

# Progetto - 23

**Progetto d'esame basi di dati  
Corso di laurea in informatica  
Anno Accademico 2020/2021**

Studente  
**Gianpiero Spinelli**  
**Matr. 144647**

Gestione di menu con allergeni per i ristoranti

Menu e ristorante

Allergene

Piatto

Utente

Ristoratore

Recensione

Glossario

Schema scheletro e schema E/R

1. Utente

Gerarchia

Proprietà

Chiavi e schema

2. Ristorante

Proprietà

Chiavi e schema

3. Menù

Proprietà

Chiavi e schema

4. Allergene

Proprietà

Chiavi e schema

5. Recensione

Proprietà

Chiavi e schema

6. Schema scheletro finale

Progetto logico

1. Eliminazione delle gerarchie ISA

2. Selezione delle chiavi primarie e eliminazioni delle identificazioni esterne

3. Traduzione di entità e associazioni in schemi di relazioni

Operazioni SQL

Operazioni previste dalla base di dati – Descrizione e relativo codice SQL

Triggers

# Gestione di menu con allergeni per i ristoranti

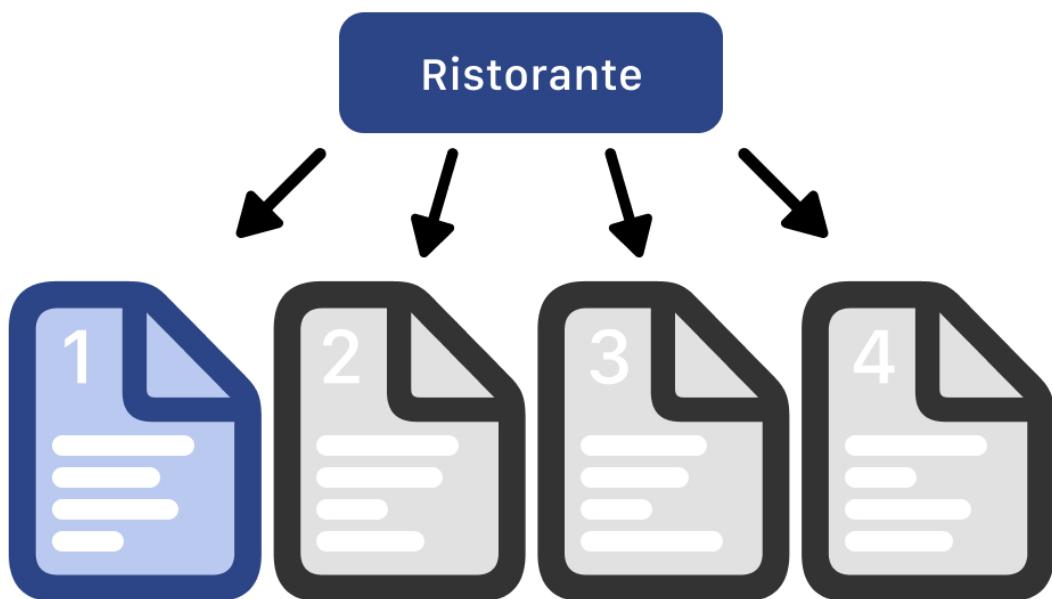
L'idea è quella di creare il database di supporto ad un sito web per permettere ai ristoratori di gestire i menù del proprio ristorante in modo facile e al contempo dettagliato. Per rendere l'esperienza di utilizzo più coinvolgente, il sito integra anche alcune componenti social.

Più che mai, in questo momento, è necessario offrire una versione online del proprio menù, in modo da poter consultare le pietanze, senza l'utilizzo di mezzi analogici difficili da sanificare.

## Menu e ristorante

Sito che contiene tutti i menu di tutti i ristoranti, facile da navigare e facile da gestire.

Ogni ristorante ha quindi un proprio menù attivo. Dal momento che solitamente i menù sono stagionali, i ristoranti hanno la possibilità di creare più di un menù, ma uno solo di questi potrà essere attivo (vedi immagine 1.1.0).



1.1.0

La **caratteristica**, però, che più caratterizza questo sito è la possibilità di avere sott'occhio tutti gli **allergeni contenuti nei vari piatti**.

## Allergene

Dal momento che il numero di persone con allergie sta aumentando di anno in anno, e che alcuni ristoranti ancora non hanno aggiornato i propri menù con tutti gli allergeni, il cambio di piattaforma potrebbe essere il giusto incentivo.

Ogni persona ha una allergia differente, ma i principali allergeni alimentari riconosciuti sono 14, e sono:

- Glutine
- Crostacei e derivati
- Uova e derivati
- Pesce e derivati
- Arachidi e derivati
- Soia e derivati
- Latte e derivati
- Frutta a guscio e derivati
- Sedano e derivati
- Senape e derivati
- Semi di sesamo e derivati
- Anidride solforosa e solfiti
- Lupino e derivati
- Molluschi e derivati

Si può quindi utilizzare questo elenco per gestire gli allergeni nel database. Ogni allergene è rappresentato da un codice numerico, e ha una denominazione.

## Piatto

Nel menù ci sarà una sezione dedicata alle allergie, con l'elenco degli allergeni contenuti in un determinato piatto, e un badge che indica se il piatto è vegano o vegetariano. Ogni piatto avrà un nome e un prezzo, e potrà avere una immagine e una descrizione opzionale.

Il **ristorante** sarà in grado quindi di pubblicare un menù con molte più informazioni di quelle che ci sono normalmente. Verrà pubblicato nella pagina del ristorante, che conterrà una **foto di copertina**, il **logo** del ristorante, il **numero di telefono**, l'**indirizzo** e un eventuale **link al sito web**.

I piatti potranno essere suddivisi in sezioni che stanno ad indicare le varie portate.

## Utente

I clienti si potranno registrare al sito per avere una visione più personalizzata. Ogni utente, inoltre, potrà recensire i ristorante, ma solo se quest'ultimo accetta le recensioni. Gli utenti accederanno al sito tramite email e password, e potranno impostare un'immagine profilo.

## Ristoratore

Il ristoratore ha la stessa rappresentazione di un utente normale. Quello che li differenzia è che il ristoratore ha una relazione `gestisce` con il proprio **ristorante**.

## Recensione

Gli utenti che hanno effettuato il login potranno lasciare una recensione ai ristoranti. Un utente potrà lasciare una sola recensione per ristorante.

## Glossario

Aa Termine	☰ Descrizione	≡ Sinonimi	↗ Relation	↗ Relation^t	Σ Legami
<u>Allergene</u>	<code>descrizione</code> <code>nome</code>		<u>Utente</u> , <u>Piatto</u>		Utente, Piatto
<u>Menu</u>	<code>data creazione</code> <code>attivo</code> <code>data aggiornamento</code> <code>nome</code>		<u>Ristorante</u> , <u>Piatto</u>		Ristorante, Piatto

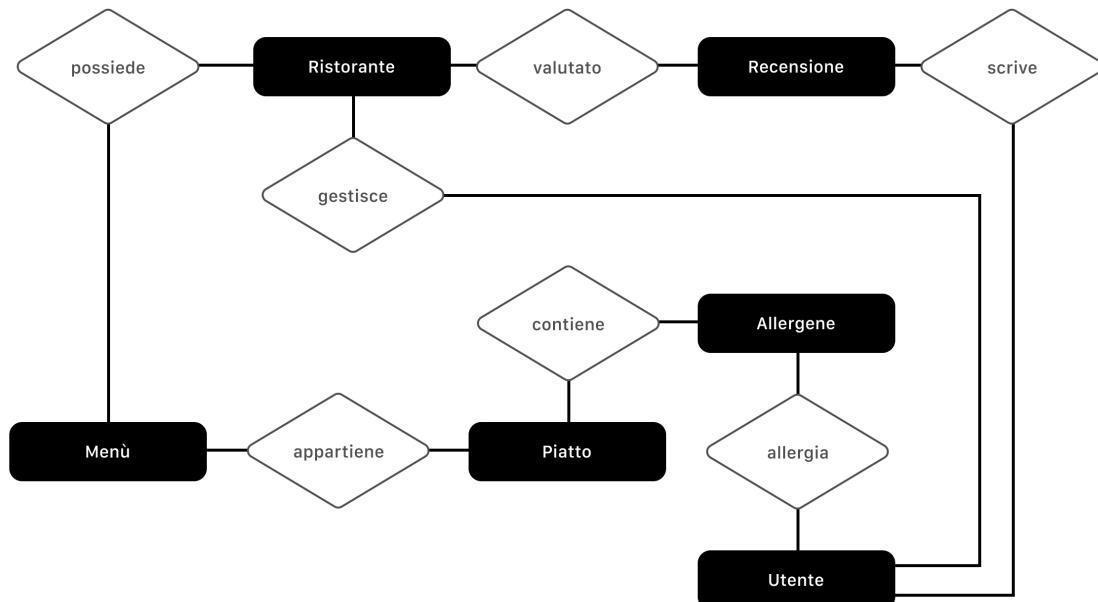
Aa Termine	Desrizione	Sinonimi	Relation	Relation^t	Legami
<u>Piatto</u>	descrizione foto nome portata posizione prezzo vegano vegetariano	Pietanza	<u>Portata</u>	<u>Allergene</u> , <u>Menu</u>	Allergene, Menu, Portata
<u>Recensione</u>	data testo valutazione	Review	<u>Utente</u> , <u>Ristorante</u>		Utente, Ristorante
<u>Ristorante</u>	accetta prenotazioni accetta recensioni immagine indirizzo logo nome sito web telefono	Locale	<u>Utente</u>	<u>Menu</u> , <u>Recensione</u>	Menu, Recensione, Utente
<u>Utente</u>	data registrazione email immagine nome password	Cliente, ristoratore, gestore, persona		<u>Allergene</u> , <u>Recensione</u> , <u>Ristorante</u>	Allergene, Recensione, Ristorante
<u>Portata</u>	nome		<u>Piatto</u>		Piatto

## Schema scheletro e schema E/R

Qui si rappresentano le relazioni tra le varie entità del database:

- menù - ristorante
- utente - ristorante
- utente - recensione
- recensione - ristorante
- allergene - piatto
- piatto - menù

- utente - allergene



© 2021 Gianpiero Spinelli.

Lo schema generale si può dividere in varie parti minori, che si possono identificare in:

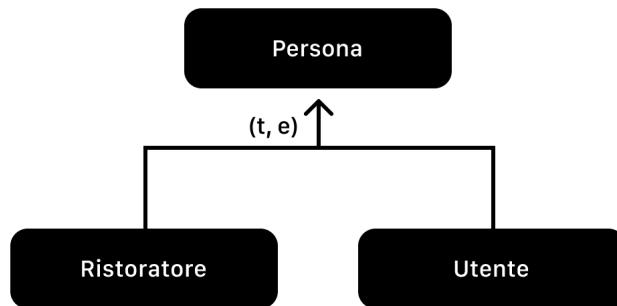
1. Utente
2. Ristorante
3. Menù
4. Allergene
5. Recensione

## 1. Utente

### Gerarchia

L'utente dell'amministratore del ristorante deve avere più privilegi degli utenti normali. Questo si può ottenere generalizzando la rappresentazione della persona. Il ristoratore è rappresentabile come un utente, ma può gestire un ristorante. La gerarchia utilizzata per rappresentare una persona è quindi:

- **totale** dal momento che non ci sono altri tipi di utenti da gestire;
- **esclusiva**, perché un ristoratore non può fare le cose di un utente normale, come ad esempio scrivere recensioni.



© 2021 Gianpiero Spinelli.

3.1.0

In questo caso, anche se al momento non ci sono differenze tra ristoratore e utente a livello di proprietà, si generalizza la persona, così da rendere anche il tutto espandibile in un secondo momento.

## Proprietà

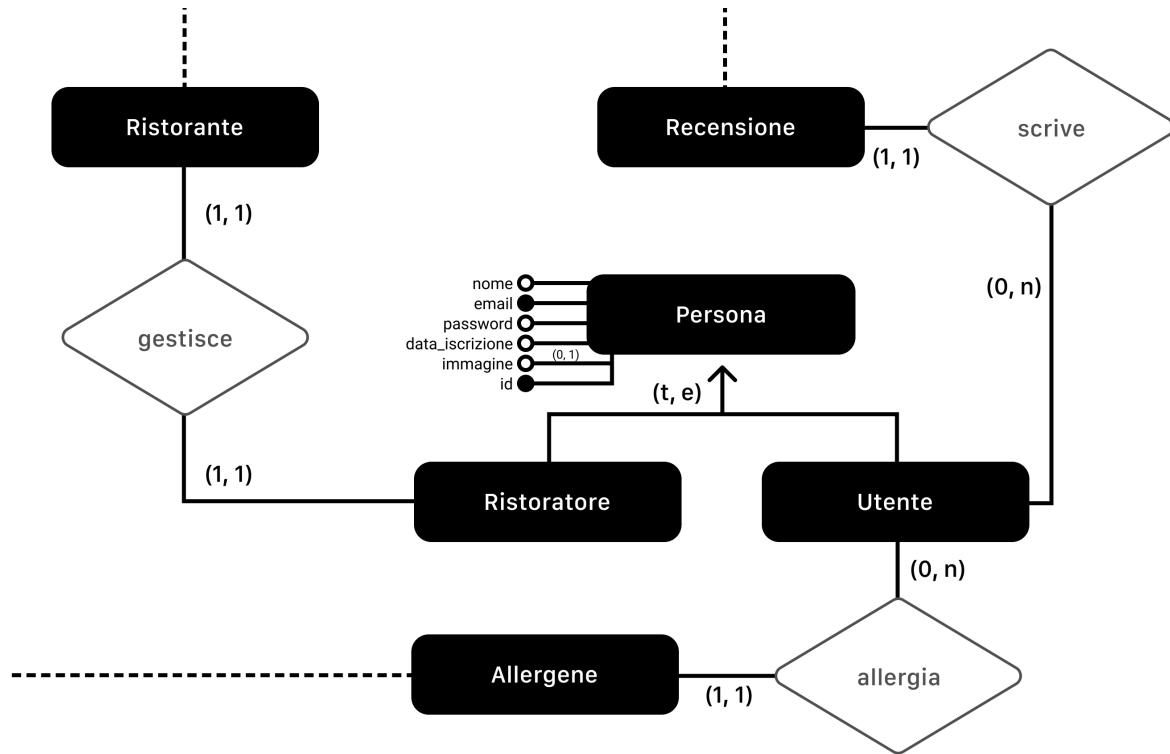
Le proprietà che condividono sono molte. La principale differenza è che il ristoratore ha una relazione con l'entità **ristorante**, mentre l'utente avrà una relazione con l'entità **recensione**.

## Chiavi e schema

Come chiave per gli utenti si utilizza uno UUID (così da anonimizzare un minimo le relazioni con gli utenti), e l'indirizzo email è constraint unique, perché due utenti non possono avere la stessa email. Dal momento che gli utenti sono principalmente persone fisiche, l'univocità del nome non è una parte interessante del progetto. Ogni

utente avrà poi una password, di cui verrà salvata la versione hash (sha1), per una questione di sicurezza.

Nel seguente schema, si evidenziano le proprietà delle persone, e si mostrano le relazioni di utente con recensione e allergene, e di ristoratore con ristorante.



3.1.1

## 2. Ristorante

### Proprietà

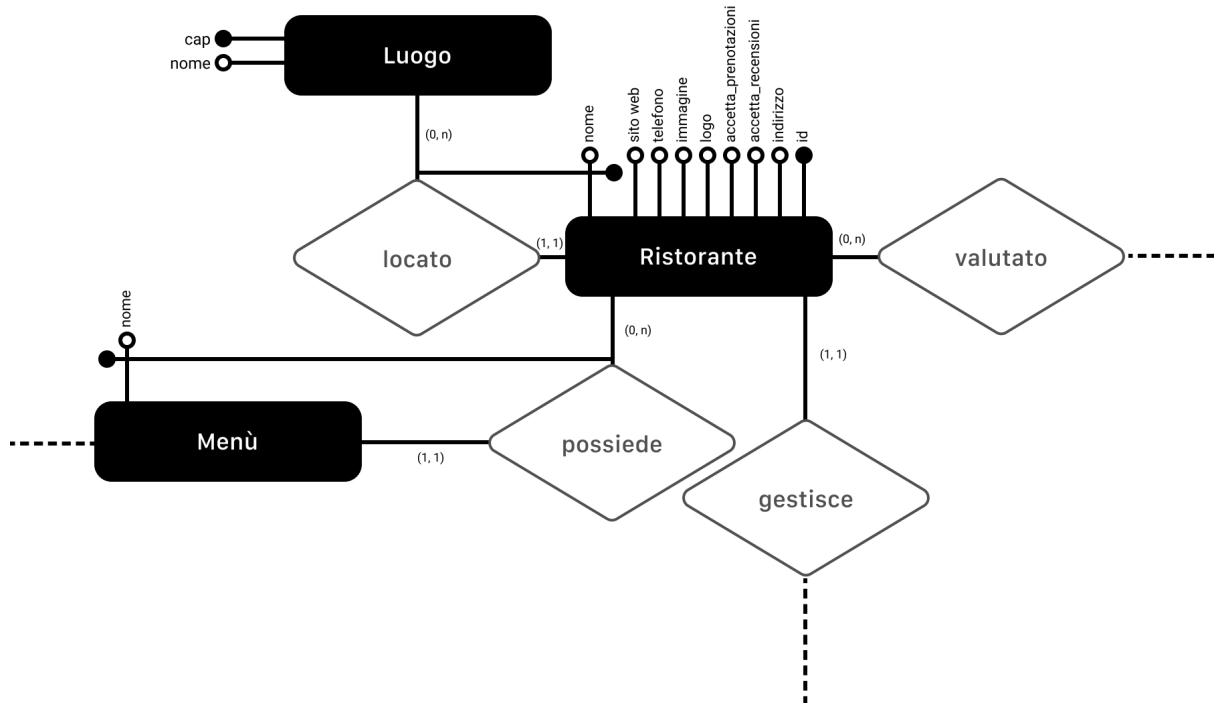
Le proprietà del ristorante sono evidenti dal glossario. Il nome del ristorante verrà utilizzato per comporre la chiave dell'entità, dal momento che solitamente il nome è unico in una data zona. Essendo lo scopo del progetto quello di pubblicare il proprio menù, ristorante avrà una relazione con menù. Ogni ristorante può avere più di un menù, e sarà cura del gestore pubblicarne solo uno alla volta.

## Chiavi e schema

Come chiave primaria per la rappresentazione del ristorante si opta per un UUID, in modo da avere una chiave semplice, e randomica.

Si utilizza anche una chiave secondaria composta dal nome del ristorante e dal codice di avviamento postale, un codice composto da 5 cifre che identifica uno specifico luogo.

Nel seguente schema, si evidenziano le proprietà del ristorante e le relazioni con menù, e luogo.



## 3. Menù

### Proprietà

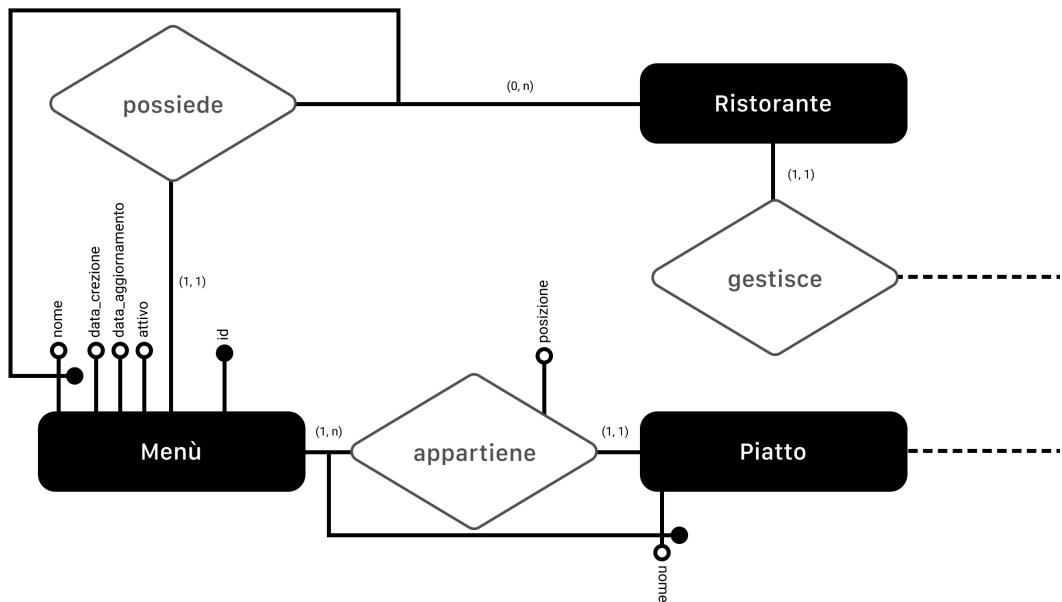
Anche in questo caso, come per l'entità ristorante, le proprietà del menù sono piuttosto evidenti dal glossario. Essendo il menù legato ad un singolo ristorante, si avrà una relazione con l'entità del ristorante.

La proprietà `attivo` sta ad indicare se quel menù è visibile o meno nella pagina del sito. Ci può essere un solo menù attivo per ristorante per volta. Per arrivare a questo livello si utilizzerà un trigger, di cui si parlerà più avanti.

## Chiavi e schema

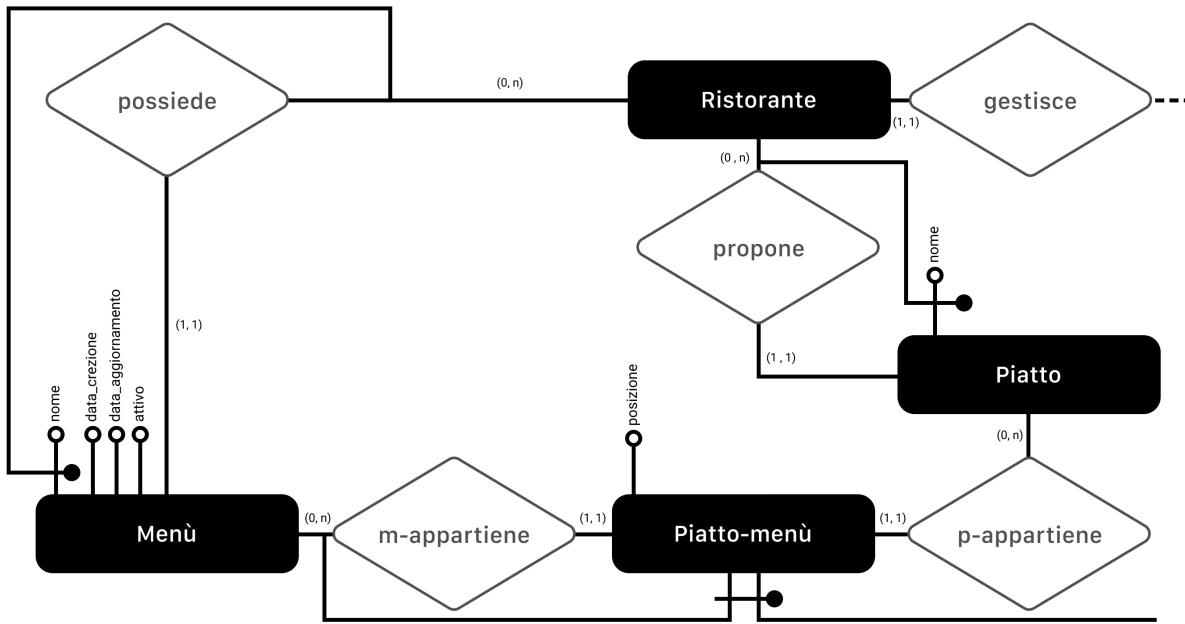
Per l'entità menù, come avveniva per il ristorante, si ha una chiave composta dal nome del menù, e dal ristorante di cui fa parte, per evitare che ci siano due menù con lo stesso nome, che possono causare confusione nella gestione degli stessi.

Menù è in relazione con ristorante, come già detto sopra, e con piatto, ovvero l'entità che rappresenta la pietanza. Quest'ultima relazione contiene una proprietà `posizione` che rappresenta la posizione del piatto nel menù, vedendo il menù come una lista di piatti.



3.3.0

In questo modo (immagine 3.3.0), però, bisogna inserire ogni piatto per ogni menù. Se un ristorante ha dei piatti che vengono proposti durante tutto l'anno, dovrebbe creare un duplice ogni volta che crea un nuovo menù. Per ovviare a questo problema, si può dereferenziare la relazione "appartiene", creando uno schema come quello che si può vedere nell'immagine 3.3.1.



3.3.1

Si inserisce nel quadro, quindi, una nuova entità denominata piatto-menù, che permette di condividere piatti tra i vari menù che un ristorante può avere. I piatti quindi sono collegati ad un ristorante, e possono essere inseriti in più menù contemporaneamente.

## 4. Allergene

L'idea di questo progetto si basa tutta sulla allergie, e quindi questa è la parte più importante. L'allergene ha relazioni con piatto e utente, dal momento che potrà selezionare le proprie allergie per avere una visualizzazione del menù ad hoc, in modo da poter vedere velocemente cosa può mangiare.

### Proprietà

Come rappresentato nel glossario, l'entità piatto ha un nome, e un prezzo. Ha poi una foto e una descrizione opzionali. L'entità si conclude con una relazione con portata, in modo da avere una selezione di portate e non lasciare la libertà al ristoratore di scrivere, e una con allergene.

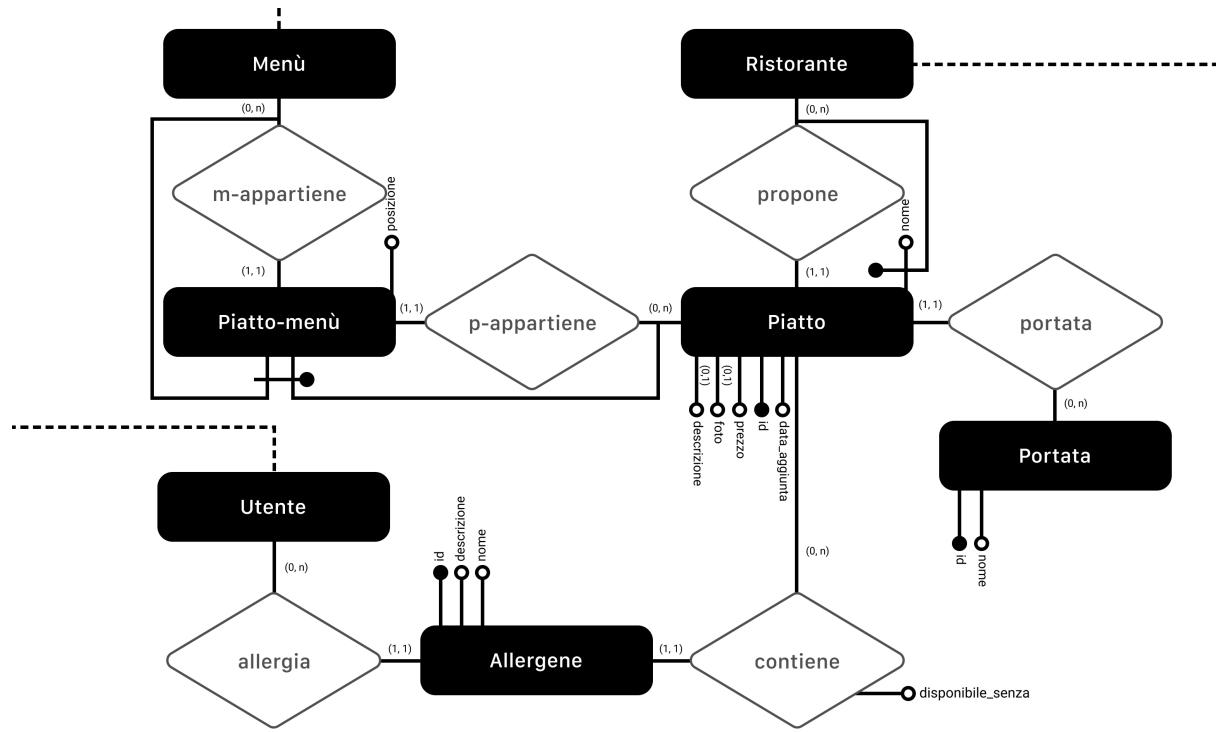
### Chiavi e schema

La chiave di piatto è uno UUID, più semplice da referenziare dalle relazioni. Il nome del piatto e dall'id esterno del ristorante che lo propone, invece, compongono una chiave alternativa, in modo da non avere duplicati dello stesso piatto in un unico

ristorante. Si usa la chiave del ristorante in modo da attribuire un piatto ad uno specifico ristorante, dal momento che se non fosse in nessun menù andrebbe perso.

Gli allergeni, dal momento che quelli riconosciuti sono 14 e sono quindi facilmente enumerabili, sono rappresentati da un numero intero univoco, `id`. Ogni allergene ha una descrizione, che contiene qualche esempio e qualche informazione in più sugli allergeni, in modo da rendere più facile la selezione da parte dei ristoratori nella compilazione del menù.

L'entità portata inizialmente è stata pensata come entità fissa, ovvero dove ogni ristorante ha le stesse portate. Questa implementazione potrebbe avere come id uno `smallint`, dal momento che le portate sono poche. Leggendo i menù di alcuni ristoranti, però, si è notato che molti utilizzano delle categorie personali. Per questo, l'id sarà di tipo `uuid`. Questo permetterà in un secondo di abilitare i ristoranti a crearsi le proprie portate, aggiungendo due attributi `ristorante` e `descrizione`.



3.4.0

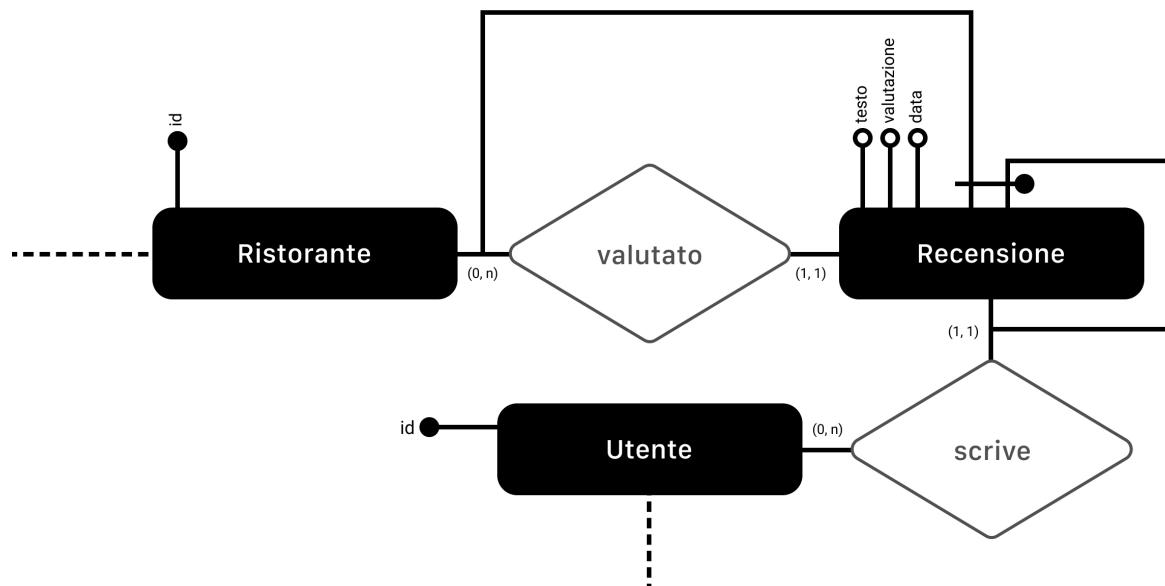
## 5. Recensione

## Proprietà

Le proprietà di una recensione sono facilmente deducibili dal glossario, e sono il testo della recensione, una valutazione da 0 a 5 (con un incremento di 0.5 alla volta), l'utente che l'ha scritta, il ristorante per cui è stata scritta e la data.

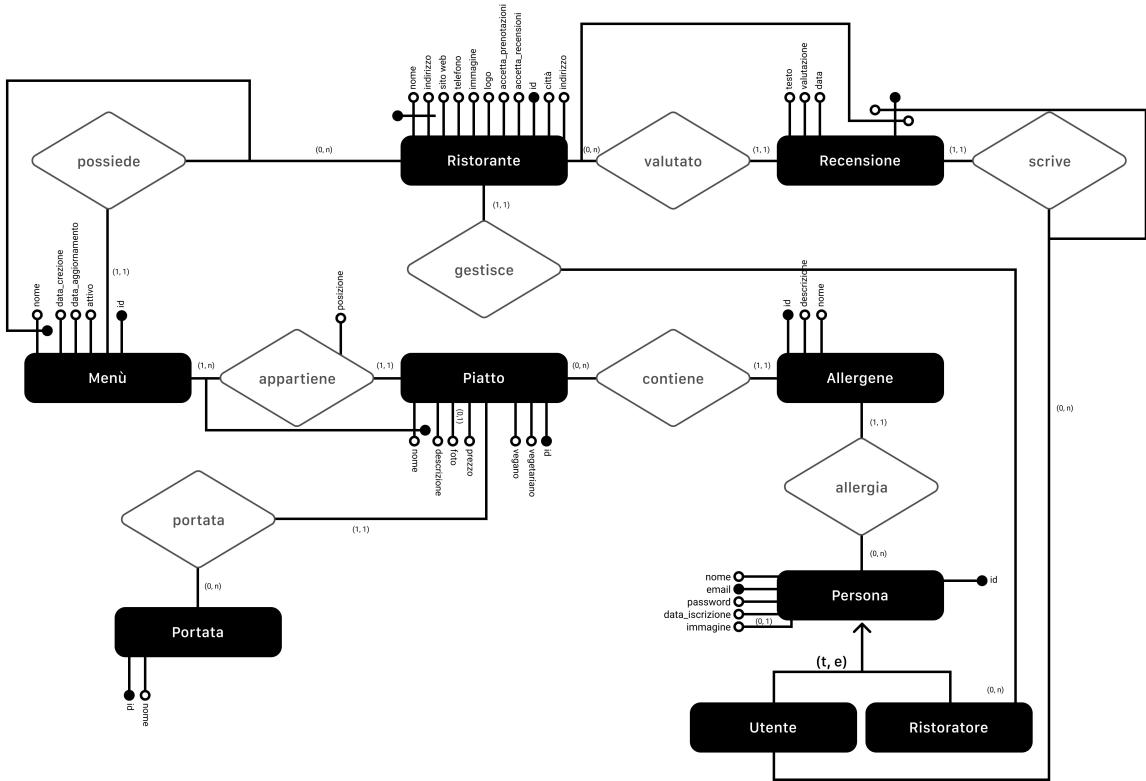
## Chiavi e schema

Per recensione si utilizza una chiave composta dalla chiave di utente e quella di ristorante, in modo da avere una recensione unica per utente per ristorante.



## 6. Schema scheletro finale

Lo schema finale è il seguente. Nel capitolo successivo andiamo a tradurre questo schema scheletro nel progetto logico.



5.1.0

## Progetto logico

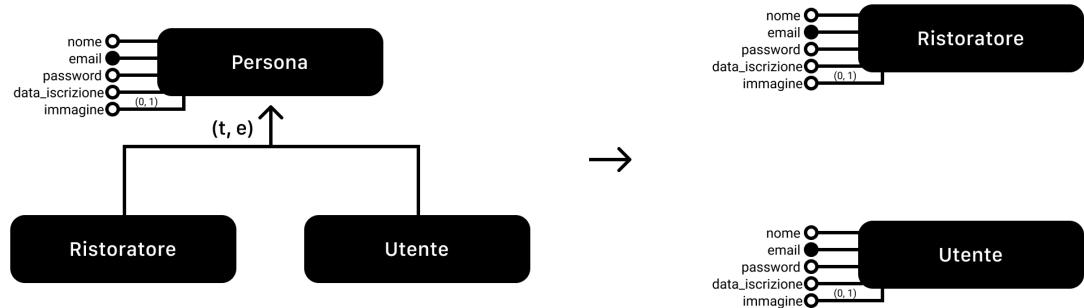
La progettazione logica della base di dati procede secondo il seguente ordine:

1. Eliminazione delle gerarchie ISA
2. Selezione delle chiavi primarie, eliminazione delle identificazioni esterne
3. Trasformazione degli attributi composti o multipli
4. Traduzione di entità e associazioni in schemi di relazioni
5. Verifica di normalizzazione

### 1. Eliminazione delle gerarchie ISA

Nel progetto si può trovare solo una gerarchia, ovvero quella per la rappresentazione degli utenti e dei ristoratori (vedi immagine 3.1.0).

In questo caso la scelta più conveniente è il **collasso verso il basso**, dal momento che l'entità padre non relazioni con alcuna altra entità. Questo rende anche più semplice introdurre nuove proprietà specifiche se in un futuro servisse.

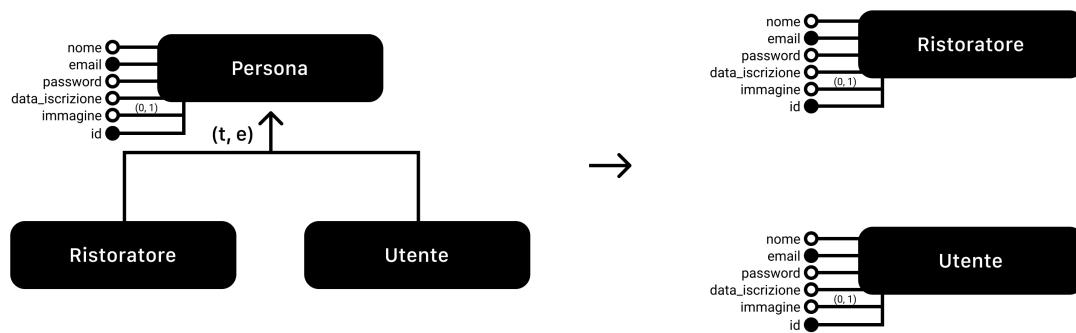


5.1.0

Verranno quindi generate due entità, utente e ristoratore, che conterranno tutte le proprietà di persona.

## 2. Selezione delle chiavi primarie e eliminazioni delle identificazioni esterne

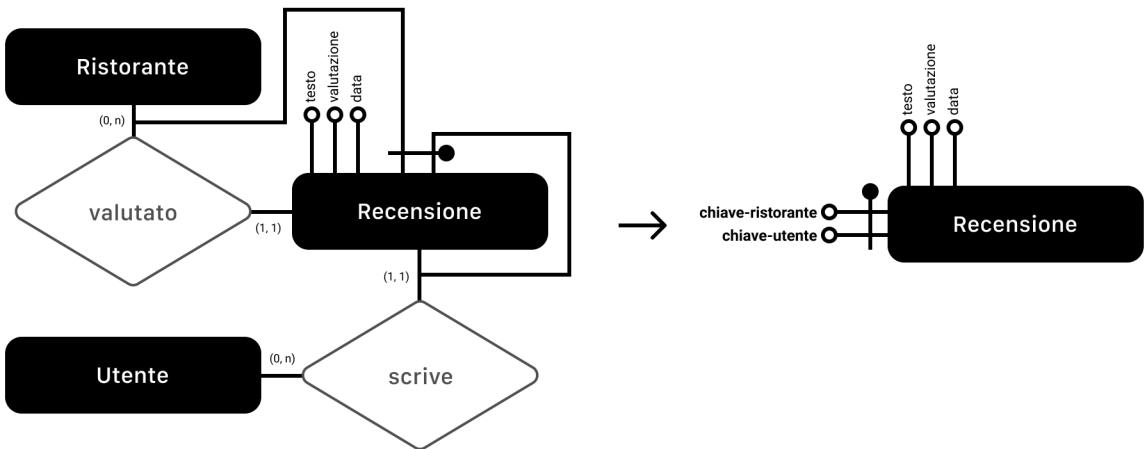
La gerarchia degli utenti **collassa verso il basso**, e si hanno quindi due diverse entità, una per ristoratore e una per utente, per una questione di future proof, in modo da non doverle separare in un futuro se si dovessero aggiungere troppe proprietà che differenziano i due utenti, e per avere un collegamento più semplice tra le entità ristorante e ristoratore. Entrambe le entità hanno una chiave primaria rappresentata da uno UUID (Universally unique identifier), e una chiave secondaria che rappresenta la mail dell'utente.



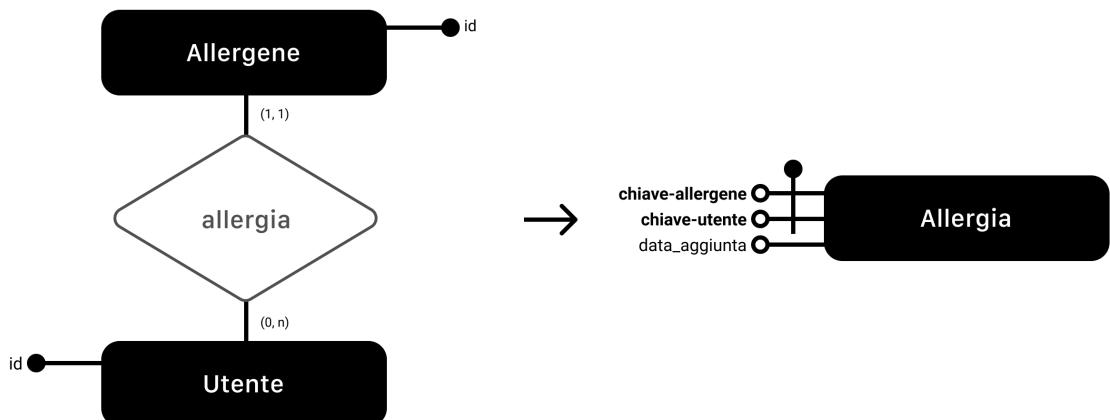
L'entità ristorante è rappresentata da uno UUID (Universally unique identifier), mentre la chiave secondaria si compone dalla cap (codice di avviamento postale), indirizzo e nome del ristorante. L'associazione "locato" viene inglobata nell'entità ristorante.



Recensione è identificata dalla chiave del ristorante, e dalla chiave dell'utente che scrive la recensione. Le associazioni "valutato" e "scrive" sono quindi inglobate nell'entità recensione.

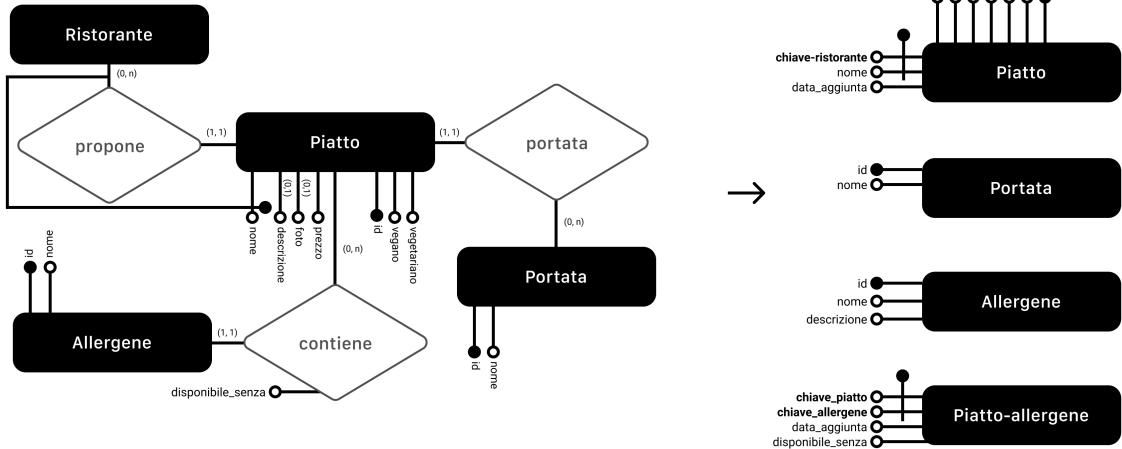


La relazione tra allergene e utente viene rappresentata come una entità, con una composizione di id dell'allergene e id dell'utente come chiave primaria, in modo da avere ogni allergene massimo una volta per utente.

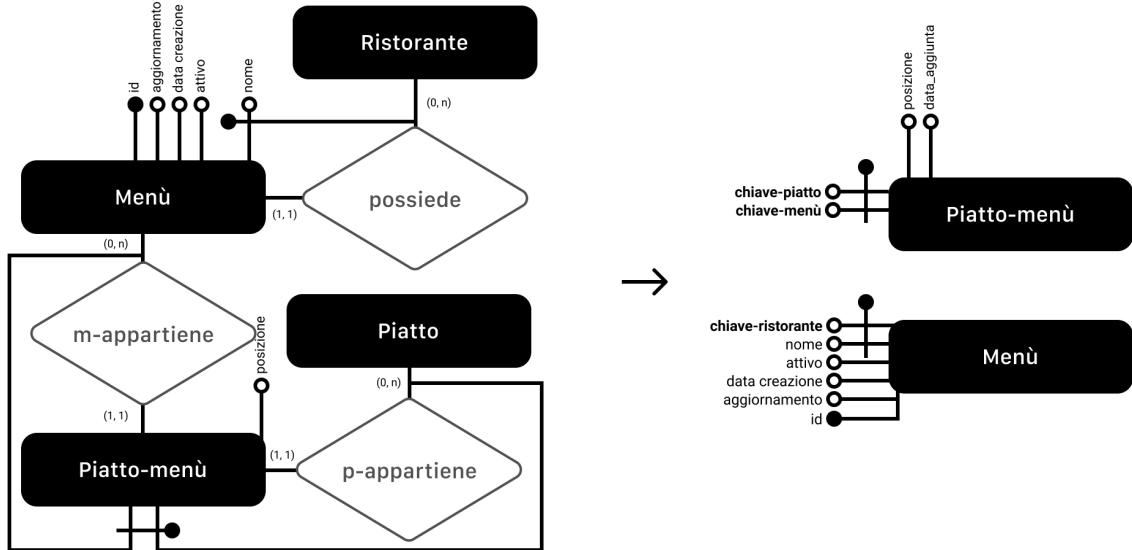


Passando al gruppo di cui fanno parte il ristorante e il piatto, si avranno quattro entità alla fine. Piatto sarà composta da tutte le sue proprietà e includerà la chiave di ristorante. La chiave primaria di piatto sarà uno UUID (Universally unique identifier). L'entità avrà una chiave secondaria, composta dalla chiave del ristorante e dal nome del piatto stesso.

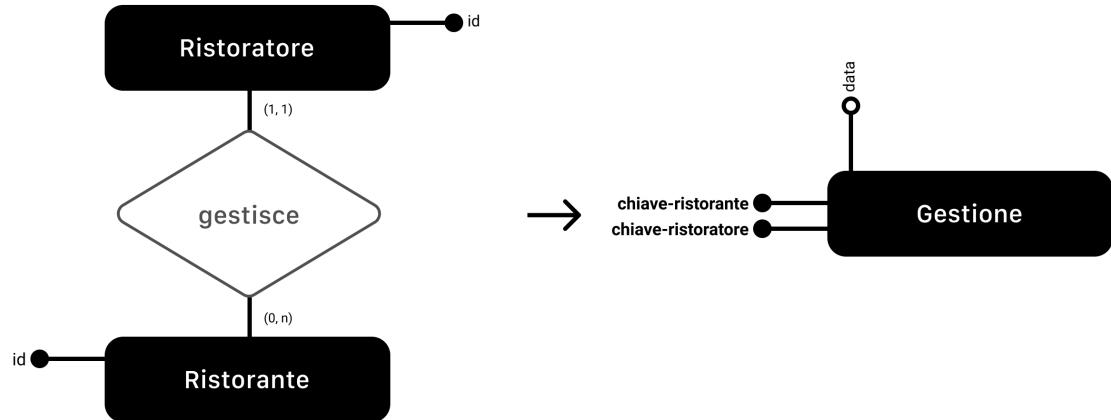
La relazione "contiene" tra piatto e allergene, diventa una entità, in modo da avere più allergeni collegati ad uno stesso piatto. Questa nuova entità avrà come chiave primaria la combinazione dell'id di allergene e dell'id del piatto.



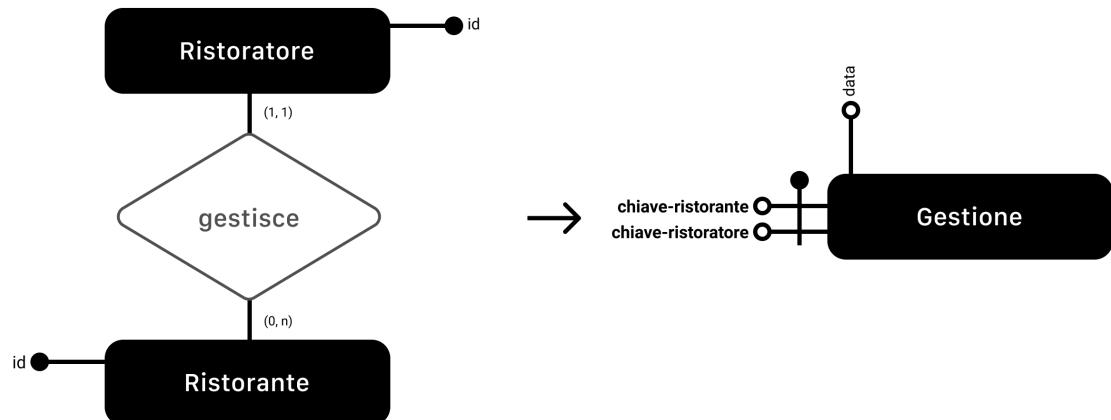
In questo caso la relazione piatto-menù si dereferenzia. Si ottengono quindi due entità: menù e piatto-menù. Piatto-menù ha come chiave primaria la combinazione della chiave di piatto e di quella di menù.



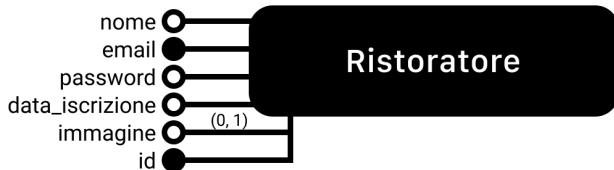
La relazione gestione tra il ristoratore e il ristorante, viene implementata tramite una entità a parte. All'inizio un ristoratore potrà gestire un solo ristorante, e un ristorante potrà gestire un solo ristoratore. Questo sarà implementato con la chiave del ristorante come primary key, e la chiave del ristoratore come **unique**.



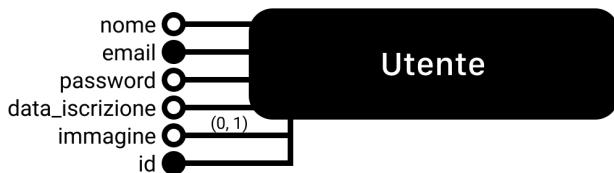
La scelta di questo modello è stata fatta per un'idea futura, dove più ristoratori potranno gestire un solo ristorante, e dove più ristoranti potranno essere gestiti da un preciso ristoratore. In futuro, quindi, lo schema diventerà come il seguente.



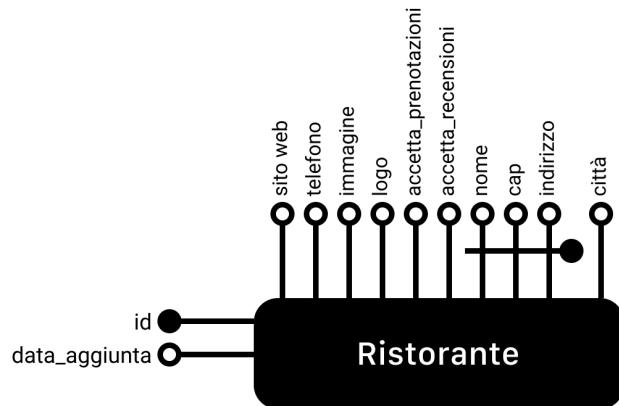
### 3. Traduzione di entità e associazioni in schemi di relazioni



```
Ristoratore(id, email, nome, password, data_iscrizione, immagine_profilo)  
AK: email
```

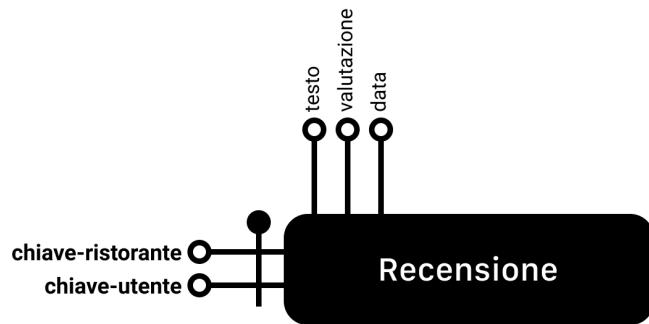


```
Utente(id, email, nome, password, data_iscrizione, immagine_profilo)  
AK: email
```



```

Ristorante(id, nome, cap, sito_web, telefono, cover, logo,
           accetta_prenotazioni, accetta_recensioni, indirizzo, città,
           data_aggiunta)
AK: nome, cap, indirizzo
    
```



```

Recensione(ristorante, utente, testo, valutazione, data)
FK: ristorante REFERENCES Ristorante
FK: utente REFERENCES Utente
    
```

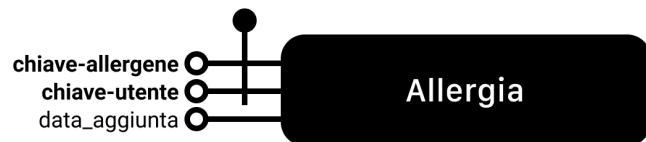


```
Allergene(id, nome, descrizione)
```

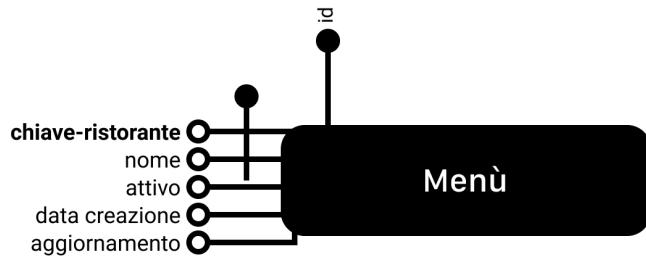


```
Portata(id, nome)
```

Come si è detto prima le portate in un secondo momento potranno essere personalizzate da ogni ristorante. Da come è strutturata questa entità, e tutte quelle ad essa collegate, sarà possibile senza troppo sforzi. L'aggiornamento della entità portata potrà risultare come segue (dove ristorante e nome potranno essere una buona **unique key**):



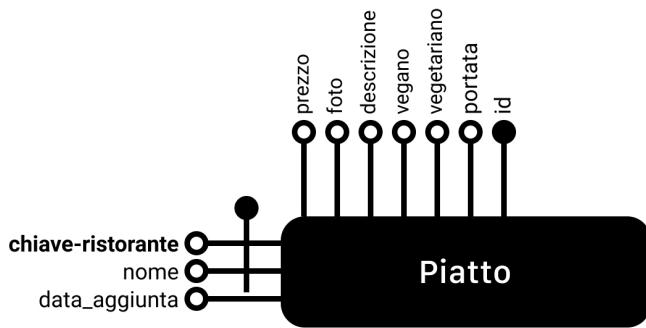
```
Allergia(allergene, utente, data_aggiunta)
FK: allergene REFERENCES Allergene
FK: utente REFERENCES Utente
```



```

Menù(id, ristorante, nome, attivo, data_creaione, ultimo_aggiornamento)
AK: ristorante, nome
FK: ristorante REFERENCES Ristorante

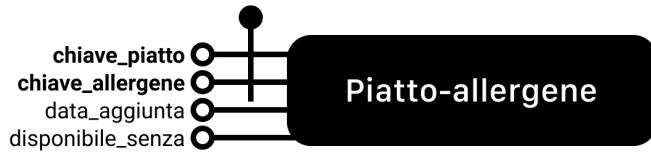
```



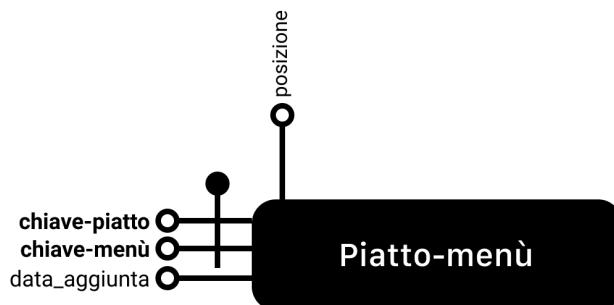
```

Piatto(id, ristorante, nome, prezzo, foto, descrizione, vegano, vegetariano,
       portata, data_aggiunta)
AK: ristorante, nome
FK: ristorante REFERENCES Ristorante
FK: portata REFERENCES Portata

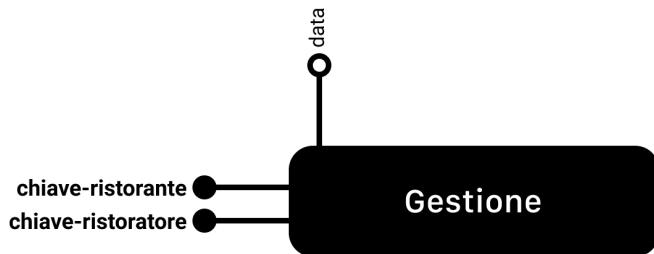
```



Piatto-allergene(piatto, allergene, disponibile\_senza, data\_aggiunta)  
 FK: piatto REFERENCES Piatto  
 FK: allergene REFERENCES Allergene



Piatto-menù(piatto, allergene, posizione, data\_aggiunta)  
 FK: piatto REFERENCES Piatto  
 FK: allergene REFERENCES Allergene



```

Gestione(ristorante, ristoratore, data_assegnazione)
FK: ristorante REFERENCES Ristorante
FK: ristoratore REFERENCES Ristoratore

```

## Operazioni SQL

Successivamente si riportano la creazione di tutte le entità, e un esempio di inserimento dati.

```

-- Creo la tabella per allergene
CREATE TABLE ALLERGENE (
    ID SMALLSERIAL PRIMARY KEY,
    NOME VARCHAR(60) NOT NULL,
    DESCRIZIONE TEXT NOT NULL
);

-- Inserisco gli allergeni nella tabella
INSERT INTO ALLERGENE VALUES (1, 'Glutine', 'Cereali, grano, segale, orzo, avena, farro, kamut, inclusi ibridati e derivati.');

```

```

-- Creo la tabella per portata
CREATE TABLE PORTATA (
    ID UUID PRIMARY KEY NOT NULL DEFAULT uuid_generate_v1(),
    NOME VARCHAR(60) NOT NULL,
    ORDINE SMALLINT NOT NULL DEFAULT 0
);

-- Inserisco le portate nella tabella

```

```
INSERT INTO PORTATA VALUES
('19CB3E0A-3ABD-4465-8FF6-6DAD85715EF4', 'Antipasti', 0);
```

```
-- Creo la tabella per gli utenti
CREATE TABLE UTENTE (
    ID UUID NOT NULL PRIMARY KEY DEFAULT uuid_generate_v1(),
    EMAIL TEXT NOT NULL,
    NOME VARCHAR(60) NOT NULL,
    PASSWORD TEXT NOT NULL,
    DATA_ISCRIZIONE TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
    IMMAGINE_PROFILE TEXT,
    CONSTRAINT utente_email_check CHECK (EMAIL ~ '^[a-zA-Z0-9._!#$%&'^*+/=?^`{|}~-]+@[a-zA-Z0-9](?:[a-zA-Z0-9-]{0,61}[a-zA-Z0-9])?(?:\.[a-zA-Z0-9](?:[a-zA-Z0-9-]{0,61}[a-zA-Z0-9])?)*$' )
);

-- Creo un indice per la mail, in modo da verificare le email in lowercase
CREATE UNIQUE INDEX USER_UNIQUE_LOWER_EMAIL_INDEX on UTENTE (lower(EMAIL));

-- Inserisco i record nella tabella
-- Password encoded in sha1
INSERT INTO UTENTE VALUES ('304635E6-DEE2-47F0-B41B-4B71BA3A0EC4', 'text@example.com',
'Giulio Arnoldi', '5baa61e4c9b93f3f0682250b6cf8331b7ee68fd8');
```

```
-- Creo la tabella per i ristoratori
CREATE TABLE RISTORATORE (
    ID UUID NOT NULL PRIMARY KEY DEFAULT uuid_generate_v1(),
    EMAIL TEXT NOT NULL,
    NOME VARCHAR(60) NOT NULL,
    PASSWORD TEXT NOT NULL,
    DATA_ISCRIZIONE TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
    IMMAGINE_PROFILE TEXT,
    CONSTRAINT ristoratore_email_check CHECK (EMAIL ~ '^[a-zA-Z0-9._!#$%&'^*+/=?^`{|}~-]+@[a-zA-Z0-9](?:[a-zA-Z0-9-]{0,61}[a-zA-Z0-9])?(?:\.[a-zA-Z0-9](?:[a-zA-Z0-9-]{0,61}[a-zA-Z0-9])?)*$' )
);

CREATE UNIQUE INDEX RISTORATORE_UNIQUE_LOWER_EMAIL_INDEX on RISTORATORE (lower(EMAIL));

-- Inserisco i record nella tabella
-- Password encoded in sha1
INSERT INTO RISTORATORE VALUES ('1442D1E5-240F-4088-B744-BF28C1B0D8CF', 'ale@typical.it',
'Alessandro di Giovanni', '5baa61e4c9b93f3f0682250b6cf8331b7ee68fd8');
```

Le password di **utente** e **ristoratore** sono salvate sotto forma di hash SHA1.

```
-- Installo uuid-ossp, per generare uuid come valore di default
CREATE EXTENSION IF NOT EXISTS "uuid-ossp";

-- Creo la tabella per ristorante
CREATE TABLE RISTORANTE (
    ID UUID PRIMARY KEY DEFAULT uuid_generate_v1(),
    NOME VARCHAR(60) NOT NULL,
    CAP VARCHAR(15) NOT NULL,
    SITO_WEB TEXT,
    TELEFONO TEXT,
    COVER TEXT,
    LOGO TEXT,
    ACCETTA_PRENOTAZIONI BOOL NOT NULL,
    ACCETTA_RECENSIONI BOOL NOT NULL DEFAULT TRUE,
    CITTA VARCHAR(100) NOT NULL,
    INDIRIZZO TEXT NOT NULL,
    DATA_AGGIUNTA TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
    CONSTRAINT ristorante_sito_web_check CHECK (SITO_WEB ~ 'https?:\/\/(www\.)?[-a-zA-Z0-9@:%._+~#=]{2,255}\.[a-z]{2,9}\y([-a-zA-Z0-9@:%._+~#=]*)$'),
    CONSTRAINT ristorante_telefono_check CHECK (TELEFONO ~ '^[\\+]?[\\(]?[0-9]{3}[\\)]?[-\\s\\.]?[0-9]{3}[-\\s\\.]?[0-9]{4,6}$'),
    CONSTRAINT ristorante_cap_check CHECK (CAP ~ '^\d{5}(?:[-\\s]\\d{4})?$'),
    CONSTRAINT UNIQUE_ADDRESS UNIQUE (CAP, INDIRIZZO, NOME)
);

-- Inserisco i record nella tabella
INSERT INTO RISTORANTE VALUES ('976A3F01-62A7-421D-88F6-FFD7CC79131F', 'Typical', '38122', 'https://typical.it', '04611975804', '', '', true, true, 'Trento', 'Via Calepina, 28');
```

```
-- Creo la tabella per recensione
CREATE TABLE RECENSIONE (
    RISTORANTE UUID NOT NULL REFERENCES RISTORANTE(ID) ON DELETE CASCADE,
    UTENTE UUID NOT NULL REFERENCES UTENTE(ID) ON DELETE CASCADE,
    TESTO TEXT NOT NULL,
    VALUTAZIONE NUMERIC NOT NULL,
    DATA TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
    CONSTRAINT recensione_valutazione_check CHECK (VALUTAZIONE <= 5.0 AND VALUTAZIONE > 0.0 AND ((VALUTAZIONE * 10) % 5 = 0)),
    PRIMARY KEY (RISTORANTE, UTENTE)
);

-- Inserisco i record nella tabella
INSERT INTO RECENSIONE VALUES
('976A3F01-62A7-421D-88F6-FFD7CC79131F', '304635E6-DEE2-47F0-B41B-4B71BA3A0EC4', 'abbi amo mangiato molto bene', 4.0);
```

Come si può vedere in questo caso, si sta utilizzando `ON DELETE CASCADE` con le foreign keys. Questo perché si vuole che quando viene eliminato ad esempio un ristorante, vengano eliminate anche tutte le sue recensioni dal database.

```
-- Creo la tabella per la relazione tra utente e allergene
CREATE TABLE ALLERGIA (
    UTENTE UUID NOT NULL REFERENCES UTENTE(ID) ON DELETE CASCADE,
    ALLERGENE INT NOT NULL REFERENCES ALLERGENE(ID),
    DATA_AGGIUNTA TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
    PRIMARY KEY (UTENTE, ALLERGENE)
);

-- Inserisco i record nella tabella
INSERT INTO ALLERGIA VALUES
('304635E6-DEE2-47F0-B41B-4B71BA3A0EC4', 3);
```

```
-- Creo la tabella per menu
CREATE TABLE MENU (
    ID UUID NOT NULL PRIMARY KEY DEFAULT uuid_generate_v1(),
    RISTORANTE UUID NOT NULL REFERENCES RISTORANTE(ID) ON DELETE CASCADE,
    NOME VARCHAR(100) NOT NULL,
    ATTIVO BOOL NOT NULL DEFAULT FALSE,
    DATA_CREAZIONE TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
    ULTIMO_AGGIORNAMENTO TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
    CONSTRAINT MENU_RISTORANTE_NOME UNIQUE(RISTORANTE, NOME)
);

-- Inserisco i record nella tabella
INSERT INTO MENU (ID, RISTORANTE, NOME, ATTIVO) VALUES ('EBA64392-2F4C-4FB4-842B-F22E4
02EEE68', '976A3F01-62A7-421D-88F6-FFD7CC79131F', 'inverno_2022', true),
```

```
-- Creo la tabella per piatto
CREATE TABLE PIATTO (
    ID UUID NOT NULL PRIMARY KEY DEFAULT uuid_generate_v1(),
    RISTORANTE UUID NOT NULL REFERENCES RISTORANTE(ID) ON DELETE CASCADE,
    NOME VARCHAR NOT NULL,
    PREZZO NUMERIC NOT NULL,
    FOTO TEXT,
    DESCRIZIONE TEXT,
    VEGANO BOOL NOT NULL,
    VEGETARIANO BOOL NOT NULL,
    PORTATA UUID NOT NULL REFERENCES PORTATA(ID) ON DELETE RESTRICT,
    DATA_AGGIUNTA TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
    CONSTRAINT PIATTO_RISTORANTE_NOME UNIQUE (RISTORANTE, NOME)
);

-- Inserisco i record nella tabella
INSERT INTO PIATTO VALUES ('2F30706F-5E79-47D0-9298-24798EB71336', '976A3F01-62A7-421D
-88F6-FFD7CC79131F', 'Jack e lo speck', 16.5, '', 'Gusto leggermente affumicato. Ricop
erte con salsa tradizionale Trentina, fatta da noi con base di mele e sfumata al Jack
Daniel. Poi guarnite con speck croccante, servite con le nostre patate della casa.', 
false, false, 3);
```

Nel caso di `portata`, invece, si può notare un `ON DELETE RESTRICT`. Questo previene l'eliminazione di una portata, se è ancora associata ad un piatto.

```
-- Creo la tabella per piatto
CREATE TABLE PIATTO_ALLERGENE (
    PIATTO UUID NOT NULL REFERENCES PIATTO(ID) ON DELETE CASCADE,
    ALLERGENE INT NOT NULL REFERENCES ALLERGENE(ID),
    DISPONIBILE_SENZA BOOL NOT NULL DEFAULT FALSE,
    DATA_AGGIUNTA TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
    PRIMARY KEY (PIATTO, ALLERGENE)
);

-- Inserisco i record nella tabella
INSERT INTO PIATTO_ALLERGENE VALUES('2F30706F-5E79-47D0-9298-24798EB71336', 1);
```

```
-- Creo la tabella per la relazione piatto-menu
CREATE TABLE PIATTO_MENU (
    MENU UUID NOT NULL REFERENCES MENU(ID) ON DELETE CASCADE,
    PIATTO UUID NOT NULL REFERENCES PIATTO(ID) ON DELETE CASCADE,
    POSIZIONE INT NOT NULL,
    DATA_AGGIUNTA TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
    PRIMARY KEY (MENU, PIATTO)
);

-- Inserisco i record nella tabella
INSERT INTO PIATTO_MENU VALUES('3cbd175f-1829-4526-be6c-8b1e3c6e8006', '2f30706f-5e79-47d0-9298-24798eb71336', 1);
```

```
-- Creo tabella per gestione
CREATE TABLE GESTIONE (
    RISTORANTE UUID NOT NULL REFERENCES RISTORANTE(ID) ON DELETE CASCADE PRIMARY KEY,
    RISTORATORE UUID NOT NULL REFERENCES RISTORATORE(ID) ON DELETE RESTRICT,
    DATA_ASSEGNAZIONE TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
    CONSTRAINT RISTORATORE_CON UNIQUE (RISTORATORE)
);

-- Inserisco i record nella tabella
INSERT INTO GESTIONE VALUES('976A3F01-62A7-421D-88F6-FFD7CC79131F', '1442D1E5-240F-4088-B744-BF28C1B0D8CF');
```

## Operazioni previste dalla base di dati – Descrizione e relativo codice SQL

Di seguito saranno elencate alcune delle operazioni più significative che permetteranno al sito web di interagire con la base di dati.

1. Lista di allergeni di un utente conoscendo il suo **id**.

```
SELECT
    AL.id,
    AL.nome,
    AL.descrizione
FROM
    ALLERGIA AS A,
    ALLERGENE AS AL
WHERE
    A.utente = '5A2E0F66-F568-4B58-A596-9CC10DBFD51D'
    AND A.allergene = AL.id;
```

2. Piatti di un menù che non contengono gli allergeni di un utente conosciuta la **chiave** dell'utente e quella del menù.

```
SELECT
    P.id,
    P.nome,
    P.prezzo,
    P.foto,
    P.descrizione,
    P.vegano,
    P.vegetariano,
    P.portata,
    ARRAY (
        SELECT
            a.nome
        FROM
            allergene AS a,
            piatto_allergene AS pa
        WHERE
            P.id = pa.piatto
            AND pa.allergene = a.id) AS nome_allergeni,
    ARRAY (
        SELECT
            a.id
        FROM
            allergene AS a,
            piatto_allergene AS pa
        WHERE
            P.id = pa.piatto
            AND pa.allergene = a.id) AS id_allergeni
FROM
    PIATTO_MENU AS PM,
```

```

PIATTO AS P
WHERE
PM.menu = 'eba64392-2f4c-4fb4-842b-f22e402eee68'
AND PM.piatt = P.id
AND P.id NOT IN(
    SELECT
        PA.piatt FROM PIATTO_ALLERGENE AS PA, ALLERGIA
    WHERE
        utente = '91c1e2cd-63da-4f8e-893b-c8bcb58a131b'
        AND ALLERGIA.allergene = PA.allergene);

```

3. Tutti i ristoranti in una `città`, con media delle recensioni, se esistono, NULL altrimenti.

```

SELECT
R.id,
R.NOME,
R.CAP,
R.SITO_WEB,
R.TELEFONO,
R.COVER,
R.LOGO,
R.ACCEPPA_PRENOTAZIONI,
R.ACCEPPA_RECENSIONI,
R.CITTA,
R.INDIRIZZO,
R.DATA_AGGIUNTA,
avg_valutazione
FROM
RISTORANTE AS R LEFT OUTER JOIN (SELECT ristorante, AVG(valutazione)::numeric(10,2)
AS avg_valutazione FROM recensione GROUP BY ristorante) REC ON R.id = REC.ristorante
WHERE
R.CITTA ILIKE 'Trento';

```

4. Menù attivo con i piatti conosciuto l'`id` del ristorante

```

SELECT
P.id,
M.id AS id_menu
P.nome,
P.prezzo,
P.foto,
P.descrizione,
P.vegano,
P.vegetariano,
P.portata,
PO.nome AS nome_portata
FROM

```

```

PIATTO AS P,
PORTATA AS PO,
MENU AS M,
PIATTO_MENU AS PM
WHERE
M.ristorante = '976a3f01-62a7-421d-88f6-ffd7cc79131f'
AND M.attivo = TRUE
AND PM.menu = M.id
AND PM.piattto = P.id
AND P.portata = PO.id;

```

## 5. Ristoranti con annessi menù di una specifica città

```

SELECT
R.id AS id_ristorante,
R.nome AS nome_ristorante,
M.id AS id_menu,
P.id AS id_piattto,
P.nome AS nome_piattto,
P.prezzo,
P.foto,
P.descrizione,
P.vegano,
P.vegetariano,
P.portata,
PO.nome AS nome_portata,
PO.ordine
FROM
RISTORANTE AS R,
MENU AS M,
PIATTO_MENU AS PM,
PIATTO AS P,
PORTATA AS PO
WHERE
R.citta ILIKE 'Trento'
AND R.id = M.ristorante
AND M.attivo = TRUE
AND M.id = PM.menu
AND PM.piattto = P.id
AND P.portata = PO.id
ORDER BY R.id ASC, PO.ordine ASC;

```

## 6. Recensioni di un ristorante conosciuto il suo id, con relativo nome dell'utente che l'ha scritta

```

SELECT
R.ristorante,
U.nome AS username,

```

```

R.testo,
R.valutazione,
R.data
FROM
    RECENSIONE AS R,
    UTENTE AS U
WHERE
    R.ristorante = '976a3f01-62a7-421d-88f6-ffd7cc79131f'
    AND R.utente = U.id;

```

## 7. Scrittura di una `recensione` da parte di un utente

```

INSERT INTO
    RECENSIONE
VALUES
    ('976A3F01-62A7-421D-88F6-FFD7CC79131F', '304635E6-DEE2-47F0-B41B-4B71BA3A0EC4', 'ab
    biamo mangiato molto bene', 4.0);

```

## 8. Aggiunta di un `piatto` da parte di un ristorante

```

INSERT INTO
    PIATTO (ID, RISTORANTE, NOME, PREZZO, FOTO, DESCRIZIONE, VEGANO, VEGETARIANO, PORTATA)
VALUES
    ('2F30706F-5E79-47D0-9298-24798EB71336', '976A3F01-62A7-421D-88F6-FFD7CC79131F', 'Ja
    ck e lo speck', 16.5, '', 'Costine dal gusto leggermente affumicato. Ricoperte con sal
    sa tradizionale Trentina, fatta da noi con base di mele e sfumata al Jack Daniel. Poi
    guarnite con speck croccante, servite con le nostre patate della casa.', false, fals
    e, '3E81CF7A-4FA1-4E6D-96A8-1C471CC78F8D');

```

## 8. Conteggio `ristoranti` in una città

```

SELECT COUNT(*)
    FROM RISTORANTE
WHERE
    citta ILIKE 'Trento';

```

## 9. Aggiunta di un ristorante e del suo gestore. Quando si inserisce un ristorante, bisogna immediatamente aggiungere anche la relazione tra il suo gestore

nell'entità `gestione`. Questo si può fare in un'unica transazione atomica. In questo modo, se per caso l'inserimento nella tabella `gestione` non dovesse avere successo, non viene inserito nemmeno il ristorante.

```
WITH NEW_RISTORANTE AS (
    INSERT INTO ristorante (NOME,
        CAP,
        ACCETTA_PRENOTAZIONI,
        CITTA,
        INDIRIZZO)
    VALUES ('Da Luigi',
        '70123',
        TRUE,
        'trento',
        'via bao, 22')
    RETURNING
        id
) INSERT INTO GESTIONE
SELECT
    id,
    'id_utente'
FROM
    NEW_RISTORANTE;
```

10. Aggiornare il nome di un piatto conoscendo il suo `id`.

```
UPDATE
    PIATTO
SET
    NOME = 'Non è una Tatin, non è una Sacher e à ghe anche l'amareina'
WHERE
    ID = '3a214a1d-6c2f-4f34-992b-e2ff297fac17';
```

11. Aggiornare il prezzo di un piatto dato il suo `id`.

```
UPDATE
    PIATTO
SET
    PREZZO = 75.0
WHERE
    ID = '3a214a1d-6c2f-4f34-992b-e2ff297fac17';
```

12. Rimuovere un piatto da un menu conoscendo `id` di entrambi. Dal momento che il piatto è legato al ristorante, e può essere utilizzato in altri menù, verrà eliminata solo la relazione tra il piatto e il menù.

```
DELETE FROM
PIATTO_MENU
WHERE
MENU='d17a7b54-d74e-4d29-815c-8ccc5134bab6'
AND PIATTO='3a214a1d-6c2f-4f34-992b-e2ff297fac17';
```

12. Rimuovere un ristorante e tutti i piatti, menù, recensioni e gestioni annesse. Il comando `DELETE` è CASCADE, dal momento che lo abbiamo settato durante la creazione della tabella, quindi basterà utilizzare:

```
DELETE FROM
RISTORANTE
WHERE
ID='b2ed7cf3-a59e-4e90-8572-3ef356bfb106';
```

## Triggers

Per fare in modo che ogni ristorante abbia un solo menù attivo alla volta, si implementa un trigger, che viene azionato prima di aggiornare un record, oppure prima di inserirne uno nuovo. In entrambi i casi, il trigger viene attivato solo se il nuovo valore di `attivo` è `TRUE`.

Per prima cosa si deve creare la funzione da chiamare quando si attiva il trigger.

```
CREATE OR REPLACE FUNCTION CREATE_TRIGGER_FOR_MENU_ACTIVITY()
RETURNS trigger
LANGUAGE plpgsql
AS $$
BEGIN
    IF OLD.ATTIVO IS DISTINCT FROM NEW.ATTIVO AND NEW.ATTIVO = TRUE
    THEN
        UPDATE MENU
        SET ATTIVO = FALSE
        WHERE RISTORANTE = NEW.RISTORANTE AND ID != NEW.ID;
```

```

END IF;

NEW.ULTIMO_AGGIORNAMENTO = CURRENT_TIMESTAMP;

RETURN NEW;
END;
$$;
```

Successivamente si possono creare i veri e propri trigger, che chiamano la funzione appena creata.

```

-- Creo il trigger per gli aggiornamenti della tabella
CREATE TRIGGER MENU_UPDATE_ACTIVITY
BEFORE UPDATE ON MENU
FOR EACH ROW
WHEN (OLD.ATTIVO IS DISTINCT FROM NEW.ATTIVO AND NEW.ATTIVO is TRUE)
EXECUTE PROCEDURE CREATE_TRIGGER_FOR_MENU_ACTIVITY();

-- Creo il trigger per gli inserimenti nella tabella
CREATE TRIGGER MENU_INSERT_ACTIVITY
BEFORE INSERT ON MENU
FOR EACH ROW
WHEN (NEW.ATTIVO is TRUE)
EXECUTE PROCEDURE CREATE_TRIGGER_FOR_MENU_ACTIVITY();
```

Un altro utile controllo è messo nella creazione e nell'aggiornamento dell'entità relazionale tra piatto e menu, dove controlliamo se il ristorante proprietario del menù e del piatto combacia. Come sopra prima si crea la funzione, che in questo caso usa anche due variabili, dove verranno inseriti l'id del ristorante proprietario del menù e quello del ristorante proprietario del piatto. Successivamente si controllano i token. Se non sono uguali, si alza un'eccezione e si ritorna `null`.

```

CREATE OR REPLACE FUNCTION CREATE_TRIGGER_FOR_PLATE_MENU_ACTIVITY()
RETURNS trigger
LANGUAGE plpgsql
AS $$

DECLARE
    menu_restaurant_id UUID;
    piatto_restaurant_id UUID;
BEGIN
    SELECT M.ristorante INTO menu_restaurant_id FROM MENU as M WHERE M.id = NEW.MENU LIMIT 1;
    SELECT P.ristorante INTO piatto_restaurant_id FROM PIATTO as P WHERE P.id = NEW.PIATO LIMIT 1;

    IF menu_restaurant_id != piatto_restaurant_id
    THEN
```

```

        RAISE EXCEPTION 'Menu e piatto utilizzati non appartengono allo stesso ristorant
e.'
        USING HINT = 'Per cortesia controllare gli id di piatto e menu.';
        RETURN NULL;
    END IF;
    RETURN NEW;
END;
$$;

```

Si creano anche ancora una volta i veri e propri trigger che implementano la funzione.

```

CREATE TRIGGER PLATE_MENU_INSERT_ACTIVITY
BEFORE INSERT ON PIATTO_MENU
FOR EACH ROW
EXECUTE PROCEDURE CREATE_TRIGGER_FOR_PLATE_MENU_ACTIVITY();

CREATE TRIGGER PLATE_MENU_UPDATE_ACTIVITY
BEFORE UPDATE ON PIATTO_MENU
FOR EACH ROW
WHEN (OLD.MENU IS DISTINCT FROM NEW.MENU OR OLD.PIATTO IS DISTINCT FROM NEW.PIATTO)
EXECUTE PROCEDURE CREATE_TRIGGER_FOR_PLATE_MENU_ACTIVITY();CREATE OR REPLACE FUNCTIO
N CREATE_TRIGGER_FOR_MENU_ACTIVITY()
RETURNS trigger
LANGUAGE plpgsql
AS $$

BEGIN
    IF OLD.ATTIVO IS DISTINCT FROM NEW.ATTIVO AND NEW.ATTIVO = TRUE
    THEN
        UPDATE MENU
            SET ATTIVO = FALSE
            WHERE RISTORANTE = NEW.RISTORANTE AND ID != NEW.ID;
    END IF;

    NEW.ULTIMO_AGGIORNAMENTO = CURRENT_TIMESTAMP;

    RETURN NEW;
END;
$$;

```

Un altro trigger è utilizzato per controllare le recensioni. In particolare, per controllare se la recensione inserita/aggiornata è scritta per un ristorante che accetta le recensioni.

```

CREATE OR REPLACE FUNCTION CREATE_TRIGGER_FOR REVIEW_ACTIVITY()
RETURNS trigger
LANGUAGE plpgsql
AS $$

DECLARE

```

```

restaurant_accepts_review BOOL;
BEGIN
  IF OLD.RISTORANTE IS DISTINCT FROM NEW.RISTORANTE
  THEN
    SELECT R.ACCEDE_ALLE_RECENSIONI INTO restaurant_accepts_review FROM RISTORANTE as R WHERE R.id = NEW.RISTORANTE LIMIT 1;
    IF restaurant_accepts_review = FALSE
    THEN
      RAISE EXCEPTION 'Il ristorante per cui si sta inserendo la recensione non accetta recensioni';
      USING HINT = 'Per cortesia controllare gli id del ristorante.';
      RETURN NULL;
    END IF;
  END IF;

  RETURN NEW;
END;
$$;

-- Creo il trigger per gli aggiornamenti della tabella
CREATE TRIGGER RECENSIONE_UPDATE_ACTIVITY
  BEFORE UPDATE ON RECENSIONE
  FOR EACH ROW
  EXECUTE PROCEDURE CREATE_TRIGGER_FOR_REVIEW_ACTIVITY();

-- Creo il trigger per gli inserimenti nella tabella
CREATE TRIGGER RECENSIONE_INSERT_ACTIVITY
  BEFORE INSERT ON RECENSIONE
  FOR EACH ROW
  EXECUTE PROCEDURE CREATE_TRIGGER_FOR_REVIEW_ACTIVITY();

```