristorante A” e consegnarlo a “Mario Rossi”

```
SELECT
    (Dista.distanza + Percorre.distanza) AS distanza_totale
FROM
    Dista
JOIN
    Percorre
ON
    Dista.nome_ristorante = Percorre.nome_ristorante
    AND Dista.via_ristorante = Percorre.via_ristorante
    AND Dista.civico_ristorante = Percorre.civico_ristorante
    AND Dista.citta_ristorante = Percorre.citta_ristorante
WHERE
    Dista.email_utente = 'mario.rossi@example.com'
    AND Dista.nome_ristorante = 'Ristorante A'
    AND Dista.via_ristorante = 'Via Napoli'
    AND Dista.civico_ristorante = '15'
    AND Dista.citta_ristorante = 'Milano'
    AND Percorre.codice_rider = 'R1';
```

Il risultato della query è 3.5, quindi è corretto.