Progetto Esame
BASI di DATI

di

Daniele Cristoni matricola n° 88460
Giacomo Guerzoni matricola n° 92469

GESTIONE DI UN FAST FOOD CON PIU` FILIALI

Indice:

1) Testo:

Stabilimenti:

ogni filiale ha un certo numero di dipendenti ognuno con uno specifico roulo(cassiere,cuoco,responsabile,inservienti,amministratore), ogni stabilimento possiede determinati attributi come numero di piastre,forni,casse...
lo stabilimento è identificato da un nome univoco e da nome della città e l'indirizzo.

Dipendenti:

I dipendenti sono descritti dagli usuali dati anagrafici, un numero di matricola e da un ruolo.
I turni sono settimanali e devono gestire il numero di dipendenti in modo che il numero del personale di uno specifico ruolo non ecceda il numero di postazioni presenti nella filiale di tale ruolo.
Ogni lavoratore può avere inoltre un suo diretto superiore.

Gestione Menu:

Il menu comprende una lista di pietanze,identificate dal nome e caratterizzate dalle kcal (dato derivato dalle singole kcal degli ingredienti) e dal prezzo, suddivisi principalmente in cibi e bevande.
I **cibi** sono composti da una serie di ingredienti**.**
Gli **ingredienti** hanno un nome univoco, kcal e una data di scadenza che viene comunicata dai fornitori.

Fornitura:

Il **fornitore** fornisce a uno stabilimento, in una certa data, determinati ingredienti.
Nella fornitura vengono specificati la data di scadenza dell'ingrediente, la quantità e il prezzo. Del fornitore interessa la partita iva e la lista di alimenti che possiede.

Clienti:

I clienti si suddividono in clienti occasionali e clienti fedeli, ai clienti fedeli viene associata una carta fedeltà il cui numero è univoco e sono caratterizzati da normali dati anagrafici.
I clienti effettuano **ordini** sui vari alimenti che sono presenti nel menu del rispettivo stabilimento. Ogni ordine è identificato da un codice incrementale giornaliero per cui viene memorizzata la data.
In un ordine può essere presente piu volte lo stesso alimento.

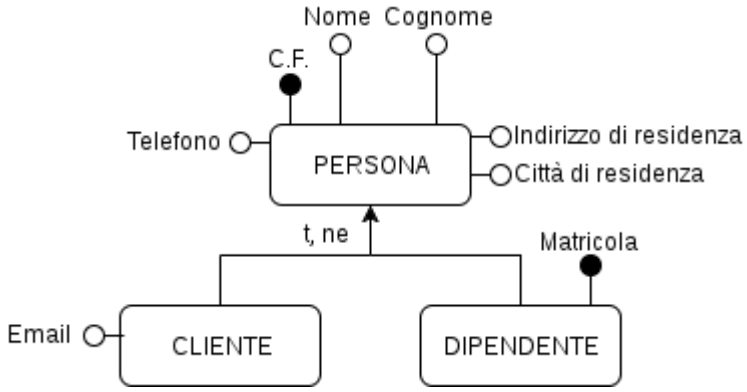
2) Glossario dei termini

Termine	Descrizione	Sinonimi	Legame
Stabilimento	Nome stabilimento, città, indirizzo, n° civico, n° di piastre, n° forni, n° casse, n° bagni	Filiale	Dipendente, Fornitore, Alimenti
Dipendente	C.F., nome, cognome, indirizzo residenza, data di nascita, matricola, ruolo	Personale	Stabilimento
Alimenti	Nome, kcal, prezzo	Cibi, bevande, prodotti	Ingrediente, Cliente, Stabilimento
Ingrediente	Nome, kcal	Prodotti	Alimenti, Fornitore
Fornitore	P.IVA, nome ditta, indirizzo, n° civico		Stabilimento, Ingrediente, Bevanda
Cliente	C.F., nome, cognome, indirizzo residenza, data di nascita, carta fedeltà	Socio, acquirente	Alimenti

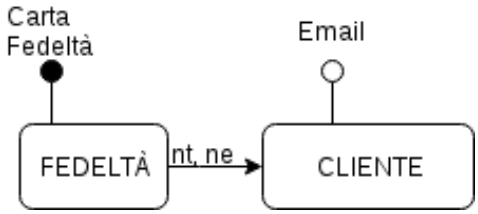
3) Progetto Concettuale

3.1 Persona, cliente, dipendente:

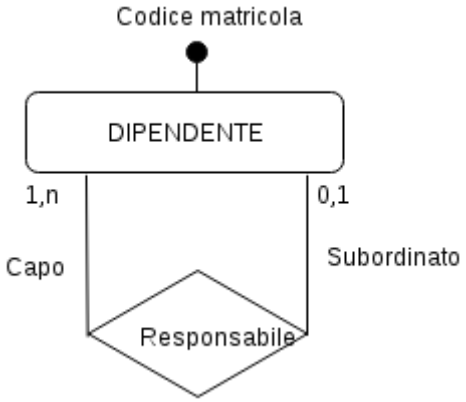
Dato che si pressuppone che sia i dipendenti che i clienti siano caratterizzati dai principali dati anagrafici abbiamo pensato che la soluzione migliore era creare una gerarchia tra le due entità per fare in modo che ereditassero gli usuali dati anagrafici da una persona.



Per quanto riguarda i clienti dato che possono essere presenti dei clienti fedeli al fast food e sono in possesso di una carta fedeltà abbiamo pensato di creare una gerarchia singola per clienti che erano anche fedeli.



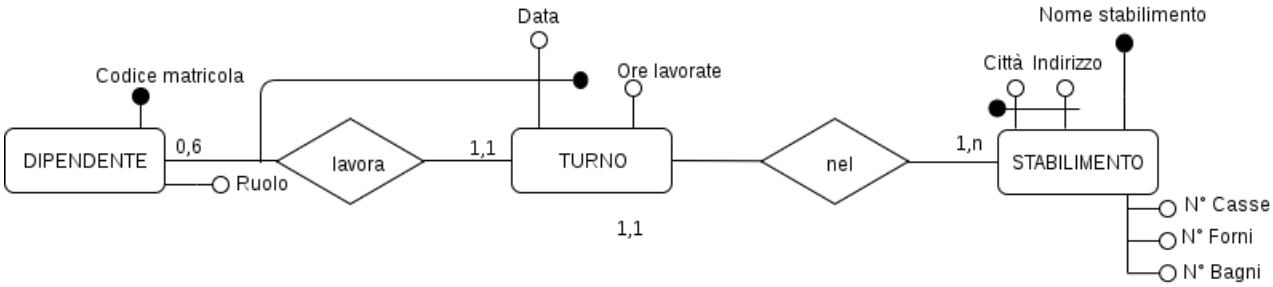
Parlando invece dei dipendenti per definire chi sia il capo/responsabile di ogni dipendente abbiamo inserito un'auto-associazione sui dipendenti affinché sia semplice ricavare i responsabili e i titolari.



3.2 Turno e stabilimento:

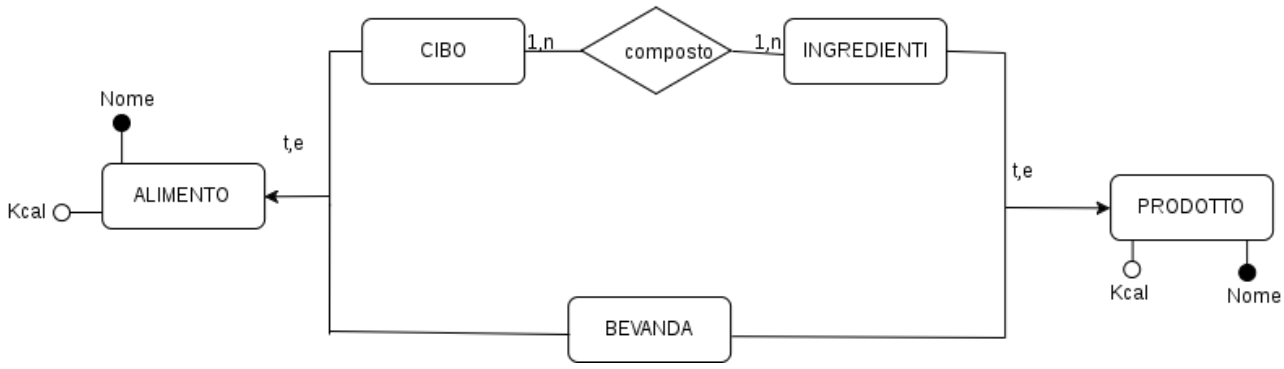
La gestione dei turni all'interno di ogni stabilimento inizialmente fa pensare ad una semplice associazione molti a molti a molti in cui il dipendente puo lavorare al massimo 6 giorni la settimana:

A questo punto pero ci si accorge che un dipendente potrebbe lavorare in piu stabilimenti nello stesso momento e se lasciassimo lo schema precedente un dipendente potrebbe lavorare solo una volta in uno stabilimento(vincolo non presente nella traccia), per cui abbiamo deciso di reificare l'associazione lavora e di trasformarla nell'entità turno mettendo come chiave dell'entita turno la data del turno e il nome del dipendente.
Il turno sara poi associato ad un solo stabilimento per turno :



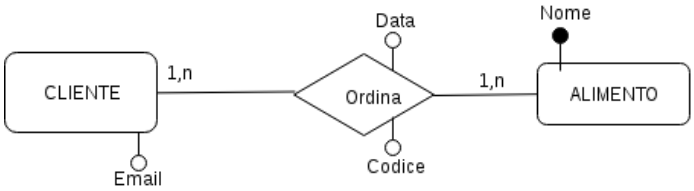
3.3 Alimento e prodotto:

Un'altra doppia gerarchia viene fuori per differenziare un alimento servito in fast food che può essere suddiviso in bevanda e cibo che a sua volta è composto da ingredienti.
Successivamente poi abbiamo pensato che un fornitore non fornisce i cibi gia completi ma fornisce i singoli ingredienti, ecco perchè abbiamo creato la seconda gerarchia di prodotti da cui ereditano ingredienti e bevande.

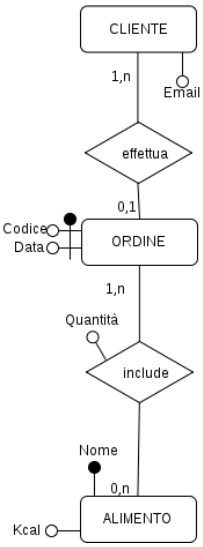


3.4 Ordine:

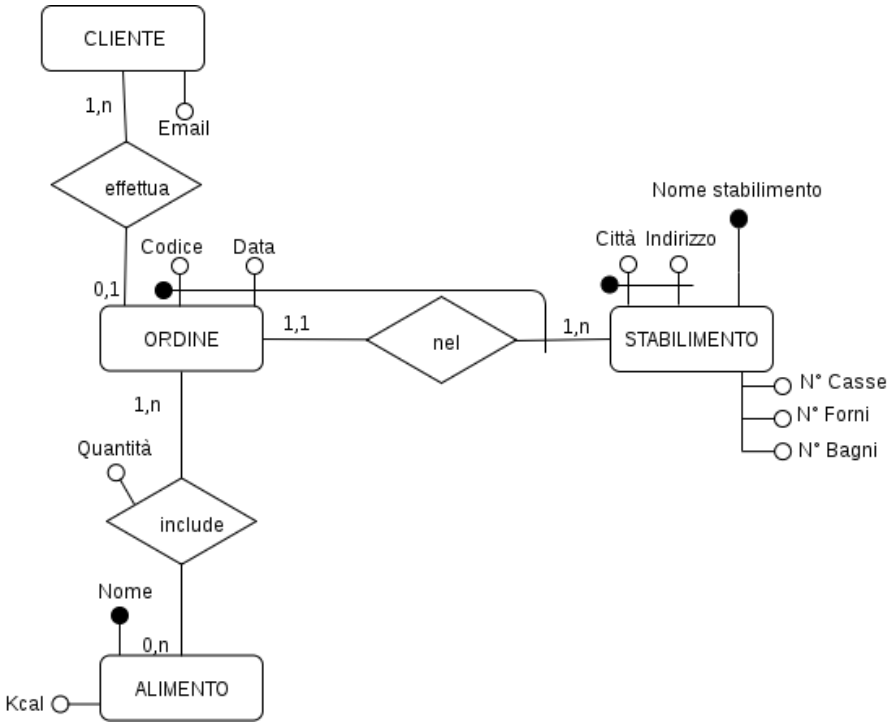
Un primo sguardo alla traccia fa pensare ad una semplice associazione molti a molti ma cosi facendo non si possono ordinare più di due volte lo stesso menù.
Inoltre Data e Codice possono ripertersi.



La soluzione che abbiamo perciò trovato è stata quella di creare una nuova entità Ordine, identitificata dalla chiave composta da Codice e Data e legata da una associazione molti a molti con gli alimenti. Troveremo una associazione ogni qualvolta un alimento sarà presente in un dato scontrino. Inoltre abbiamo deciso di impostare l'associazione tra ordine e cliente con cardinalità 0,1 in quanto cosi facendo non è necessario che un cliente debba dare tutti i dati anagrafici nel caso in cui sia un cliente occasionale che non voglia lasciare i propi dati.

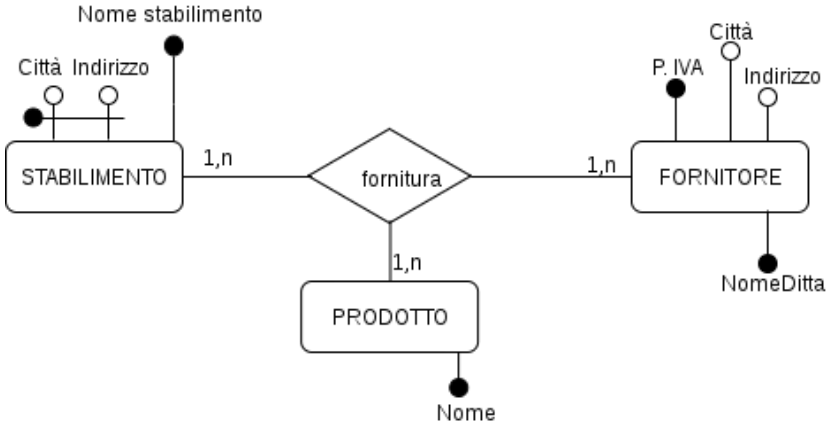


Dopodichè abbiamo ragionato e capito che dato un alimento o un cliente non era possibile risalire allo stabilimento in cui era stato battuto lo scontrino per cui abbiamo aggiunto una associazione tra stabilimento e ordine aggiungendo un'identificatore esterno dello stabilimento e mettendolo in chiave con data e codice dell'ordine:



3.5 Fornitore e Forniture:

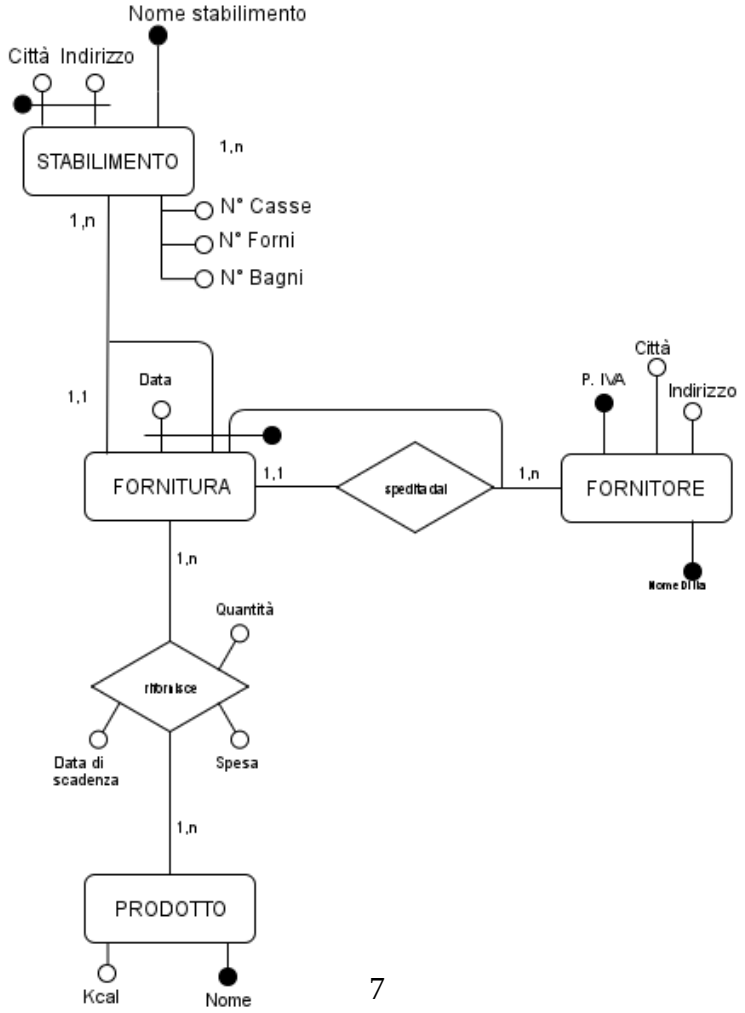
Come primo pensiero per gestire le forniture di prodotti verso degli stabilimenti avevamo pensato ad una semplice associazione ternaria molti a molti:



Con questo schema pero un fornitore non potrebbe rifornire lo stesso stabilimento con lo stesso alimento piu di una volta.

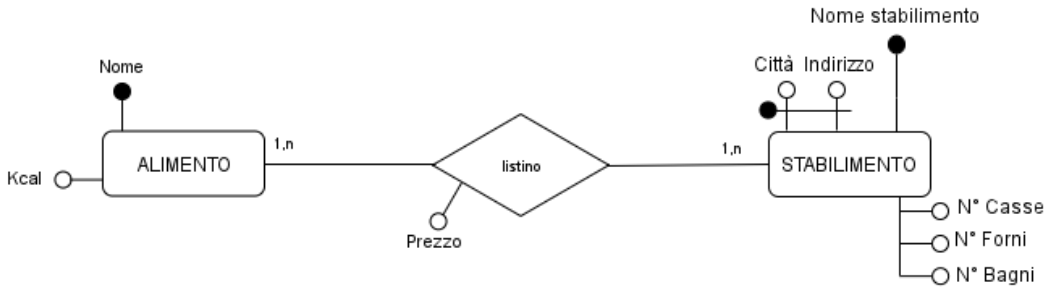
Per questo motivo abbiamo deciso di reificare creando una nuova entità fornitura scegliendo come chiave il nome dello stabilimento e la partita iva del fornitore assieme alla data della fornitura; in questo modo abbiamo imposto come vincolo che un fornitore può fornire qualsisi stabilimento a patto che cambi la data.

Inoltre per poter capire quali alimenti sono presenti nella fornitura è stata messa una associazione molti a molti tra fornitura e prodotto dove si può anche leggere: data di scadenza, quantità e spesa.

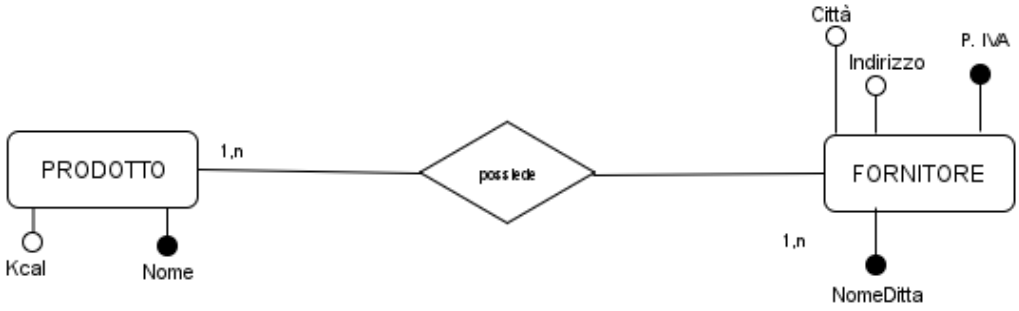


3.6 Altre associazioni:

Ci siamo accorti infine che mancavano due associazioni importanti per poter definire come prima cosa quale fosse il listino degli alimenti di ogni stabilimento, per cui abbiamo creato una associazione molti a molti tra lo stabilimento e gli alimenti e inseriamo come attributo dell'associazione il prezzo dell'alimento:



Come altra associazione abbiamo aggiunto quella che lega i fornitori con i diversi prodotti in modo tale che sia possibile sapere tutti i prodotti che un determinato fornitore riesce a procurare.



3.7 Dato derivato Kcal cibo

Per quanto riguarda l'attributo sull'entita cibo eravamo indecisi se tenerla oppure calcolarla tutte le volte, allora abbiamo effettuato i calcoli standard per il calcolo degli accessi che avremmo dovuto effettuare giornalmente dopo aver definito le due operazione che potrebbero essere maggiormente richieste:

-OP 1 Lettura di tutti i dati di un cibo compreso il totale delle kcal

-OP 2 Modifica di una ricetta(vengono cambiate le associazioni tra cibi e ingredienti)

Abbiamo ipotizzato una possibile **tabella dei volumi**:

Concetto	**Tipo**	**Volume Dati**
Cibo	E	50
Composto	R	150
Ingredienti	E	75

E anche la **tabella delle operazioni:**

Operazione	**Tipo**	**Frequenza**
OP 1	I	50/giorno
OP 2	I	1/giorno

Con dato derivato:

Operazione numero	**Concetto**	**Accessi**	**Tipo**	**Accessi*Frequenza**
OP 1	Cibo	1	L	1*50= 50
OP 2	Composto	3	L	3*1=3
	Composto	3	S	3*2=6
	Prodotto	3	L	3*1=3
	Cibo	1	L	1*1=1
	Cibo	1	S	1*2=2

Quindi con il dato derivato avremo un totale di accessi pari a:
OP1 + OP2 = 50 + 15 = 65/giorno

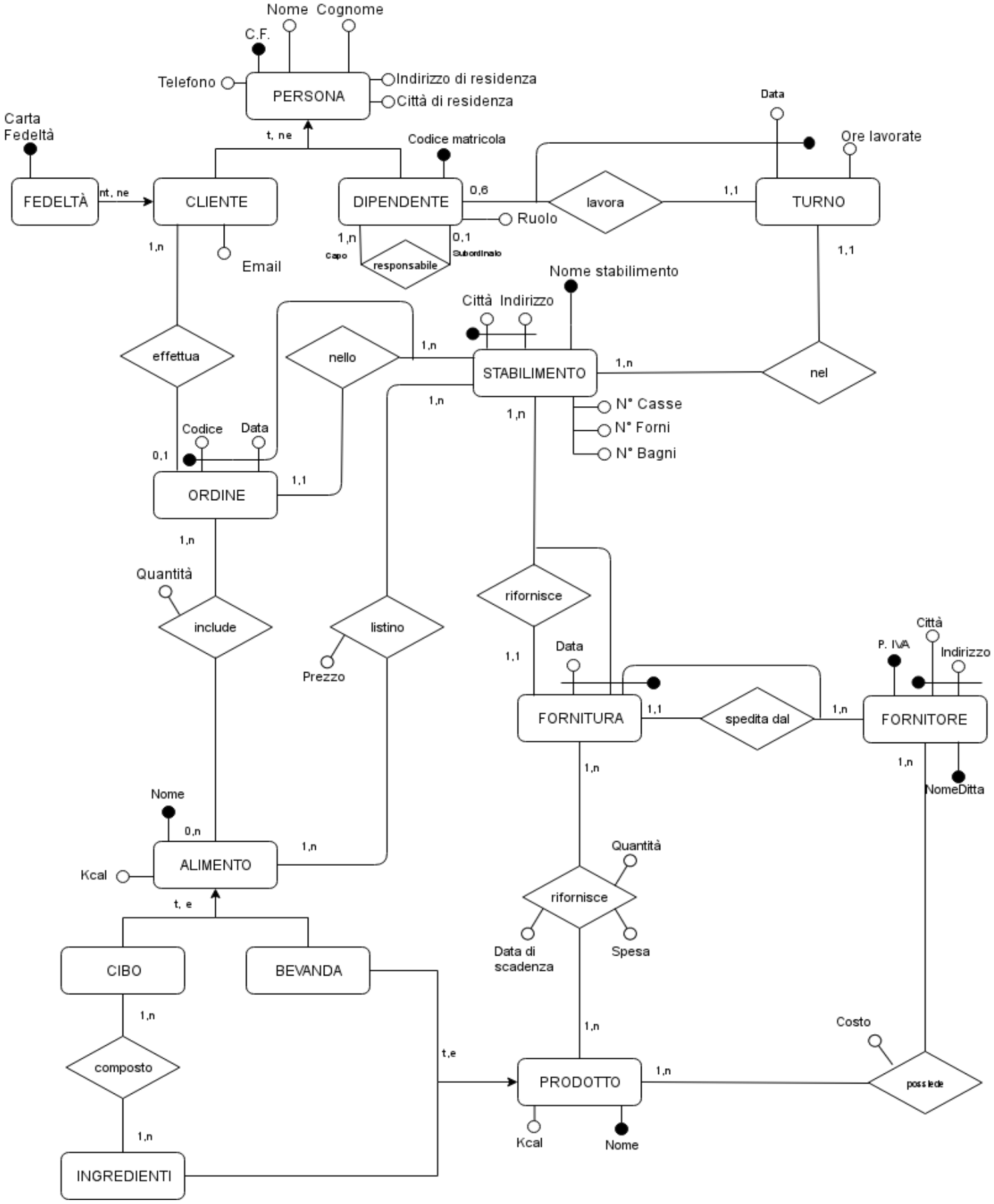
Senza il dato derivato:

Operazione numero	**Concetto**	**Accessi**	**Tipo**	**Accessi*Frequenza**
OP 1	Cibo	1	L	1*50= 50
	Composto	3	L	3*50=150
	Prodotto	3	L	3*50=150
OP 2	Composto	3	L	3*1=3
	Composto	3	S	3*2=6

Senza il dato derivato il numero di accessi è pari a:
OP1 + OP2 = 350 + 9 = 359/giorno

Motivo per cui abbiamo scelto di mantere il dato derivato e di costriure il trigger nella sezione 6.2 che si occupa di gestire il valore delle kcal totali in caso di modifica della ricetta di un cibo.

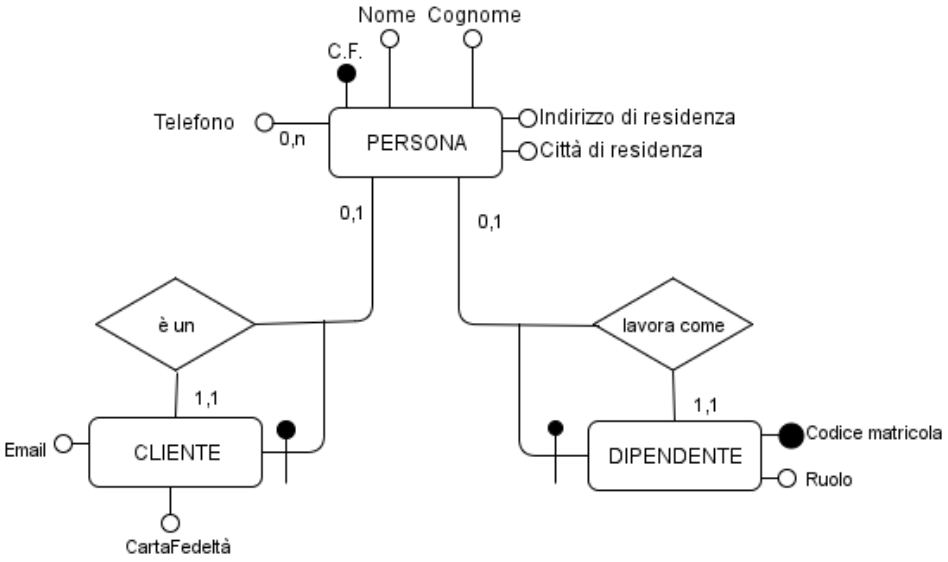
4) Schema ER completo



5) Progetto logico

Per prima cosa abbiamo cercato il metodo migliore per trasformare le gerarchie scegliendo tra i tre metodi principali: mantenimento di tutte le entità, collasso verso l'alto o verso il basso.

Per quanto riguarda la gerarchia che comprende persona cliente e dipendente abbiamo scelto il mantenimento di tutte e tre le entità in quanto una persona, pur essendo dipendente, potrebbe effetuare degli ordini quando non lavora e se avessimo effettuato un collasso verso il basso avremmo avuto ridondanza di dati anagrafici nel caso di persone sia dipendenti che clienti.
Se avessimo scelto invece di effettuare un collasso verso l'alto, allora avremmo avuto vari campi null, di conseguenza ci sarebbe stato uno spreco di memoria.
Inoltre per la gerarchia singola dei clienti in possesso di carta fedeltà abbiamo optato per un collasso verso l'alto aggiungendo l'attributo CartaFedelta al generico cliente a costo di qualche valore null nel caso dei clienti non fedeli.



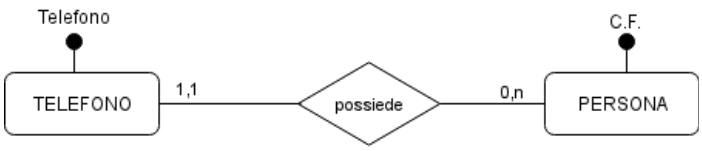
Di conseguenza le tre tabelle verrebbero nel seguente modo:

Persona (CF, Nome, Cognome, IndirizzoResisdenza, CittaResidenza)

Cliente (CF, Email, NumeroCarta)
FK: CF references Persona

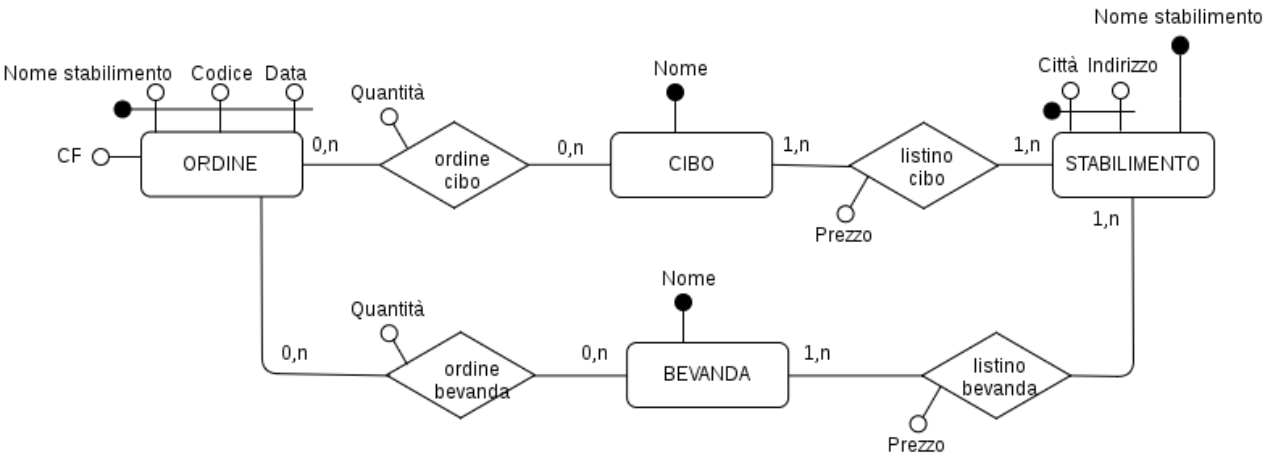
Dipendente (Matricola, CF, Ruolo, MatricolaCapo)
AK: CF
FK: CF references Persona
FK: MatricolaCapo references Dipendente

Per quanto riguarda l'attributo Telefono associato alla Persona che può essere più di uno, abbiamo creato una nuova entità telefono in associazione con persona:



Telefono (Telefono, CF)
FK: CF references Persona

Per quanto riguarda la gerarchia dell'alimento abbiamo optato per un collasso verso il basso a costo di raddoppiare le essociazioni ma di renderle piu chiare e specifiche ovvero, in questo modo ci sarà una inclusione nell'ordine di cibi e bevande ma soprattutto un listino per le bevande e uno per i cibi a sé stanti.



Cibo (Nome, kcal)

Bevanda (Nome)
FK: Nome references Prodotto

CiboOrdine (Codice, Data, NomeStabilimento, NomeCibo, Quantita)
FK: Codice, Data, NomeStabilimento references Ordine
FK: NomeCibo references Cibo

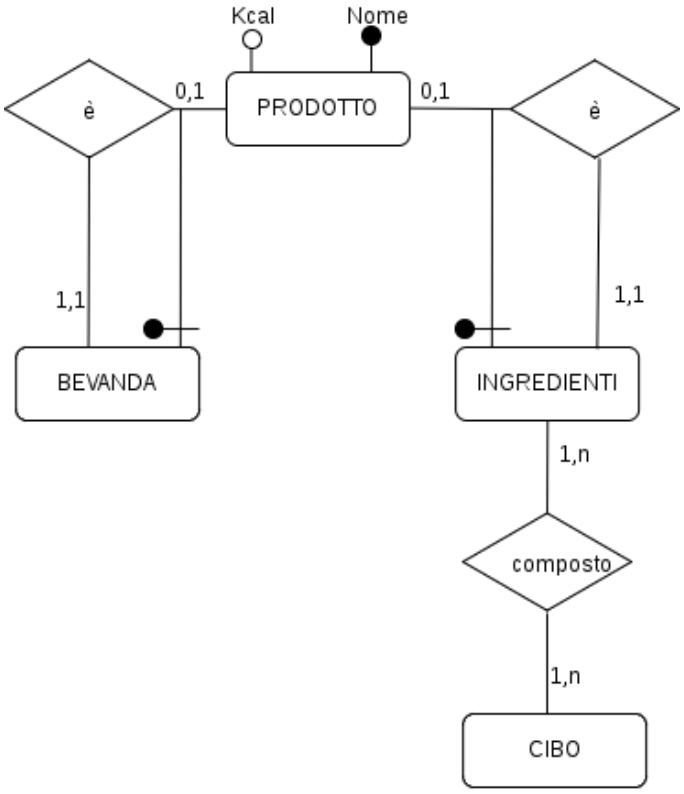
BevandaOrdine (Codice, Data, NomeStabilimento NomeBevanda, Quantita)
FK: Codice, Data, NomeStabilimento references Ordine
FK: NomeBevanda references Cibo

Stabilimento (Nome, Citta, Indirizzo, NForni, NCasse, NBagni)
AK: Citta, Indirizzo

Ordine (Codice, Data, NomeStabilimento, CF)
FK: NomeStabilimento references Stabilimento
FK: CF references Cliente

L'ultima gerarchia che invece fa la distinzione dei vari prodotti tra bevande e ingrdienti abbiamo deciso di manteretutte le intita sia prodotto che cibo che bevanda in modo dale da ridurre le associazione poiché se avessimo svolto un collasso verso il basso sarebbero raddoppiate verso la fornitura e l'associazione che lega i prodotti ai vari fornitori .
Cosi facendo inoltre riusciamo a collegarci bene alla gerarchia degli alimenti che avevamo deciso di collassare verso il basso e che quindi ci faceva creare una entita a se bevanda che entrera in asoociazione anche con il prodotto.

Inoltre mantenendo sia il cibo che gli ingredienti come due entità a sé stanti è semplice creare l'associazione tra i due che ci indica di quali ingredienti è composto un cibo.



Ingrediente (Nome)
FK: Nome references Prodotto

ComposizioneCibo (NomeCibo, NomeIngrediente)
FK: NomeCibo references Cibo
FK: NomeIngrediente references Ingrediente

Prodotto (Nome, Kcal)

Le restanti entità possone essere così tradotte dallo schema ER con i normali metodi di aggiunta di attributi per creare le essociazioni in caso di associazioni uno a molti opure una a una.
Per le associazioni molti a molti è stata creata una semplice nuova entità avente come chiave le chiavi delle entità che creavano l'associazione.

Fornitura (Data, NomeStabilimento, PIVA)
FK: NomeStabilimento references Stabilimento
FK: PIVA references Fornitore

Fornitore (PIVA, NomeDitta, Citta, Indirizzo)
AK: NomeDitta
AK: Città, Indirizzo

Rifornimento (Data, NomeStabilimento, PIVA, NomeProdotto, Quantita, Spesa, DataScadenza)
FK: Data, NomeStabilimento, PIVA references Fornitura
FK: NomeProdotto references Prodotto

InventarioFornitore (PIVA, NomeProdotto)
FK: PIVA references Fornitore
FK: NomeProdotto references Prodotto

ListinoCibo (NomeCibo, NomeStabilimento, Prezzo)
FK: NomeCibo references Cibo
FK: NomeStabilimento references Stabilimento

ListinoBevande (NomeBevanda, NomeStabilimento, Prezzo)
FK: NomeBevanda references Bevanda
FK: NomeStabilimento references Stabilimento

Turno (Matricola, Data, Ore, NomeStabilimento)
FK: Matricola references Dipendente
FK: NomeStabilimento references Stabilimento

6) Implementazione in SQL

Per il codice SQL abbiamo utilizzato come DBMS PostgreSQL 9.6 .

6.1 Creazione tabelle

Per quanto riguarda la conversione dallo schema logico al codice SQL è stato tutto molto immediato e lo si puo vedere all'interno del file **FastFood.sql**.
L'unica cosa che è stata aggiunta sono i check in alcune tabelle affinché venissero rispettati tutti i vincoli imposti dalla traccia.
Per esempio, per sodisfare il fatto che i ruoli dei dipendenti sono ben definiti è stato aggiunto un controllo sul ruolo che se diverso da quelli preimpostati annulla l'inserimento o la modifica di quest'ultimo:

```
Create table dipendente
(
        Matricola varchar(6) primary key,
        cf varchar(16) unique not null references persona on delete cascade on update cascade,
        ruolo varchar(14) not null check(  ruolo='inserviente' or
                                           ruolo='cassiere' or
                                           ruolo='cuoco' or
                                           ruolo='responsabile' or
                                           ruolo='amministratore'),
        matricola_capo varchar(6)default '000001' references dipendente on delete set default on update
        cascade
);
```

Inoltre sono stati inseriti numerosi controlli affinche valori fossero maggiori di zero o altri per verificare la correttezza di date come per esempio quella in cui si verifica che la data di scadenza dei prodotti di un rifornimento sia piu grande della data odierna altrimenti non avrebbe senso ordinare prodotti gia scaduti:

```
create table rifornimento
(
        data date,
        nome_stabilimento varchar(50),
        p_iva varchar(11),
        nome_prodotto varchar(20) references prodotto on delete cascade on update cascade,
        quantita smallint check(quantita >0) default null,
        spesa real check(spesa>0),
        data_scadenza date check(data_scadenza>data),
        foreign key (data,nome_stabilimento,p_iva) references fornitura (data,nome_stabilimento,p_iva)
        on delete cascade on update cascade
);
```

6.2 Trigger e procedure

Per soddisfare il vincolo:

> *"I turni sono settimanali e devono gestire il numero di dipendenti in modo che il numero del personale di uno specifico ruolo non ecceda il numero di postazioni presenti nella filiale di tale ruolo"*

abbiamo utilizzato il seguente trigger:

```
CREATE OR REPLACE FUNCTION controllo_disponibilita_stabilimento() RETURNS trigger as $$
DECLARE
        nuovoRuolo varchar(14);
        postiDisponibili numeric(2,0);
        postiOccupati numeric(2,0);
        posto char(12);
BEGIN

        SELECT INTO nuovoRuolo ruolo FROM dipendente WHERE matricola=NEW.matricola;

        IF nuovoRuolo='cassiere' THEN
                SELECT INTO postiDisponibili numero_casse FROM stabilimento WHERE nome=NEW.nome_stabilimento;
END IF;
        IF nuovoRuolo='inserviente' THEN
                SELECT INTO postiDisponibili numero_bagni FROM stabilimento WHERE nome=NEW.nome_stabilimento;
END IF;
        IF nuovoRuolo='cuoco' THEN
                SELECT INTO postiDisponibili numero_forni FROM stabilimento WHERE nome=NEW.nome_stabilimento;
END IF;
        IF nuovoRuolo='responsabile' THEN
                postiDisponibili=1; END IF;
        SELECT INTO postiOccupati count(*) FROM turno AS T,dipendente AS D
        WHERE T.data=NEW.data AND T.Matricola=D.Matricola AND ruolo=nuovoRuolo;

        IF postiOccupati>postiDisponibili THEN RAISE EXCEPTION
        'Lo stabilimento % in data % ha i posti per il ruolo % al completo',
        NEW.nome_stabilimento,NEW.data,nuovoRuolo; END IF;
return NEW;
END
$$LANGUAGE plpgsql;

CREATE TRIGGER controllo_inserimento_turno
AFTER INSERT OR UPDATE on turno
FOR EACH ROW EXECUTE PROCEDURE controllo_disponibilita_stabilimento();
```

Ad ogni inserimento o aggiornamento della relazione turno viene richiamata la procedura controllo_disponibilita_stabilimento(), in cui per prima cosa viene memorizzato il ruolo del dipendente che partecipa all'inserimento, dopodiché controlla se sono presenti postazioni disponibili per tale ruolo nello stabilimento specificato, in caso di risposta affermativa, la tupla viene inserita(o la modifica viene effettuata), altrimenti si alza una eccezione con un messaggio di errore e la tupla non viene inserita.

Vista la presenza del codice fiscale e della partita iva abbiamo deciso di implementare due trigger per verificare la loro correttezza in fase di inserimento o di aggiornamento.

```
CREATE OR REPLACE FUNCTION controllo_codice_fiscale_partita_iva() RETURNS trigger as $$
DECLARE
	risultato text;
	CF_PIVA_normalizzato text;
	CF_PIVA varchar(16);
BEGIN

	IF pg_typeof(NEW)::text = 'persona'
	THEN
		CF_PIVA=NEW.cf;
	ELSE
		CF_PIVA=NEW.p_iva;
	END IF;
	select into CF_PIVA_normalizzato codice_fiscale_normalize(CF_PIVA::text);
	select into risultato codice_fiscale_error(CF_PIVA_normalizzato);
	IF risultato IS NOT NULL THEN RAISE EXCEPTION'codice fiscale o partita iva non valido/a: %',risultato; END IF;

	IF pg_typeof(NEW)::text = 'persona'
    THEN
	NEW.cf:=CF_PIVA_normalizzato::char(16);
    ELSE
	NEW.p_iva=CF_PIVA_normalizzato::char(11);
    END IF;

RETURN NEW;
END
$$LANGUAGE plpgsql;


CREATE TRIGGER trigger_controllo_codice_fiscale
BEFORE INSERT OR UPDATE on persona
FOR EACH ROW EXECUTE PROCEDURE controllo_codice_fiscale_partita_iva();

CREATE TRIGGER trigger_controllo_partita_iva
BEFORE INSERT OR UPDATE on fornitore
FOR EACH ROW EXECUTE PROCEDURE controllo_codice_fiscale_partita_iva();
```

Nella procedura sono presenti due funzioni che controllano la correttezza dei dati inseriti.
codice_fiscale_normalize(text) prende come argomento una stringa e ritorna un testo formattato a dovere(cancella gli spazi, imposta a maiuscolo eventuali lettere minuscole...)

codice_fiscale_error(text) prende come argomento una stringa e ritorna valore null se si tratta di un codice fiscale o partita iva corretti, oppure ritorna un testo nel caso in cui il codice non sia stato inserito correttamente; il messaggio di ritono specifica il tipo di errore.

Se consideriamo l'attributo Kcal della relazione CIBO come somma delle Kcal di tutti gli ingredienti che partecipano alla associazione COMPOSTO di una specifica tupla in CIBO, è necessario aggiornare tale elemento ogni qualvolta si effettui un aggiornamento o un nuovo inserimento di un ingrediente.

```
CREATE OR REPLACE FUNCTION aggiornamento_kcal() RETURNS trigger AS $$
DECLARE
        totale smallint;
BEGIN
        SELECT INTO totale SUM(P.kcal) FROM composizione_cibo AS C, prodotto AS P
        WHERE C.nome_ingrediente=P.nome AND C.nome_cibo=NEW.nome_cibo;


        UPDATE cibo SET kcal=totale WHERE nome=NEW.nome_cibo;
RETURN NEW;
END
$$ LANGUAGE plpgsql;


CREATE TRIGGER trigger_aggiornamento_kcal
AFTER INSERT OR UPDATE on composizione_cibo
FOR EACH ROW EXECUTE PROCEDURE aggiornamento_kcal();
```

Nel momento in cui vengono effettuati degli ordini, per poter procedere con l'inserimento dei dati è necessario conoscere: il cliente che richiede l'ordine, il nome dello stabilimento, la lista dei cibi e la lista delle bevande.
La presenza delle liste permette al cliente di effettuare un ordine specificando un numero arbitrario di alimenti nella stessa procedura di inserimento.
La gestione delle liste è stata fatta tramite l'utilizzo di array la cui struttura è composta ciclicamente in [nome alimento,quantità,nome alimento,quantità...]

```
CREATE OR REPLACE FUNCTION ordine(CF char(16), nome_stab varchar(50), lista_cibo varchar(50)[],lista_bevande
varchar(50)[]) RETURNS void as $$
DECLARE
        cod integer;
BEGIN
        select into cod creazione_ordine(CF, nome_stab);

        FOR i IN 1..array_length(lista_cibo,1) BY 2
        LOOP
                insert into cibo_ordine values(cod,now(),nome_stab,lista_cibo[i],lista_cibo[i+1]::INTEGER);
        END LOOP;

        FOR i IN 1..array_length(lista_bevande,1) BY 2

        LOOP
        insert into bevanda_ordine values(cod,now(),nome_stab,lista_bevande[i],lista_bevande[i+1]::INTEGER);
    END LOOP;

END;
 $$ LANGUAGE plpgsql;
```

La prima operazione che la procedura ordine() esegue è la creazione dell'ordine, infatti la funzione creazione_ordine() inserisce una nuova istanza nella relazione ordine(tabella che contiene solo informazioni relative all'ordine stesso, come ad esempio la data e il codice giornaliero, ma non contiene gli alimenti ordinati) e ritorna un codice necessario per rendere univoca la chiave delle relazioni cibo_ordine e bevanda_ordine.

Di seguito è riportata la funzione creazione_ordine()

```
CREATE OR REPLACE FUNCTION creazione_ordine(CF char(16), nome_stab varchar(50)) RETURNS Integer as $$
DECLARE
        cod Integer;
        oggi date;
BEGIN
        oggi=now();

        SELECT INTO cod MAX(O.codice) FROM ordine as O
        WHERE O.data=oggi AND O.nome_stabilimento=nome_stab;

        IF cod IS NULL THEN cod=0; END IF;

        INSERT INTO ordine VALUES(oggi,cod+1,nome_stab,CF);

RETURN cod+1;
END;
 $$ LANGUAGE plpgsql;
```

7) Operazioni implementate nel programma:

All'interno del software CLI da noi implementato per la gestione del database abbiamo inserito alcune operazioni di base che ci sembravano piu opportune e che sarebbero state maggiormente utilizzate.

Le principale azioni possono essere categorizzate in inserimento, modifica e visualizzazione di dati all'interno del database.

7.1 Inserimenti (INSERT)

7.1.1 Inserimento cliente

Come prima operazione di inserimento abbiamo quella che ci permette di inserire un nuovo cliente. Sono state utilizzate due Insert differenti, una per l'inserimento di una normale persona e una per l'inserimento del cliente:

1. *"INSERT INTO PERSONA(cf,nome,cognome,indirizzo_residenza,citta_residenza) VALUES (?,?,?,?,?);"*
2. *"INSERT INTO CLIENTE(cf,email,numero_carta) VALUES (?,?,?);"*

Affinchè sia possibile l'inserimento di un cliente deve esistere il riferimento della persona ovvero il Codice Fiscale motivo per cui viene creata prima la persona poi il cliente.

7.1.2 Inserimento nuovo ordine

La seconda operazione è stata creata per poter effettuare un nuovo ordine e per fare cio viene utilizzata la procedura implementata in SQL e sopra mostrata che viene richiamata attraverso un metodo del JDBC che si chiama **prepareCall("{call ordine(?,?,?,?)}").**
A questa prepareCall vengono settati i valori presi in input che verranno sostituiti ai ?.
I valori di input previsti sono:
-Cf
-Nome stabilimento
-Lista dei cibi
-Lista delle bevande.
Per le due liste abbiamo utilizzato due ArrayList ed è stato effettuato un casting per renderli compatibili con gli array in SQL attraverso la funzione: **createArrayOf("VARCHAR", elencoCibi.toArray()))** che si occupa di creare un array di VARCHAR da quello che gli viene passato in input.

7.1.3 Inserimento nuova fornitura

Come terza operazione è stato usato lo stesso metodo usato per la seconda ovvero inserimento di nuove forniture attraverso l'utilizzo di una procedura: **"{call fornitura(?,?,?)}".**
Questa funzione opera in un modo molto simile a quella precedente da come si puo vedere sul file **FastFood.sql**.

7.1.4 Inserimento di un nuovo cliente ai clienti fedeli

Anche per questa funzionalita è stata aggiunta una funzione SQL che opera passandogli il solo codice fiscale di un cliente e gli associa un numero di carta fedeltà progressivo ovvero il numero massimo di carta fedeltà a cui si è arrivati più uno.
La procedura va chiamata nel seguente modo: **"{? = call fedele(?)}"** e cosi facendo attraverso il primo punto di domanda è possibile ricavare il numero di carta che viene associato al cliente essendo gestito automaticamente dal database.

7.1.5 Inserimento di un nuovo stabilimento

Come ultimo inserimento abbiamo quello che prevede l'inserimento di un nuovo stabilimento. Per fare cio abbiamo usato una semplice insert:

"INSERT INTO stabilimento(nome,citta,indirizzo,numero_forni,numero_bagni,numero_casse) VALUES (?,?,?,?,?,?)"

7.2 Aggiornamenti (UPDATE)

7.2.1 Aggiornamento prezzo di un prodotto venduto da un dato fornitore

Per l'aggiornamento del prezzo di un prodotto rivenduto da un certo fornitore abbiamo utilizzato un semplice update inserito in un prepareStatement() contenente la seguente query di aggiornamento:

"UPDATE inventario_fornitore SET costo = ? WHERE p_iva = ? and nome_prodotto = ?;"

7.2.2 Aggiornamento listino prezzi cibi di uno stabilimento

Per quanto riguarda l'aggiornamento del listino prezzi dei cibi di uno stabilimento abbiamo usato una query praticamente uguale a quella per l'aggiornamento di prezzi di prodotti:

"UPDATE listino_cibo SET prezzo = ? WHERE nome_cibo = ? and nome_stabilimento = ?;"

7.2.2 Aggiornamento listino prezzi bevande di uno stabilimento

Come ultima operazione abbiamo inserito l'aggiornaneto del prezzo di una bevanda con una query come le precedenti:

"UPDATE listino_bevande SET prezzo = ? WHERE nome_bevanda = ? and nome_stabilimento = ?;"

7.3 Visualizzazione dati (SELECT)

Per quanto riguarda la visualizzazione di determinati dati del database abbiamo scelto tre principali interrogazioni quali:

1.Mostra del listino dei cibi e delle bevande di uno stabilimento

```
"SELECT c.nome_cibo, c.prezzo
FROM listino_cibo AS c, stabilimento AS s
WHERE c.nome_stabilimento=s.nome AND c.nome_stabilimento=?;"
```

il cui output è seguito dal listino delle bevande ottenuto da:

```
"SELECT b.nome_bevanda, b.prezzo
FROM listino_bevande AS b, stabilimento AS s
WHERE b.nome_stabilimento=s.nome AND b.nome_stabilimento=?;"
```

2. Mostra dello storico di tutti gli ordini di un cliente:

```
"SELECT data,codice,nome_stabilimento,alimento,quantita "
FROM(
        (SELECT  o.cf,o.data,o.codice,o.nome_stabilimento,b.nome_bevanda as alimento,b.quantita
        FROM ordine as o, bevanda_ordine as b ORDER BY o.data,o.codice,nome_bevanda)
        UNION
        (SELECT o.cf,o.data,o.codice,o.nome_stabilimento,c.nome_cibo as alimento,c.quantita
        FROM ordine as o, cibo_ordine as c ORDER BY o.data,o.codice,c.nome_cibo )
) as t WHERE cf = ? ORDER BY alimento;"
```

3. Mostra su schermo il numero di ordini che sono stati fatti nell'ultimo anno da parte di ogni cliente:

```
"SELECT  p.cf, p.nome, p.cognome, count(*) as numero_ordini
FROM ordine o, persona p WHERE o.cf=p.cf and extract(year from data) = extract(year from
current_date ) GROUP BY p.cf ORDER BY numero_ordini desc;"
```