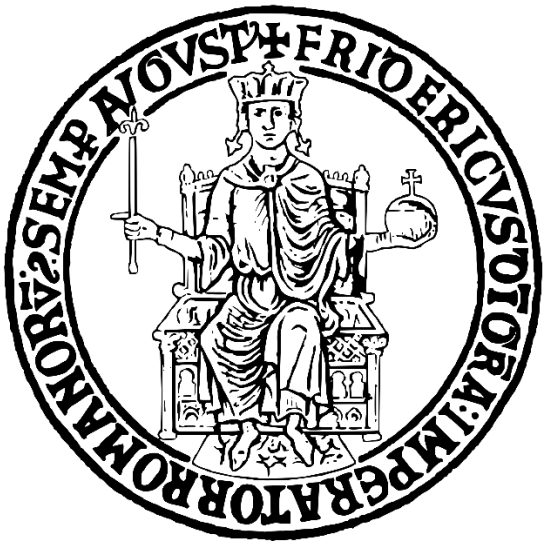


Progettazione e sviluppo di una Base di dati relazionale per la gestione di una catena di ristorazione

Fabio Cinicolo
N86003340

Umberto Elias De Angelis
N86003190

Mirco Napolitano
N86003332

Indice

Descrizione

La base di dati è stata pensata e realizzata per soddisfare le esigenze di diverse tipologie di utenti della catena di ristorazione, si è quindi deciso di categorizzarli in:

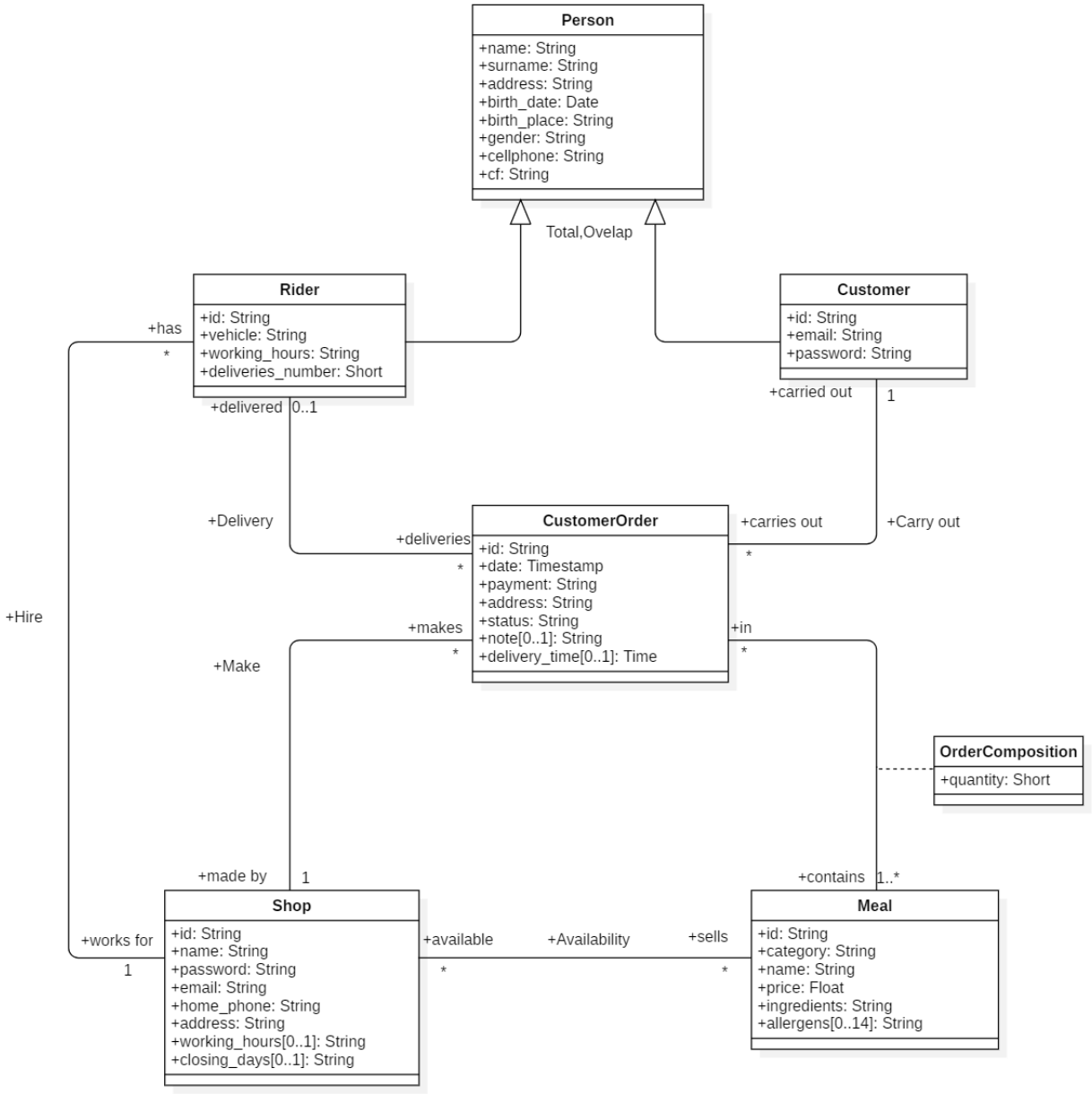
• Clienti : potranno registrarsi ed effettuare l'accesso all'applicativo. Potranno effettuare ordini verso la catena di ristorazione. Inoltre, sarà possibile effettuare una ricerca di alimenti combinando categoria, fascia di prezzo e allergie alimentari.

• Ristoranti : ristoratori che accederanno all'applicativo di gestione del proprio ristorante. Potranno gestire i rider (in particolare assumerne o licenziarne), inserire o rimuovere alimenti della catena di ristorazione nel menù e visualizzarli. Potranno, inoltre, assegnare una consegna ad un rider a patto che non abbia più di tre consegne in corso. Un ordine verrà considerato completato quando questo verrà consegnato al cliente.

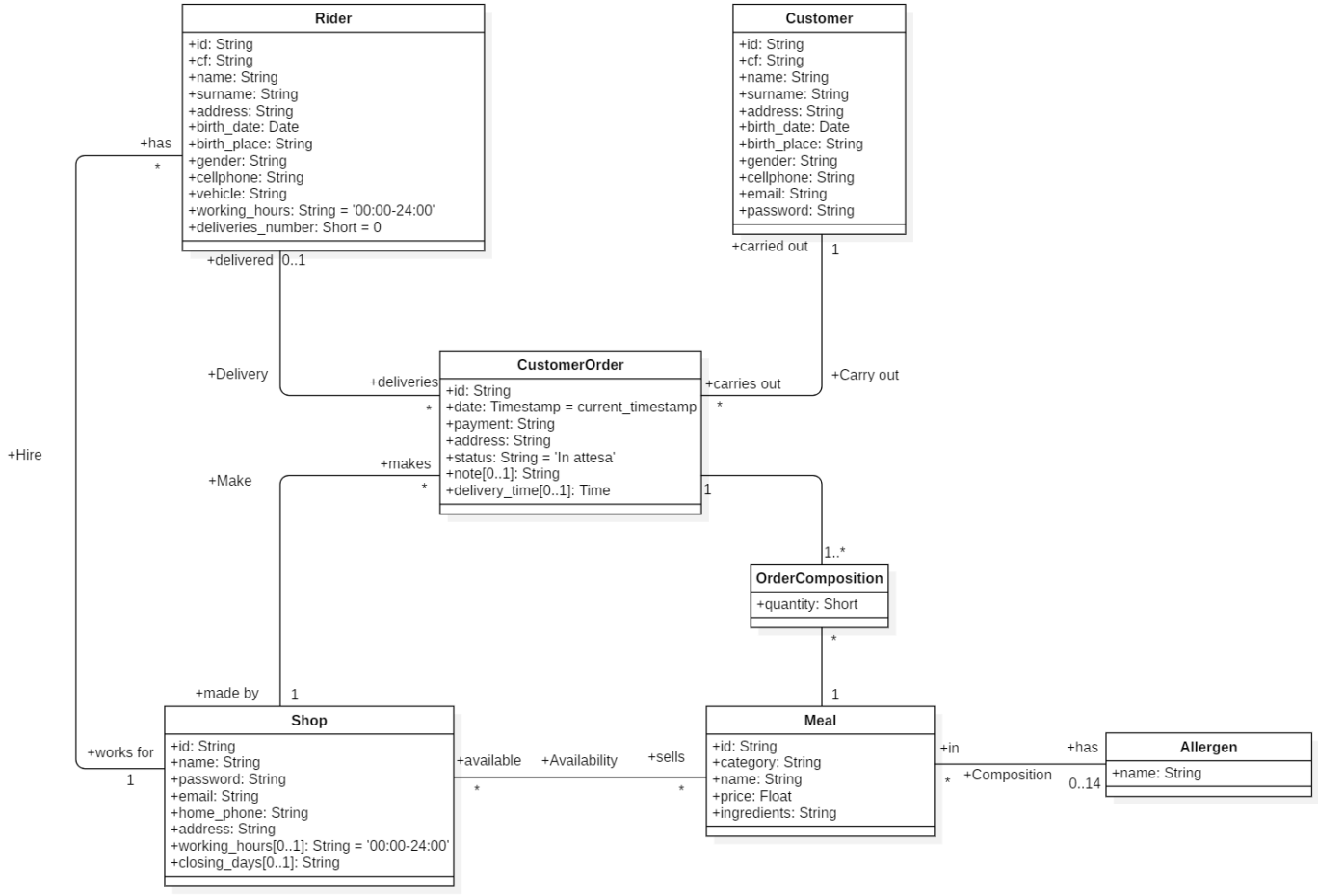
• Amministratore : effettuerà l'accesso al pannello di amministrazione della catena di ristorazione, potrà poi visualizzare e gestire i diversi ristoranti, inserire nuovi alimenti, visualizzare la clientela. Potrà inoltre effettuare una ricerca complessa di ordini combinando: veicolo del rider, fascia di prezzo del totale dell'ordine, categoria dei pasti e provincia di consegna. Il software, inoltre, permette al generazione del codice fiscale in fase di registrazione di un nuovo cliente (permettendo la modifica di questo in caso di omocodia dal pannello profilo del cliente) e in fase di assunzione di un rider.

E' possibile visionare un manuale per l'utilizzo del software tramite il seguente link:
Manuale per l'utilizzo del software Food Overflow

Class Diagram



Class Diagram Ristrutturato



Dizionario delle classi

Classe	Descrizione	Attributi
Customer	Contiene le informazioni personali di un cliente registrato nell'applicativo	**id** (string): codice identificativo univoco attribuito ad un cliente. **cf** (string): codice fiscale del cliente. **name** (string): nome del cliente. **surname** (string): cognome del cliente. **address** (string): indirizzo di consegna del cliente. **birth_date** (date): anno di nascita del cliente. **birth_place** (string): luogo di nascita del cliente. **gender** (string): sesso del cliente. **cellphone** (string): numero di cellulare del cliente. **email** (string): indirizzo e-mail del cliente. **password** (string): chiave d'accesso del cliente.
Rider	Contiene le informazioni personali e lavorative di un rider della catena di ristorazione	**cf** (string): codice fiscale del rider. **name** (string): nome del rider. **surname** (string): cognome del rider. **address** (string): indirizzo di residenza del rider. **birth_date** (date): anno di nascita del rider. **birth_place** (string): luogo di nascita del rider. **gender** (string): sesso del rider. **cellphone** (string): numero di cellulare del rider. **vehicle** (string): tipologia di veicolo utilizzato per le consegne. **working_hours** (string): Fascia oraria lavorativa del rider. **deliveries_number** (short): numero di consegne in corso (massimo 3).
Shop	Contiene le informazioni di un determinato ristorante della catena di ristorazione	**id** (string): codice identificativo univoco attribuito ad un ristorante. **name** (string): nome del ristorante. **email** (string): indirizzo e-mail del ristorante. **home_phone** (string): numero di telefono del ristorante. **address** (string): indirizzo del ristorante. **working_hours** (string): fascia oraria lavorativa del ristorante. **closing_days** (string): giorni di chiusura del ristorante.
Meal	Contiene le informazioni di un alimento della catena di ristorazione	**id** (string): codice identificativo univoco attribuito ad un alimento. **category** (string): tipologia dell'alimento (carne, pasta, etc.). **name** (string): nome dell'alimento. **price** (float): prezzo dell'alimento. **ingredients** (string): ingredienti di quell'alimento.
Allergen	Contiene i diversi allergeni	**name** (string): nome di un allergene.
CustomerOrder	Contiene le informazioni di diversi ordini eseguiti da diversi ristoranti della catena di ristorazione	**id** (string): codice identificativo univoco attribuito ad un ordine. **date** (timestamp): data di creazione ordine da parte di un cliente. **payment** (string): tipologia di pagamento selezionato da un cliente. **address** (string): indirizzo di consegna dell'ordine. **status** (string): stato dell'ordine (In attesa, In consegna, etc.). **note** (string): eventuali note lasciate da un cliente in fase di creazione ordine. **delivery_time** (time): orario in cui lo stato della consegna è diventato "Consegnato" o "Errore".
OrderComposition	Contiene la composizione di alimenti di diversi ordini	**quantity** (short): quantità dell'alimento dal cliente selezionata in fase di creazione ordine.

Dizionario delle associazioni

Nome	Descrizione	Classi coinvolte
Hire	Esprime il contratto tra un rider ed uno shop.	**Rider [*]** ruolo (**works for**): indica il Rider che lavora per lo Shop. **Shop [1]** ruolo (**has**): indica lo Shop che assume dei Rider.
Delivery	Esprime l'azione di consegna.	**Rider [0..1]** ruolo (**deliveries**): indica il Rider che effettua la consegna. **CustomerOrder [*]** ruolo (**delivered**): indica la consegna effettuata dal Rider.
Availability	Esprime la presenza o meno di un Meal nel menù di un negozio.	**Shop [*]** ruolo (**sells**): indica lo Shop che dispone degli alimenti. **Meal [*]** ruolo (**available**): indica il Meal disponibile per lo Shop.
Composition	Esprime gli allergeni di un alimento.	**Meal [*]** ruolo (**has**): indica il Meal che potrà avere allergeni. **Allergens [0..14]** ruolo (**in**): indica l'allergene contenuto in un alimento.
Make	Esprime la preparazione degli ordini da parte di un negozio.	**CustomerOrder [*]** ruolo (**made by**): indica l'ordine gestito da uno Shop. **Shop [1]** ruolo (**makes**): indica lo Shop che prepara l'ordine.
Carry out	Esprime un legame tra l'utente e l'ordine.	**Customer [1]** ruolo (**carried out**): indica l'utente che effettua l'ordine. **CustomerOrder [*]** ruolo (**carries out**): indica l'ordine effettuato da un utente.

Dizionario dei vincoli

Vincolo	Descrizione
customer_address_check	L'indirizzo del Customer deve avere il seguente formato: "<denominazione urbanistica>, <indirizzo>, <numero civico>, <CAP>, <Comune>, <Provincia>".
customer_order_address_check	L'indirizzo di consegna deve avere il seguente formato: "<denominazione urbanistica>, <indirizzo>, <numero civico>, <CAP>, <Comune>, <Provincia>".
rider_address_check	L'indirizzo di residenza del Rider deve avere il seguente formato: "<denominazione urbanistica>, <indirizzo>, <numero civico>, <CAP>, <Comune>, <Provincia>".
shop_address_check	L'indirizzo dello Shop deve avere il seguente formato: "<denominazione urbanistica>, <indirizzo>, <numero civico>, <CAP>, <Comune>, <Provincia>".
customer_email_check	L' email del customer deve essere scritta in modo corretto.
shop_email_check	L'email dello Shop deve avere per dominio : "foodoverflow.it".
customer_cf_check	Il codice fiscale del Customer deve essere scritto in modo legittimo.
rider_cf_check	Il codice fiscale del Rider deve essere scritto in modo legittimo.
shop_closing_days_check	I giorni di chiusura di uno shop devono essere scritti in sequenza, separati tra di loro dai caratteri virgola-spazio.
shop_working_hours_check	L'orario di apertura dello Shop deve essere scritto nel formato specificato (HH:MM-HH:MM).
rider_working_hours_check	L'orario lavorativo del Rider deve essere scritto nel formato specificato (HH:MM-HH:MM).
customer_cellphone_check	Il numero di cellulare del Customer deve essere scritto in modo legittimo, ovvero deve contenere dieci cifre.
rider_cellphone_check	Il numero di cellulare del Rider deve essere scritto in modo legittimo, ovvero deve contenere dieci cifre.
shop_home_phone_check	Il numero di telefono dello Shop deve essere scritto in modo legittimo, ovvero deve contenere un prefisso numerico nazionale.
rider_delieveries_number_check	Il numero di consegne di un Rider deve essere compreso tra zero e tre.
meal_price_check	Il prezzo di un Meal deve essere maggiore di zero.
order_composition_quantity_check	La quantità di un OrderComposition deve essere maggiore di zero
customer_order_status_check	Lo status di un ordine deve essere uno tra quelli specificati.
meal_category_check	La categoria di un Meal deve essere una tra quelle specificate.
allergen_name_check	Il nome di un allergene deve essere uno tra quelli specificati.
rider_gender_check	Il genere di un Rider deve essere uno tra quelli specificati.
customer_gender_check	Il genere di un Customer deve essere uno tra quelli specificati.

Schema Logico

Customer (id, cf, name, surname, address, birth_date, birth_place, gender, cellphone, email, password)

Rider (cf, name, surname, address, birth_date, birth_place, gender, cellphone, vehicle, working_hours, deliveries_number, shop_id)

shop_id→Shop.id

Shop (id, name, address, working_hours, closing_days, password, email, home_phone)

Meal (id, category, name, price, ingredients)

Allergen (name)

MealComposition (meal_id, allergen_name)

meal_id→Meal.id
allergen_name→Allergen.name

Supply (shop_id, meal_id)

shop_id→Shop.id
meal_id→Meal.id

CustomerOrder (id, date, delivery_time, address, status, payment, note, rider_cf shop_id, customer_id)

rider_cf→Rider.cf
shop_id→Shop.id
customer_id→Customer.id

OrderComposition (order_id, meal_id, quantity)

order_id→Order.id
meal_id→Meal.id

Definizione delle tabelle

```
-- Creazione sequenze
CREATE SEQUENCE shop_sequence;
CREATE SEQUENCE meal_sequence;
CREATE SEQUENCE customer_order_sequence;
CREATE SEQUENCE customer_sequence;

-- Creazione tabella "Customer"
CREATE TABLE customer (

    id CHAR(8) PRIMARY KEY DEFAULT
to_char(nextval('customer_sequence'),'00000000FM'),
    cf CHAR(16),
    name VARCHAR(50) NOT NULL,
    surname VARCHAR(50) NOT NULL,
    address VARCHAR(255) NOT NULL,
    birth_date DATE NOT NULL,
    birth_place VARCHAR(64) NOT NULL,
    gender CHAR(1) NOT NULL,
    cellphone CHAR(10) NOT NULL UNIQUE,
    email VARCHAR(320) NOT NULL UNIQUE,
    password VARCHAR(32) NOT NULL

);

-- Creazione tabella "Shop"
 CREATE TABLE shop (

    id CHAR(3) PRIMARY KEY DEFAULT to_char(nextval('shop_sequence'),'000FM'),
    name VARCHAR(50) NOT NULL,
    address VARCHAR(255) NOT NULL UNIQUE,
    working_hours CHAR(11) NOT NULL DEFAULT '00:00-24:00',
    closing_days VARCHAR(62),
    password VARCHAR(32) NOT NULL,
    email VARCHAR(320) NOT NULL UNIQUE,
    home_phone VARCHAR(12) NOT NULL UNIQUE

);

-- Creazione tabella "Rider"
CREATE TABLE rider (

    cf CHAR(16) PRIMARY KEY,
    name VARCHAR(50) NOT NULL,
    surname VARCHAR(50) NOT NULL,
    address VARCHAR(255) NOT NULL,
    birth_date DATE NOT NULL,
    birth_place VARCHAR(64) NOT NULL,
    gender CHAR(1) NOT NULL,
    cellphone CHAR(10) NOT NULL UNIQUE,

    vehicle VARCHAR(18) NOT NULL,
    working_hours CHAR(11) NOT NULL DEFAULT '00:00-24:00',
    deliveries_number SMALLINT NOT NULL DEFAULT 0,

    shop_id CHAR(3),
    FOREIGN KEY (shop_id) REFERENCES shop(id) ON DELETE CASCADE );
```

```
-- Creazione tabella "Meal"
CREATE TABLE meal (

    id CHAR (4) PRIMARY KEY DEFAULT to_char(nextval('meal_sequence'),'0000FM'),
    category VARCHAR(32) NOT NULL,
    name VARCHAR (50) NOT NULL UNIQUE,
    price REAL NOT NULL,
    ingredients VARCHAR(255)

);

-- Creazione tabella "CustomerOrder"
CREATE TABLE customerorder (

    id CHAR(12) PRIMARY KEY DEFAULT
to_char(nextval('customer_order_sequence'),'000000000000FM'),
    date TIMESTAMP NOT NULL DEFAULT current_timestamp,
    delivery_time TIME,
    address VARCHAR(255) NOT NULL,
    status VARCHAR(20) NOT NULL DEFAULT 'In attesa',
    payment VARCHAR(32) NOT NULL,
    note VARCHAR(255),

    rider_cf CHAR(16),
    shop_id CHAR(3),
    customer_id CHAR(8),

    FOREIGN KEY(rider_cf) REFERENCES Rider(cf) ON DELETE SET NULL ON UPDATE
CASCADE,
    FOREIGN KEY(shop_id) REFERENCES Shop(id) ON DELETE SET NULL,
    FOREIGN KEY(customer_id) REFERENCES Customer(id) ON DELETE SET NULL

);

-- Creazione tabella "Allergen"
CREATE TABLE allergen (

    name VARCHAR(32) PRIMARY KEY

);

-- Creazione tabella "OrderComposition"
CREATE TABLE ordercomposition (

    order_id CHAR(12),
    meal_id CHAR(4),
    quantity SMALLINT NOT NULL,

    PRIMARY KEY(order_id, meal_id),
    FOREIGN KEY (order_id) REFERENCES CustomerOrder(id) ON DELETE CASCADE,
    FOREIGN KEY (meal_id) REFERENCES Meal(id) ON DELETE SET NULL

);
```

```sql
-- Creazione tabella "MealComposition"
CREATE TABLE mealcomposition (

    meal_id CHAR(4),
    allergen_name VARCHAR(32),

    PRIMARY KEY(meal_id,allergen_name),
    FOREIGN KEY (meal_id) REFERENCES Meal(id) ON DELETE CASCADE,
    FOREIGN KEY (allergen_name) REFERENCES Allergen(name) ON DELETE CASCADE ON
UPDATE CASCADE

);

-- Creazione tabella "Supply"

CREATE TABLE Supply (

    shop_id char(3),
    meal_id char(4),
    PRIMARY KEY (shop_id,meal_id),
    FOREIGN KEY (shop_id) REFERENCES Shop(id) ON DELETE CASCADE,
    FOREIGN KEY (meal_id) REFERENCES Meal(id) ON DELETE CASCADE

);
```

Implementazione dei vincoli di CHECK

```
-- Controlla che gli indirizzi abbiano il seguente formato: "<denominazione
urbanistica>, <indirizzo>, <numero civico>, <CAP>, <Comune>, <Provincia>". Non
sono case sensitive.
ALTER TABLE Customer
ADD CONSTRAINT customer_address_check CHECK
(address~*'^(accesso|arco|belvedere|borgo|campo|canale|cascina|cavone|cavalcavia
|contrada|corso|cortile|costa|discesa|fondo|galleria|frazione|isola|lido|litoran
ea|lungo|lungomare|masseria|molo|mura|passaggio|passo|pendio|piazza|piazzale|pia
zzetta|ponte|portico|porto|rampa|recinto|rione|riva|rotonda|salita|scalinata|sce
sa|sentiero|spiaggia|spiazzo|strada|stradale|stretto|stretta|strettoia|terrazza|
traversa|via|vicoletto|vico|villaggio|viuzza)( )[ A-Za-z0-9]+(, )[ A-Z0-9]+(,
)[0-9]{5}(, )[\x20-\xA5]+(, )[\x20-\xFF]+$' );

ALTER TABLE CustomerOrder
ADD CONSTRAINT customer_order_address_check CHECK
(address~*'^(accesso|arco|belvedere|borgo|campo|canale|cascina|cavone|cavalcavia
|contrada|corso|cortile|costa|discesa|fondo|galleria|frazione|isola|lido|litoran
ea|lungo|lungomare|masseria|molo|mura|passaggio|passo|pendio|piazza|piazzale|pia
zzetta|ponte|portico|porto|rampa|recinto|rione|riva|rotonda|salita|scalinata|sce
sa|sentiero|spiaggia|spiazzo|strada|stradale|stretto|stretta|strettoia|terrazza|
traversa|via|vicoletto|vico|villaggio|viuzza)( )[ A-Za-z0-9]+(, )[ A-Z0-9]+(,
)[0-9]{5}(, )[\x20-\xA5]+(, )[\x20-\xFF]+$' );

ALTER TABLE Rider
ADD CONSTRAINT rider_address_check CHECK
(address~*'^(accesso|arco|belvedere|borgo|campo|canale|cascina|cavone|cavalcavia
|contrada|corso|cortile|costa|discesa|fondo|galleria|frazione|isola|lido|litoran
ea|lungo|lungomare|masseria|molo|mura|passaggio|passo|pendio|piazza|piazzale|pia
zzetta|ponte|portico|porto|rampa|recinto|rione|riva|rotonda|salita|scalinata|sce
sa|sentiero|spiaggia|spiazzo|strada|stradale|stretto|stretta|strettoia|terrazza|
traversa|via|vicoletto|vico|villaggio|viuzza)( )[ A-Za-z0-9]+(, )[ A-Z0-9]+(,
)[0-9]{5}(, )[\x20-\xA5]+(, )[\x20-\xFF]+$' );

ALTER TABLE Shop
ADD CONSTRAINT shop_address_check CHECK
(address~*'^(accesso|arco|belvedere|borgo|campo|canale|cascina|cavone|cavalcavia
|contrada|corso|cortile|costa|discesa|fondo|galleria|frazione|isola|lido|litoran
ea|lungo|lungomare|masseria|molo|mura|passaggio|passo|pendio|piazza|piazzale|pia
zzetta|ponte|portico|porto|rampa|recinto|rione|riva|rotonda|salita|scalinata|sce
sa|sentiero|spiaggia|spiazzo|strada|stradale|stretto|stretta|strettoia|terrazza|
traversa|via|vicoletto|vico|villaggio|viuzza)( )[ A-Za-z0-9]+(, )[ A-Z0-9]+(,
)[0-9]{5}(, )[\x20-\xA5]+(, )[\x20-\xFF]+$' );

-- Controlla che l' email sia scritta correttamente.
ALTER TABLE Customer
ADD CONSTRAINT customer_email_check CHECK (email ~* '^[A-Za-z0-9._%-]+(@)[A-Za-
z0-9.-]+[.][A-Za-z]+$');

-- Controlla che l' email abbia dominio "foodoverflow.it".
ALTER TABLE Shop
ADD CONSTRAINT shop_email_check CHECK (email ~ '^[A-Za-z0-9._%-
]+(@foodoverflow.it)$');
```

```
-- Controlla che il codice fiscale sia scritto correttamente.
ALTER TABLE Customer
ADD CONSTRAINT customer_cf_check CHECK (cf ~ '^[a-zA-Z]{6}[0-
9]{2}[abcdehlmprstABCDEHLMPRST]{1}[0-9]{2}([a-zA-Z]{1}[0-9]{3})[a-zA-Z]{1}$');

-- Controlla che il codice fiscale sia scritto correttamente.
ALTER TABLE Rider
ADD CONSTRAINT rider_cf_check CHECK (cf ~ '^[a-zA-Z]{6}[0-
9]{2}[abcdehlmprstABCDEHLMPRST]{1}[0-9]{2}([a-zA-Z]{1}[0-9]{3})[a-zA-Z]{1}$');

-- Controlla che l' attributo closing_days sia una stringa formata da giorni
della settimana separati da virgola-spazio, senza ripetizioni. Non è case
sensitive.
ALTER TABLE Shop
ADD CONSTRAINT shop_closing_days_check CHECK (closing_days ~*
'^((lunedì|lunedi|martedì|martedi|mercoledì|mercoledi|giovedì|giovedi|venerdì|ve
nerdi|sabato|domenica)(, (?!$)|$))*$');

-- Controllano che la fasce orarie lavorative siano del formato (HH:MM-HH:MM) .
ALTER TABLE Shop
ADD CONSTRAINT shop_working_hours_check CHECK (working_hours ~* '^([0-2]{1}[0-
9]{1})(:)([0-5]{1}[0-9]{1})(-)([0-2]{1}[0-9]{1})(:)([0-5]{1}[0-9]{1})$');

ALTER TABLE Rider
ADD CONSTRAINT rider_working_hours_check CHECK (working_hours ~* '^([0-2]{1}[0-
9]{1})(:)([0-5]{1}[0-9]{1})(-)([0-2]{1}[0-9]{1})(:)([0-5]{1}[0-9]{1})$');

-- Controllano che il numero di cellulare abbia 10 cifre numeriche.
ALTER TABLE Customer
ADD CONSTRAINT customer_cellphone_check CHECK (cellphone ~ '^[0-9]{10}$');

ALTER TABLE Rider
ADD CONSTRAINT rider_cellphone_check CHECK (cellphone ~ '^[0-9]{10}$');


-- Controlla che il prefisso sia nazionale, seguito da un trattino e da 7 cifre
numeriche.
ALTER TABLE Shop
ADD CONSTRAINT shop_home_phone_check CHECK (home_phone ~
'^(004191|010|011|0121|0122|0123|0124|0125|0131|0141|0142|0143|0144|015|0161|016
3|0165|0166|0171|0172|0173|0174|0175|0182|0183|0184|0185|0187|019|02|030|031|032
1|0322|0323|0324|0331|0332|0341|0342|0343|
0344|0345|0346|035|0362|0363|0364|0365|0371|0372|0373|0374|0375|0376|0377|0381|0
382|0383|0384|0385|0386|039|040|041|0421|0422|0423|0424|0425|0426|0427|0428|0429
|0431|0432|0433|0434|0435|0436|0437|0438|
0439|0442|0444|0445|045|0461|0462|0463|0464|0465|0471|0472|0473|0474|0481|049|05
0|051|0521|0522|0523|0524|0525|0532|0533|0534|0535|0536|0541|0542|0543|0544|0545
|0546|0547|0549|055|0564|0565|0566|0571|
0572|0573|0574|0575|0577|0578|0583|0584|0585|0586|0587|0588|059|06|0623|070|071|
0721|0722|0731|0732|0733|0734|0735|0736|0737|0742|0743|0744|0746|075|0761|0763|0
765|0766|0771|0773|0774|0775|0776|0781|
0782|0783|0784|0785|0789|079|080|081|0823|0824|0825|0827|0828|0831|0832|0833|083
5|0836|085|0861|0862|0863|0864|0865|0871|0872|0873|0874|0875|0881|0882|0883|0884
|0885|089|090|091|0921|0922|0923|
0924|0925|0931|0932|0933|0934|0935|0941|0942|095|0961|0962|0963|0964|0965|0966|0
967|0968|0971|0972|0973|0974|0975|0976|0981|0982|0983|0984|0985|099)(-)[0-
9]{7}$');

-- Controlla che un rider sia associato al più a 3 consegne.
ALTER TABLE Rider
ADD CONSTRAINT rider_delieveries_number_check CHECK (deliveries_number >= 0 AND
deliveries_number<=3);
```

```
-- Controlla che il prezzo di un meal sia maggiore di 0.
ALTER TABLE Meal
ADD CONSTRAINT meal_price_check CHECK (price > 0);

-- Controlla che la quantità di un meal nel carrello sia maggiore di 0.
ALTER TABLE OrderComposition
ADD CONSTRAINT order_composition_quantity_check CHECK (quantity > 0);

-- Controlla che lo stato dell'ordine sia uno tra quelli riportati di sotto
nell' espressione regolare. Non è case sensitive.
ALTER TABLE CustomerOrder
ADD CONSTRAINT customer_order_status_check CHECK (status ~* '^(In attesa|In
consegna|Consegnato|Errore)$');

-- Controlla che la categoria di un meal sia una tra quelle riportate di sotto
nell' espressione regolare. Non è case sensitive.
ALTER TABLE Meal
ADD CONSTRAINT meal_category_check CHECK (category ~* '^(Primo
piatto|Carne|Pesce|Pizza|Panino|Fritto|Dolce|Bevanda analcolica|Bevanda
alcolica)$');

-- Controlla che l' allergene sia uno tra quelli riportati di sotto
nell'espressione regolare. Non è case sensitive.
ALTER TABLE Allergen
ADD CONSTRAINT allergen_name_check CHECK (name ~* '^(Cereali e
derivati|Crostacei|Uova|Pesce|Arachidi|Soia|Latte e derivati|Frutta a
guscio|Sedano|Senape|Sesamo|An. solforosa e solfiti|Lupini|Molluschi)$');

-- Controllano che il genere sia "M" oppure "F". Non è case sensitive.
ALTER TABLE Rider
ADD CONSTRAINT rider_gender_check CHECK (gender ~* '^(M|F)$');

ALTER TABLE Customer
ADD CONSTRAINT customer_gender_check CHECK (gender ~* '^(M|F)$');
```

Implementazione delle procedure e dei trigger

```
-- Permette di effettuare una ricerca filtrata di pasti combinando categoria di
un pasto, fascia di prezzo; sono scartati i cibi che hanno almeno un allergia
alimentare tra quelle incluse nella lista.
CREATE OR REPLACE FUNCTION effettuaRicercaComplessaCustomer(category varchar,
min_price FLOAT, max_price FLOAT, allergen_list varchar, shop_email varchar)
RETURNS SETOF RECORD AS $$
DECLARE
command text;
i integer DEFAULT 1;
BEGIN
-- Nel caso in cui non voglio limiti sulla categoria
IF $1='Visualizza tutti i pasti' THEN command = 'SELECT DISTINCT name, category,
price, ingredients, id FROM Meal WHERE price >= '||$2||'AND price <= '||$3||'AND
id IN(SELECT meal_id FROM Supply WHERE shop_id=(SELECT id FROM Shop WHERE
email='||quote_literal($5)||')) AND id NOT IN (SELECT meal_id FROM
MealComposition WHERE allergen_name =';
ELSE
-- Ricerca applicando tutti i filtri
command = 'SELECT DISTINCT name, category, price, ingredients, id FROM Meal
WHERE category ='||quote_literal($1)||' AND price >= '||$2||'AND price <=
'||$3||'AND id IN(SELECT meal_id FROM Supply WHERE shop_id=(SELECT id FROM Shop
WHERE email='||quote_literal($5)||')) AND id NOT IN (SELECT meal_id FROM
MealComposition WHERE allergen_name =';
END IF;
-- Se la lista degli allergeni è NULL oppure è vuota non vengono presi in
considerazione le allergie alimentari
IF allergen_list IS NULL OR allergen_list='' THEN command = command||
quote_literal(' ')||') ORDER BY price';
END IF;
-- Finquando ci sono allergeni nella lista o la lista è NULL
LOOP
EXIT WHEN allergen_list='' OR allergen_list IS NULL;
IF SPLIT_PART(allergen_list,', ',i)='' THEN
 command = command ||quote_literal(allergen_list)||') ORDER BY price';
 allergen_list='';
ELSE
-- Concatena le diverse allergie alimentari
 command = command || quote_literal(SPLIT_PART(allergen_list,', ',i))|| ' OR
allergen_name=';
 i = i+1;
END IF;
END LOOP;
RETURN QUERY EXECUTE command;
END;
$$ LANGUAGE plpgsql;
```

```
-- Ricerca di ordini che hanno almeno un pasto della categoria selezionata, con
totale dell ordine compreso nella fascia di prezzo selezionata il cui rider ha
il veicolo selezionato e la cui provincia di consegna è quella selezionata
CREATE OR REPLACE FUNCTION effettuaRicercaComplessaAdmin(cat varchar, min_price
FLOAT, max_price FLOAT, vehc varchar, prov varchar) RETURNS SETOF RECORD AS $$
DECLARE
-- Le variabili "ok1" "ok2" permettono ad una condizione di essere sempre
verificata, quando richiesto.
ok1 int default 0;
ok2 int default 0;
no_rider varchar(200) default ' AND CO.rider_cf IN (SELECT cf FROM Rider WHERE
vehicle = '||quote_literal(vehc)||') GROUP BY CO.id HAVING
SUM(M.price*OC.quantity)>='||min_price||' AND
SUM(M.price*OC.quantity)<='||max_price||') ORDER BY CO.date';
command varchar(1000) default '';
BEGIN
IF cat = 'Seleziona categoria' OR cat = '------------------' THEN ok1=1; END
IF;
IF vehc = 'Seleziona veicolo del rider' OR vehc = '------------------' THEN
no_rider = ' GROUP BY CO.id HAVING SUM(M.price*OC.quantity)>='||min_price||' AND
SUM(M.price*OC.quantity)<='||max_price||') ORDER BY CO.date'; END IF;
IF prov ='Seleziona provincia di consegna' OR prov = '------------------' THEN
ok2=1; END IF;
command = 'SELECT  CO.id, CO.date, CO.delivery_time, CO.address, CO.status,
CO.payment, CO.note, CO.rider_cf, S.email, C.email
                                        FROM CustomerOrder AS CO JOIN Shop AS
S ON S.id = CO.shop_id JOIN Customer AS C ON C.id = CO.customer_id
                                        WHERE CO.id IN (SELECT CO.id
                                                FROM CustomerOrder CO
JOIN OrderComposition OC ON CO.id = OC.order_id JOIN Customer AS C ON C.id =
CO.customer_id JOIN Meal AS M ON M.id = OC.meal_id
                                                WHERE (M.category =
'||quote_literal(cat)||' OR 1='||ok1||') AND (SPLIT_PART(C.address,'','',5) ='||
quote_literal(prov)||' OR 1='||ok2||') ';
command = command || no_rider;
RETURN QUERY EXECUTE command;
END;
$$ LANGUAGE plpgsql;
```

```
-- Permette il salvataggio di un ordine sul database.
-- Verrà prima inserito un record in CustomerOrder e successivamente per ogni
pasto dell' ordine viene inserito un record in OrderComposition contenente l' id
dell' ordine, id del pasto con rispettiva quantità.
CREATE OR REPLACE PROCEDURE createOrder(addr varchar, payment varchar, notes
varchar, shop_email varchar, customer_email varchar, meal_list_name varchar,
quantity_list varchar)
-- meal_list_name è la lista dei nomi dei cibi ordinati dall' utente separati da
", "
-- quantity_list è la lista delle quantità dei rispettivi cibi separati da ", "
LANGUAGE PLPGSQL AS $$
DECLARE
meal_name Meal.name%TYPE;
quant varchar;
meal_counter int default 1;
id_meal Meal.id%TYPE;
id_shop Shop.id%TYPE;
id_customer Customer.id%TYPE;
BEGIN
IF notes='' THEN notes=null; END IF;
SELECT id INTO id_shop FROM Shop WHERE email=shop_email;
SELECT id INTO id_customer FROM Customer WHERE email=customer_email;
INSERT INTO CustomerOrder VALUES (DEFAULT, DEFAULT, null, addr, DEFAULT,
payment, notes, null, id_shop, id_customer);
LOOP -- Finquando c'è un nome di un alimento in meal_list_name allora effettua
l'inserimento in OrderComposition
 meal_name=split_part(meal_list_name, ', ', meal_counter);
 quant=split_part(quantity_list, ', ', meal_counter);
 SELECT id INTO id_meal FROM Meal WHERE name=meal_name;
 IF meal_name<>'' THEN
 INSERT INTO OrderComposition
VALUES(to_char(currval('customer_order_sequence'),'000000000000FM'),id_meal,quan
t::real);
 meal_counter=meal_counter+1;
 ELSE
 exit;
 END IF;
END LOOP;
END;
$$;

-- Permette di aggiornare i campi di un ristorante.
CREATE OR REPLACE PROCEDURE updateShop(shop_name varchar, addr varchar, hours
varchar, days varchar, passw varchar,newEmail varchar, phone varchar, oldEmail
varchar)
AS $$
BEGIN
IF hours=''THEN hours=DEFAULT; END IF;
IF days=''THEN days=null; END IF;
UPDATE Shop SET name=shop_name, working_hours=hours, closing_days=closing_days,
password=passw, email=newEmail, home_phone=phone, address=addr
WHERE email=oldEmail;
END;
$$
LANGUAGE PLPGSQL;
```

```
-- Permette di aggiungere ad un alimento una lista di allergeni
CREATE OR REPLACE PROCEDURE addAllergens(meal_name varchar, allergens varchar)
LANGUAGE plpgsql AS $$
DECLARE
allerg Allergen.name%TYPE DEFAULT '';
count_allergens INT DEFAULT 1;
meal_id Meal.id%TYPE;
BEGIN
SELECT id INTO meal_id FROM meal WHERE name=meal_name;
LOOP
 allerg=split_part(allergens, ', ', count_allergens);
 IF allerg<>'' THEN
 INSERT INTO MealComposition VALUES(meal_id, allerg);
 count_allergens = count_allergens + 1;
 ELSE
 exit;
 END IF;
END LOOP;
END;
$$;

-- Restituisce gli allergeni di un pasto.
CREATE OR REPLACE FUNCTION getAllergensOfAMeal(meal varchar) RETURNS VARCHAR
language plpgsql AS $$
DECLARE
my_curs cursor FOR SELECT allergen_name FROM MealComposition WHERE meal_id=meal;
allergens Allergen.name%TYPE DEFAULT '';
BEGIN
FOR I IN my_curs
LOOP
allergens = i.allergen_name||', '||allergens;
END LOOP;
return substr(allergens,1,length(allergens)-2);
END;
$$;

-- Permette di associare un rider ad una consegna. Se si prova ad associare un
ordine ad un rider che ha 3 consegne in corso, allora verrà lanciata un
eccezione.
CREATE OR REPLACE FUNCTION linkRiderToOrder() RETURNS TRIGGER AS
$link_rider_to_order$
BEGIN

        IF (SELECT deliveries_number FROM rider WHERE cf=new.rider_cf) < 3
        THEN
            UPDATE Rider
            SET deliveries_number=deliveries_number+1
            WHERE cf=new.rider_cf;
            Update CustomerOrder
            SET status = 'In consegna' WHERE id = new.id;
        else
        RAISE EXCEPTION 'Rider cannot be associated to more than 3 activities!';
        END IF;
    return new;

END;
$link_rider_to_order$ LANGUAGE plpgsql;
```

```
-- Trigger che viene innescato non appena un rider viene associato ad una
consegna.
CREATE TRIGGER link_rider_to_order
AFTER UPDATE of rider_cf ON customerorder
FOR EACH ROW
WHEN (OLD.rider_cf IS null)
EXECUTE PROCEDURE linkRiderToOrder();


-- Permette di aggiornare il numero di consegne di un rider nel caso in cui
questo ha consegnato l' ordine oppure ha avuto problemi durante la consegna.
-- Se si prova ad aggiornare il numero di consegne di un rider che non ha
consegne verrà lanciata un eccezione.
CREATE OR REPLACE FUNCTION updateDeliveriesNumberOfARider() RETURNS TRIGGER AS
$update_deliveries_number_of_a_rider$
BEGIN

    IF (SELECT deliveries_number FROM Rider WHERE cf = OLD.rider_cf )>0 THEN
        UPDATE rider
        SET deliveries_number=deliveries_number-1
        WHERE cf = OLD.rider_cf;
    ELSE
        RAISE EXCEPTION 'Rider with cf: % has no pending
deliveries',OLD.rider_cf;
        END IF;
    RETURN NEW;

END;
$update_deliveries_number_of_a_rider$ LANGUAGE plpgsql;

-- Trigger che viene innescato quando un ordine viene consegnato(cioè quando il
delivery_time viene aggiornato) oppure quando c'è stato un problema durante la
consegna di questo.
CREATE TRIGGER update_deliveries_number_of_a_rider
AFTER UPDATE of delivery_time ON customerorder
FOR EACH ROW
WHEN (OLD.delivery_time IS NULL)
EXECUTE PROCEDURE updateDeliveriesNumberOfARider();
```