Basi di Dati

Progetto A.A. 2022/2023

Frutta S.r.l.

0285063

Nicola Violante

Indice

1. Descrizione del Minimondo

Sistema per la gestione di un mercato ortofrutticolo

L'azienda Frutta S.r.l. gestisce la vendita all'ingrosso di frutta e verdura. L'azienda tratta diversi prodotti, ciascuno caratterizzato da un nome e da un codice univoco alfanumerico attraverso cui il prodotto viene identificato. Per ciascun prodotto è inoltre noto se appartenga alla categoria frutta o verdura. Ovviamente, viene mantenuta l'informazione sul prezzo al chilogrammo.

L'azienda effettua vendite all'ingrosso verso privati cittadini che siano muniti di partita IVA. Per ogni cittadino sono noti la partita IVA, il nome e l'indirizzo di residenza. È possibile mantenere un numero arbitrario di contatti, ad esempio numeri di telefono, di cellulare, di indirizzi email. I clienti devono avere un indirizzo di fatturazione, che può essere differente dall'indirizzo di spedizione dei prodotti.

I fornitori di Frutta S.r.l. sono identificati attraverso un codice fornitore; per ciascun fornitore sono inoltre noti il nome, il codice fiscale ed un numero arbitrario di indirizzi. Il fornitore può fornire diversi prodotti ortofrutticoli.
Frutta S.r.l. ha un dipartimento di gestione di magazzino che tiene traccia delle giacenze ed effettua, periodicamente, ordini ai fornitori per mantenere una giacenza di tutti i prodotti trattati. Di ciascun prodotto in magazzino è di interesse tenere traccia anche della data di scadenza.
I gestori del magazzino possono generare un report che indichi quali e quanti prodotti sono in scadenza, organizzati per giorno, nella settimana successiva alla generazione del report.

Nell'ambito di un ordine è di interesse sapere a quale indirizzo questo deve essere inviato, e quale contatto fornire al corriere per mettersi in contatto con il destinatario in caso di problemi nella consegna. Non è possibile aprire un ordine se non vi è disponibilità in magazzino. L'ordine deve indicare quali prodotti sono stati ordinati ed in quali quantità.

2. Analisi dei Requisiti

Identificazione dei termini ambigui e correzioni possibili

Linea	Termine	Nuovo termine	Motivo correzione
9	Privati cittadini	Clienti	Disambuiguato parlando con committente
16	Indirizzi	Indirizzi delle sedi	Indirizzi non specifica bene di cosa
27	Disponibilità	Giacenze	Indicano lo stesso concetto
19	Ordini	Richieste	Il termine ordini è usato per indicare la spedizione da Frutta S.r.l. al cliente
25	Indirizzo	Indirizzo di spedizione	Indirizzo non specifica bene di cosa

Specifica disambiguata

Sistema per la gestione di un mercato ortofrutticolo

L'azienda Frutta S.r.l. gestisce la vendita all'ingrosso di frutta e verdura. L'azienda tratta diversi prodotti, ciascuno caratterizzato da un nome e da un codice univoco alfanumerico attraverso cui il prodotto viene identificato. Per ciascun prodotto è inoltre noto se appartenga alla categoria frutta o verdura. Ovviamente, viene mantenuta l'informazione sul prezzo al chilogrammo.

L'azienda effettua vendite all'ingrosso verso clienti che siano muniti di partita IVA. Per ogni cittadino sono noti la partita IVA, il nome e l'indirizzo di residenza. È possibile mantenere un numero arbitrario di contatti, ad esempio numeri di telefono, di cellulare, di indirizzi email. I clienti devono avere un indirizzo di fatturazione, che può essere differente dall'indirizzo di spedizione dei prodotti.

I fornitori di Frutta S.r.l. sono identificati attraverso un codice fornitore; per ciascun fornitore sono inoltre noti il nome, il codice fiscale ed un numero arbitrario di indirizzi delle sedi. Il fornitore può fornire diversi prodotti ortofrutticoli.
Frutta S.r.l. ha un dipartimento di gestione di magazzino che tiene traccia delle giacenze ed effettua, periodicamente, richieste ai fornitori per mantenere una giacenza di tutti i prodotti trattati. Di ciascun prodotto in magazzino è di interesse tenere traccia anche della data di scadenza.

I gestori del magazzino possono generare un report che indichi quali e quanti prodotti sono in scadenza, organizzati per giorno, nella settimana successiva alla generazione del report.

Nell'ambito di un ordine è di interesse sapere a quale indirizzo di spedizione questo deve essere inviato, e quale contatto fornire al corriere per mettersi in contatto con il destinatario in caso di problemi nella consegna. Non è possibile aprire un ordine se non vi è giacenza in magazzino. L'ordine deve indicare quali prodotti sono stati ordinati ed in quali quantità.

Glossario dei Termini

Termine	Descrizione	Sinonimi	Collegamenti
Prodotto	Il tipo di merce che si intende vendere o acquistare (frutta o verdura)	Prodotto ortofrutticolo	Fornitore, Ordine
Cliente	Colui che richiede il prodotto dell'azienda	Cittadino	Ordine
Fornitore	Chi fornisce i prodotti all'azienda		Prodotti
Ordine	Insieme di prodotti che il cliente richiede all'azienda		Cliente, Prodotto

Raggruppamento dei requisiti in insiemi omogenei

Frasi di carattere generale
L'azienda Frutta S.r.l. gestisce la vendita all'ingrosso di frutta e verdura. […] Frutta S.r.l. ha un dipartimento di gestione di magazzino che tiene traccia delle giacenze ed effettua, periodicamente, richieste ai fornitori per mantenere una giacenza di tutti i prodotti trattati. […] I gestori del magazzino possono generare un report che indichi quali e quanti prodotti sono in scadenza, organizzati per giorno, nella settimana successiva alla generazione del report.

Frasi relative a Prodotto
L'azienda tratta diversi prodotti, ciascuno caratterizzato da un nome e da un codice univoco alfanumerico attraverso cui il prodotto viene identificato. Per ciascun prodotto è inoltre noto se appartenga alla categoria frutta o verdura. Ovviamente, viene mantenuta l'informazione sul prezzo al chilogrammo. […] Di ciascun prodotto in magazzino è di interesse tenere traccia anche della data di scadenza.

Frasi relative a Cliente
L'azienda effettua vendite all'ingrosso verso clienti che siano muniti di partita IVA. Per ogni cittadino sono noti la partita IVA, il nome e l'indirizzo di residenza. È possibile mantenere un numero arbitrario di contatti, ad esempio numeri di telefono, di cellulare, di indirizzi email. I clienti devono avere un indirizzo di fatturazione, che può essere differente dall'indirizzo di spedizione dei prodotti.

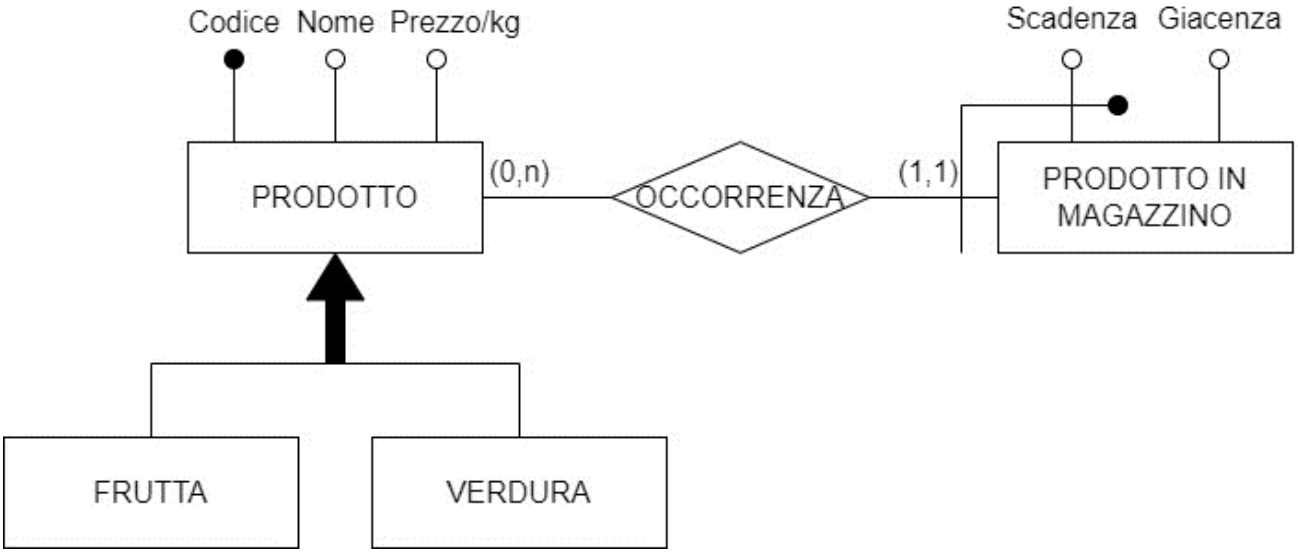
Frasi relative a Fornitore
I fornitori di Frutta S.r.l. sono identificati attraverso un codice fornitore; per ciascun fornitore sono inoltre noti il nome, il codice fiscale ed un numero arbitrario di indirizzi delle sedi. Il fornitore può fornire diversi prodotti ortofrutticoli.

Frasi relative a Ordine
Nell'ambito di un ordine è di interesse sapere a quale indirizzo di spedizione questo deve essere inviato, e quale contatto fornire al corriere per mettersi in contatto con il destinatario in caso di problemi nella consegna. Non è possibile aprire un ordine se non vi è giacenza in magazzino. L'ordine deve indicare quali prodotti sono stati ordinati ed in quali quantità.

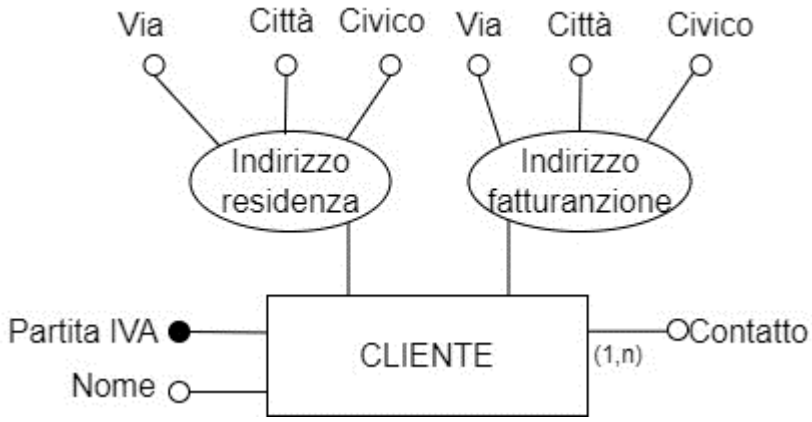
3. Progettazione concettuale

Costruzione dello schema E-R

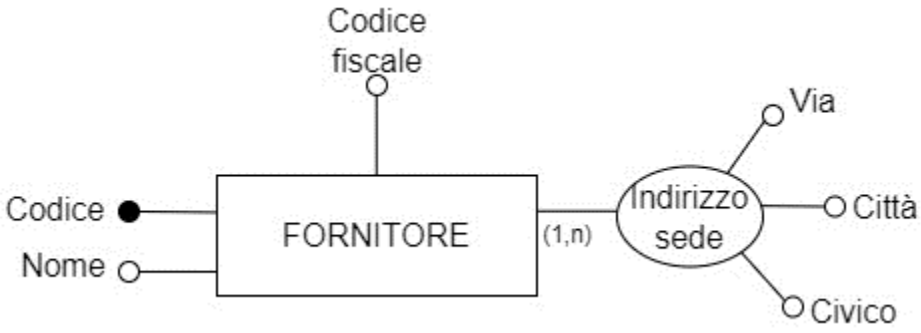
La prima entità che ho deciso di sviluppare è quella che rappresenta i prodotti che Frutta S.r.l. ha a disposizione per la vendita. Come si evince dal testo un prodotto può essere di tipo frutta o verdura, per questo ho optato per una generalizzazione totale. Come altri attributi siamo interessati a sapere il codice (che è identificativo), il nome e il prezzo al kg. Inoltre, ho creato una relazione di tipo "instance of" per tenere traccia delle diverse date di scadenza dei prodotti disponibili.



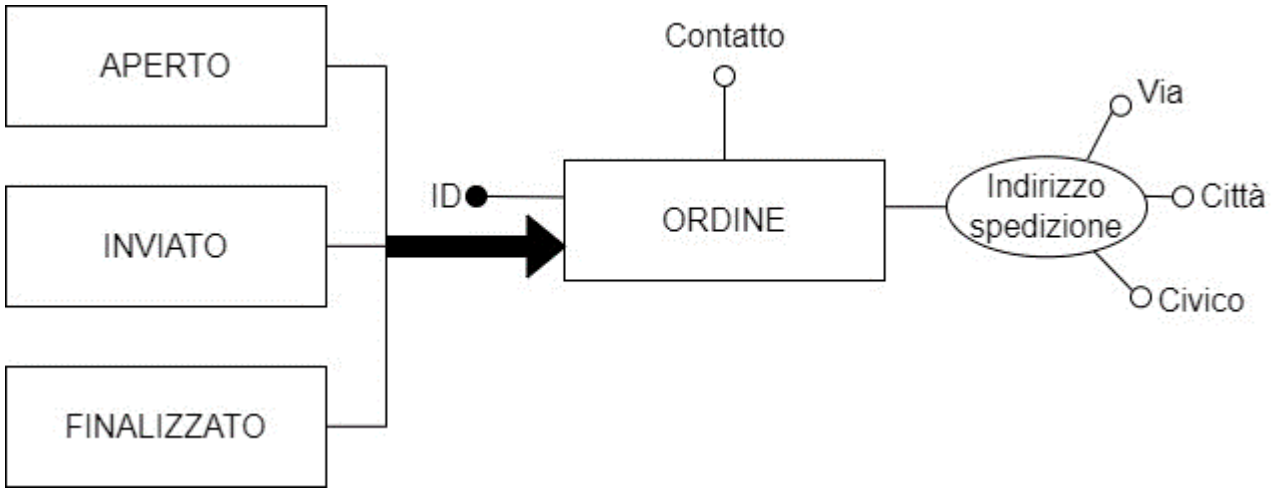
La seconda entità che ho sviluppato è quella che rappresenta i clienti che acquistano da Frutta S.r.l.
I clienti sono identificati dalla Partita IVA e hanno come attributi il nome, l'indirizzo di residenza, l'indirizzo di fatturazione (che può essere differente rispetto a quello di residenza). Inoltre, ho introdotto l'attributo multi valore contatto in quanto un cliente può avere un numero arbitrario di contatti col vincolo che ne abbia almeno uno.



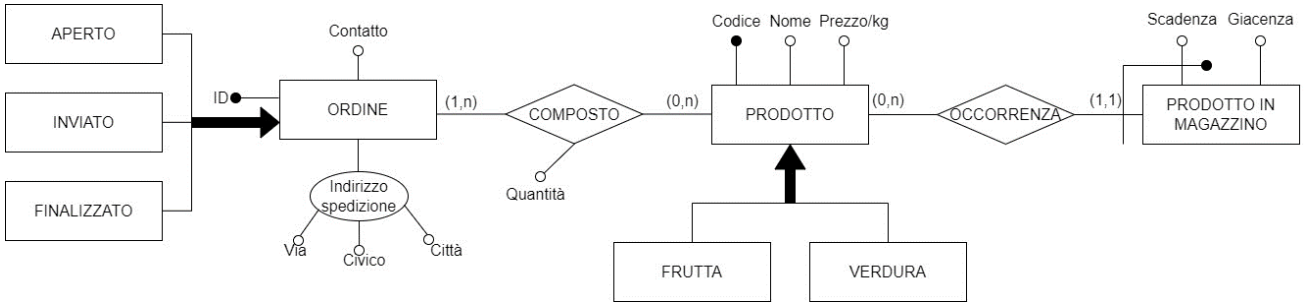
La terza entità che ho deciso di sviluppare è quella che rappresenta il fornitore. Questo viene identificato da un codice univoco, mentre come attributi ha nome, codice fiscale e indirizzo delle sedi (quest'ultimo è multi valore).



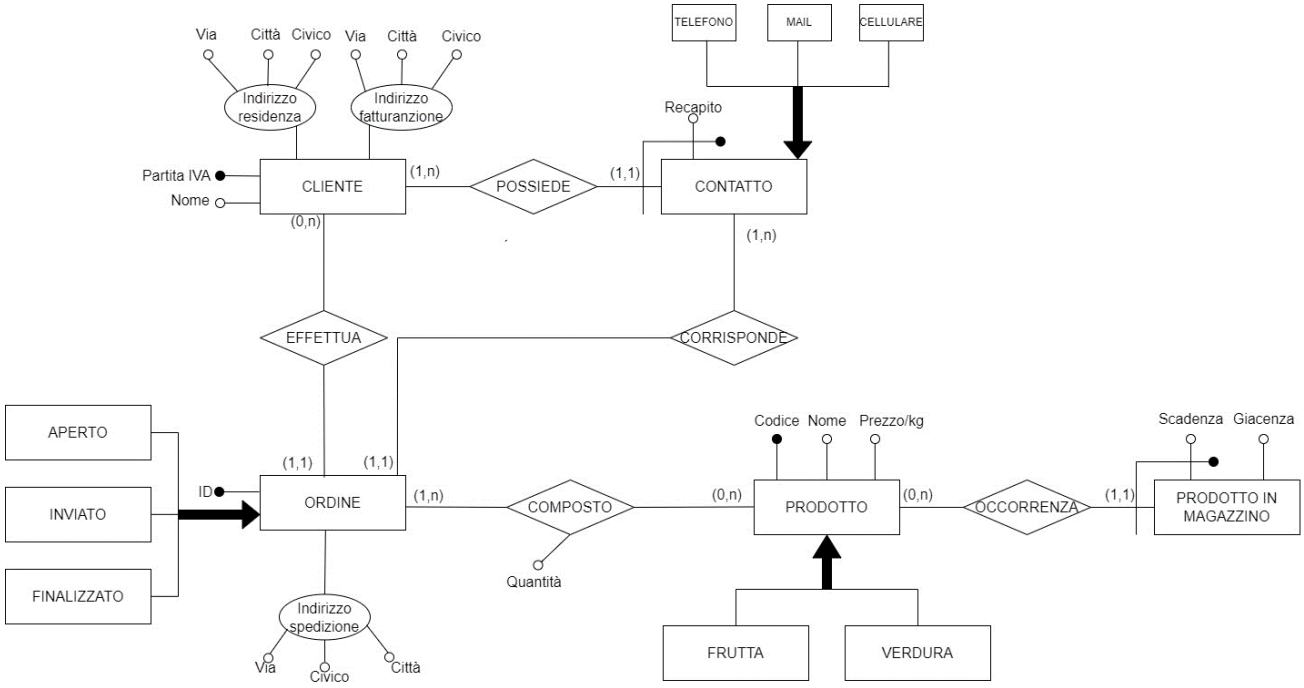
L'ultima entità è quella che rappresenta un ordine. Questo è identificato da un ID univoco e come attributi possiede l'indirizzo di spedizione e un contatto che deve essere fornito. Ho inoltre introdotto una generalizzazione per descrivere i diversi stati di un ordine.



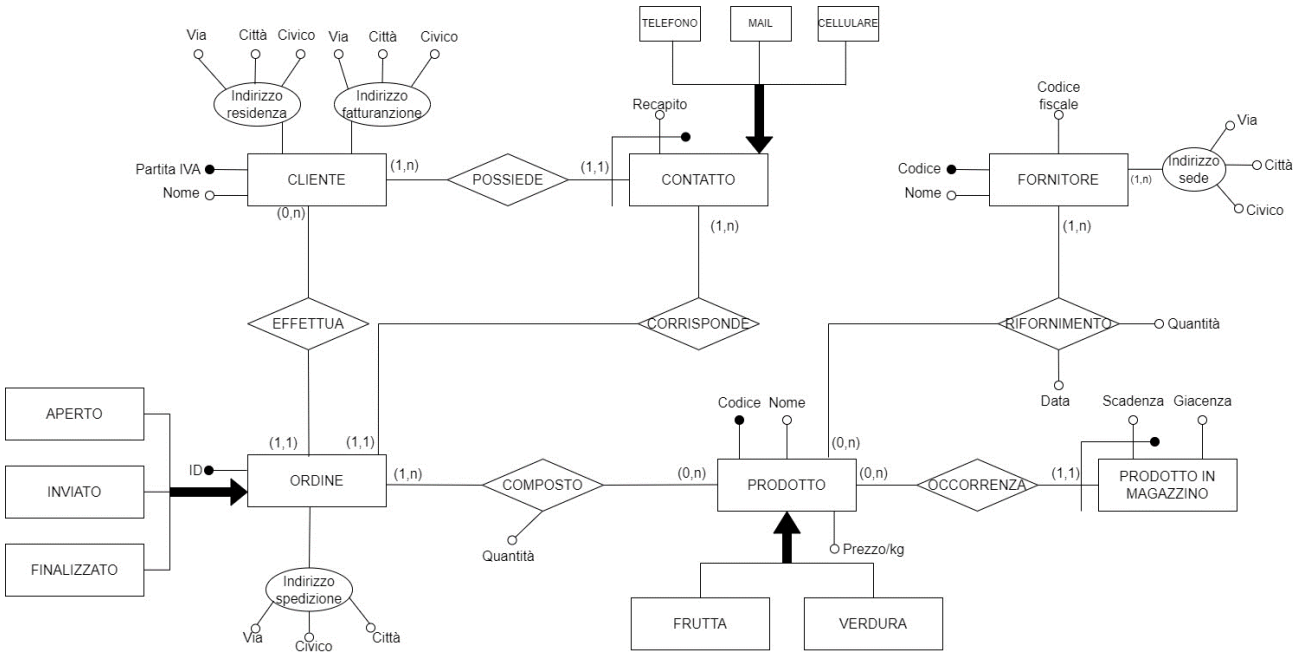
Inoltre, in un ordine devono figurare quali prodotti sono stati ordinati ed in quali quantità.



Ho deciso poi di reificare l'attributo contatto



Successivamente ho legato l'entità "Prodotto" e "Fornitore" con una relazione.

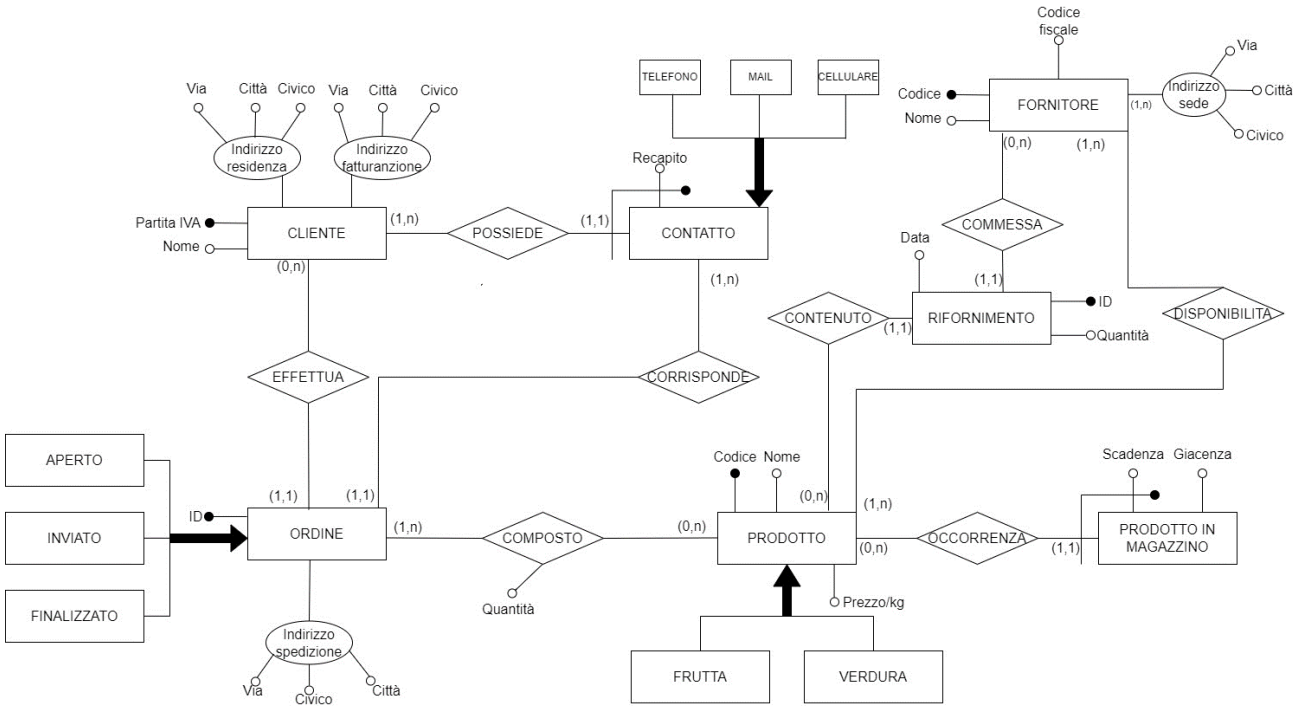


Infine, ho deciso di reificare la relazione "Rifornimento" per fare in modo che un prodotto potesse essere fornito più volte da uno stesso fornitore.

Inoltre, ho aggiunto un'altra relazione per tener traccia dei prodotti disponibili verso un determinato fornitore.

Si arriva così all'integrazione finale.

Integrazione finale



Regole aziendali

La quantità di un determinato prodotto in un ordine deve essere non superiore alla quantità di quel determinato prodotto disponibile alla vendita.

I primi prodotti ad essere venduti sono quelli con la scadenza più vicina.

La data di scadenza di un prodotto non può essere precedente a una data passata.

Dizionario dei dati

Entità	Descrizione	Attributi	Identificatori
Cliente	Le informazioni sul cliente che effettua un ordine	Nome, Indirizzo residenza, Indirizzo fatturazione	Partita IVA
Contatto	Informazioni sul contatto fornito dal cliente		Recapito, Cliente(entità)
Telefono	Tipo di contatto		Contatto, Cliente(entità)
Mail	Tipo di contatto		Contatto, Cliente(entità)
Cellulare	Tipo di contatto		Contatto, Cliente(entità)
Ordine	Informazioni sull'insieme di prodotti ordinati dal cliente	Indirizzo spedizione	ID

Aperto	Stato dell'ordine		Ordine
Inviato	Stato dell'ordine		Ordine
Finalizzato	Stato dell'ordine		Ordine
Fornitore	Informazioni su chi fornisce prodotti all'azienda	Codice fiscale, Indirizzo sede, Nome	Codice
Prodotto	Informazione sul prodotto che l'azienda fornisce	Prezzo/kg, Nome	Codice
Frutta	Tipologia di prodotto fornito		Prodotto
Verdura	Tipologia di prodotto fornito		Prodotto
Prodotto in magazzino	Informazione su prodotto in magazzino	Giacenza	Scadenza, Prodotto(entità)
Rifornimento	Ordine che Frutta S.r.l. fa a un fornitore	Quantità	ID, Data

4. Progettazione logica

Volume dei dati

Nell' analisi, si ipotizza che il sistema mantenga i dati relativi alle entità per un periodo dieci anni. Ipotizzando che i clienti nel corso di 10 anni siano 500 e che ogni cliente fornisca un numero di telefono, una mail e un numero di cellulare, la cardinalità di "Contatto" sarà 1500. Ipotizzando che mediamente ogni anno Frutta S.r.l abbia 200 clienti che effettuano ordini e ogni cliente effettui un ordine con cadenza di 3 giorni la cardinalità di "Ordine" sarà 240.000. In un ordine si ipotizza una media di 8 prodotti. Ipotizzando che, ogni mese, siano disponibili circa la metà dei prodotti a listino e che Frutta S.r.l. faccia ordini ai fornitori 1 volta a settimana per ogni prodotto in quel periodo disponibile e che, in memoria, non si mantengano rifornimenti più vecchi di un mese la cardinalità di "Rifornimento" sarà 400.

Concetto nello schema	Tipo[1]	Volume atteso
Cliente	E	500
Contatto	E	1500
Telefono	E	500
Mail	E	500
Cellulare	E	200
Ordine	E	240.000
Aperto	E	400
Inviato	E	600
Finalizzato	E	239.000
Fornitore	E	50
Prodotto	E	160
Frutta	E	80
Verdura	E	80
Prodotto in magazzino	E	320
Rifornimento	E	400
Effettua	R	240.000
Possiede	R	600
Corrisponde	R	240.000
Composto	R	2.000.000
Disponibilità	R	3200
Occorrenza	R	320
Commessa	R	400
Contenuto	R	400

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

Tavola delle operazioni

Cod.	Descrizione	Frequenza attesa
OP1	Visualizza prodotti disponibili	140/giorno
OP2	Apri ordine	70/giorno
OP3	Aggiungi prodotto all'ordine	560/giorno
OP4	Rimuovi prodotto dall'ordine	100/giorno
OP5	Invia ordine	70/giorno
OP6	Report ordine	70/giorno
OP7	Registra cliente	30/anno
OP8	Aggiungi contatto	60/anno
OP9	Visualizza dettagli cliente	50/giorno
M1	Modifica prezzo/kg di un prodotto	80/mese
M2	Aggiungi prodotto al listino	5/anno
M3	Registra nuovo fornitore	2/anno
M4	Aggiungi indirizzo fornitore	2/anno
M5	Aggiungi prodotto del fornitore	40/anno
MAG1	Visualizza ordini da spedire	100/giorno
MAG2	Mostra prodotti in un ordine	70/giorno
MAG3	Finalizza ordine	70/giorno
MAG4	Effettua richiesta al fornitore	400/mese
MAG5	Report prodotti in scadenza	5/mese
MAG6	Registra rifornimento	400/mese
MAG7	Visualizza fornitori che hanno prodotto	100/mese
MAG8	Mostra richieste ai fornitori della settimana	10/settimana
L1	Login	250/giorno

Costo delle operazioni

OP1 - Visualizza prodotti disponibili

Concetto	Costrutto	Accessi	Tipo
Frutta	E	80	L
Verdura	E	80	L

Devo accedere in lettura a tutte le occorrenze delle entità figlie di "Prodotto".

Frequenza 140/giorno ⇒22400 accessi/giorno

OP2 - Apri ordine

Concetto	Costrutto	Accessi	Tipo
Aperto	E	1	S

Composto	R	1	S
Corrisponde	R	1	S
Prodotto	E	1	S

Per far in modo che la complessità computazionale risultasse minore ho deciso di aggiungere un attributo in prodotto chiamato "quantità" dato dalla somma delle quantità di quel prodotto in magazzino.
Devo andare a scrivere una nuova occorrenza nell'entità figlia di "Ordine", successivamente devo andare ad aggiungere una nuova occorrenza in "Composto" per andare a indicare il primo prodotto dell'ordine, poi aggiungo una coppia nella relazione "Corrisponde", infine vado a modificare l'attributo "quantità" di "Prodotto".
Frequenza 70/giorno ⇒560 accessi/giorno

OP3 - Aggiungi prodotto all'ordine

Concetto	Costrutto	Accessi	Tipo
Composto	R	1	S
Prodotto	E	1	S

Vado ad aggiungere una nuova occorrenza in "Composto" e vado a modificare l'attributo "quantità" in "Prodotto".
Frequenza 560/giorno ⇒2240 accessi/giorno

OP4 - Rimuovi prodotto dall'ordine

Concetto	Costrutto	Accessi	Tipo
Composto	R	1	S
Prodotto	E	1	S

Considero il caso in cui il prodotto da rimuovere non sia l'unico prodotto nell'ordine. In questo caso vado a eliminare un'occorrenza di "Composto" e vado a modificare l'attributo "quantità" in "Prodotto".
Nel caso in cui il prodotto da rimuovere fosse l'unico dell'ordine dovrei andare a cancellare tutto ciò che è stato scritto in OP2 e andare a modificare l'attributo "quantità" in "Prodotto".
Frequenza 100/giorno ⇒400 accessi/giorno

OP5 - Invia ordine

Concetto	Costrutto	Accessi	Tipo
Ordine	E	1	S
Composto	R	8	L
Prodotto	E	8	L
Occorrenza	R	16	L
Prodotto in magazzino	E	8	S

Per semplicità computazionale i calcoli sono eseguiti considerando la generalizzazione totale sostituita da un attributo "stato" nell'entità "Ordine" come sarà poi nello schema ristrutturato.
Per inviare un ordine devo leggere prima l'id, in media un ordine contiene 8 prodotti, quindi, accedo in lettura ai prodotti dell'ordine, in media un prodotto ha 2 prodotti corrispondenti in magazzino, vado quindi a leggere le occorrenze e modificare l'attributo "giacenza" dell'entità "Prodotto in magazzino" che ha la data di scadenza più bassa.
Nel caso in cui servissero più occorrenze dello stesso prodotto si dovrebbe considerare anche il costo di eliminazione di alcune occorrenze di "Occorrenza" e "Prodotto in magazzino".
Frequenza 70/giorno ⇒3500 accessi/giorno

OP6 – Report ordine

Concetto	Costrutto	Accessi	Tipo
Prodotto	E	8	L
Composto	R	8	L

Vado a leggere le occorrenze di "Composto" e "Prodotto" relative all'ordine.
Frequenza 70/giorno ⇒1120 accessi/giorno

OP7 - Registra cliente

Concetto	Costrutto	Accessi	Tipo
Cliente	E	1	S
Contatto	E	1	S
Possiede	R	1	S

Devo andare a scrivere una nuova occorrenza di “Cliente”, successivamente una nuova occorrenza di “Contatto” e infine una coppia in “Possiede”.
Frequenza 30/anno ⇒180 accessi/anno

OP8 - Aggiungi contatto

Concetto	Costrutto	Accessi	Tipo
Contatto	E	1	S
Possiede	R	1	S

Devo andare a scrivere una nuova occorrenza di “Contatto” e, successivamente, una coppia in “Possiede”.
Frequenza 60/anno ⇒240 accessi/anno

OP9 – Visualizza dettagli cliente

Concetto	Costrutto	Accessi	Tipo
Cliente	E	1	L
Possiede	R	3	L
Contatto	E	3	L

Devo andare a leggere l’entità “Cliente”, poi, supponendo che in media un cliente abbia tre contatti, vado a leggere la relazione “Possiede” e l’entità “Contatto”
Frequenza 50/giorno ⇒350 accessi/mese

M1 - Modifica prezzo/kg di un prodotto

Concetto	Costrutto	Accessi	Tipo
Prodotto	E	1	S

Devo modificare l’attributo “Prezzo/kg” nell’entità “Prodotto”.
Frequenza 80/mese ⇒160 accessi/mese

M2 - Aggiungi prodotto a listino

Concetto	Costrutto	Accessi	Tipo
Prodotto	E	1	S

Vado a scrivere una nuova occorrenza di “Prodotto”.

Frequenza 5/anno ⇒10 accessi/anno

M3 - Registra nuovo fornitore

Concetto	Costrutto	Accessi	Tipo
Fornitore	E	1	S

Vado a scrivere una nuova occorrenza di “Fornitore”.

Frequenza 2/anno ⇒4 accessi anno

M4 - Aggiungi indirizzo fornitore

Concetto	Costrutto	Accessi	Tipo
Fornitore	E	1	S

Vado ad aggiungere un valore all’attributo “Indirizzo sede”.

Frequenza 2/anno ⇒4 accessi anno

M5 - Aggiungi prodotto fornitore

Concetto	Costrutto	Accessi	Tipo
Disponibilità	R	1	S

Vado ad aggiungere una nuova coppia nella relazione “Disponibilità”.

Frequenza 40/anno ⇒80 accessi anno

MAG1 - Visualizza lista ordini

Concetto	Costrutto	Accessi	Tipo
Inviato	E	70	L

Vado a leggere le occorrenze dell'entità "Inviato".

Frequenza 100/giorno ⇒7000 accessi/giorno

MAG2 - Mostra prodotti in un ordine

Concetto	Costrutto	Accessi	Tipo
Composto	R	8	L
Prodotto	E	8	L

Vado a leggere le occorrenze di "Composto" e "Prodotto" relative all'ordine.

Frequenza 70/giorno ⇒1120 accessi/giorno

MAG3 - Finalizza ordine

Concetto	Costrutto	Accessi	Tipo
Ordine	E	1	S

Per semplicità computazionale i calcoli sono eseguiti considerando la generalizzazione totale sostituita da un attributo "stato" nell'entità "Ordine" come sarà poi nello schema ristrutturato.

Per finalizzare un ordine devo modificare l'attributo "stato".

Frequenza 70/giorno ⇒140 accessi/giorno

MAG4 - Effettua richiesta al fornitore

Concetto	Costrutto	Accessi	Tipo
Rifornimento	E	1	S
Commessa	R	1	S
Contenuto	R	1	S

Vado a scrivere una nuova occorrenza nell'entità "Rifornimento" e nelle due relazioni "Commessa" e "Contenuto".

Frequenza 400/mese ⇒2400 accessi/mese

MAG5 - Report prodotti in scadenza

Concetto	Costrutto	Accessi	Tipo
Prodotto	E	80	L
Occorrenza	E	160	L
Prodotto in magazzino	E	160	L

In media un prodotto ha due corrispondenti prodotti in magazzino e in un determinato periodo l'azienda ha disponibili la metà dei suoi prodotti.
Si accede quindi in lettura alle occorrenze di "Prodotto in magazzino" e poi alle relative occorrenze in "prodotto".
Frequenza 5/mese ⇒2000 accessi/mese

MAG6 – Registra rifornimento

Concetto	Costrutto	Accessi	Tipo
Prodotto in magazzino	E	1	S
Occorrenza	R	1	S
Prodotto	E	1	S

Vado a modificare l'attributo "giacenza" dell'entità "Prodotto in magazzino" oppure ad aggiungere una nuova occorrenza.
Nel caso di aggiunta di una nuova occorrenza vado a scrivere una nuova occorrenza in "Occorrenza".
Infine, vado a modificare l'attributo "quantità" dell'entità Prodotto.
Frequenza 400/mese ⇒2400 accessi/mese

MAG7 – Visualizza fornitori che hanno prodotto

Concetto	Costrutto	Accessi	Tipo
Disponibilità	R	20	L
Fornitore	R	20	L

Vado a leggere le occorrenze di "Disponibilità" e "Fornitore" relative al determinato prodotto.
Frequenza 100 volte/mese ⇒4.100 accessi/mese

MAG8 – Mostra richieste ai fornitori della settimana

Concetto	Costrutto	Accessi	Tipo
Rifornimento	E	100	L
Commessa	R	100	L
Contenuto	R	100	L
Prodotto	E	100	L
Fornitore	E	100	L

Vado ad accedere in lettura alle occorrenze di “Rifornimento” con l’attributo “data” non inferiore a sette giorni rispetto al momento dell’invocazione dell’operazione. Successivamente accedo in lettura alle relazioni “Commessa” e “Contenuto” e alle entità “Prodotto” e “Fornitore”.
Frequenza 10 volte/settimana ⇒5000 accessi/settimana

Ristrutturazione dello schema E-R

1) Analisi delle ridondanze

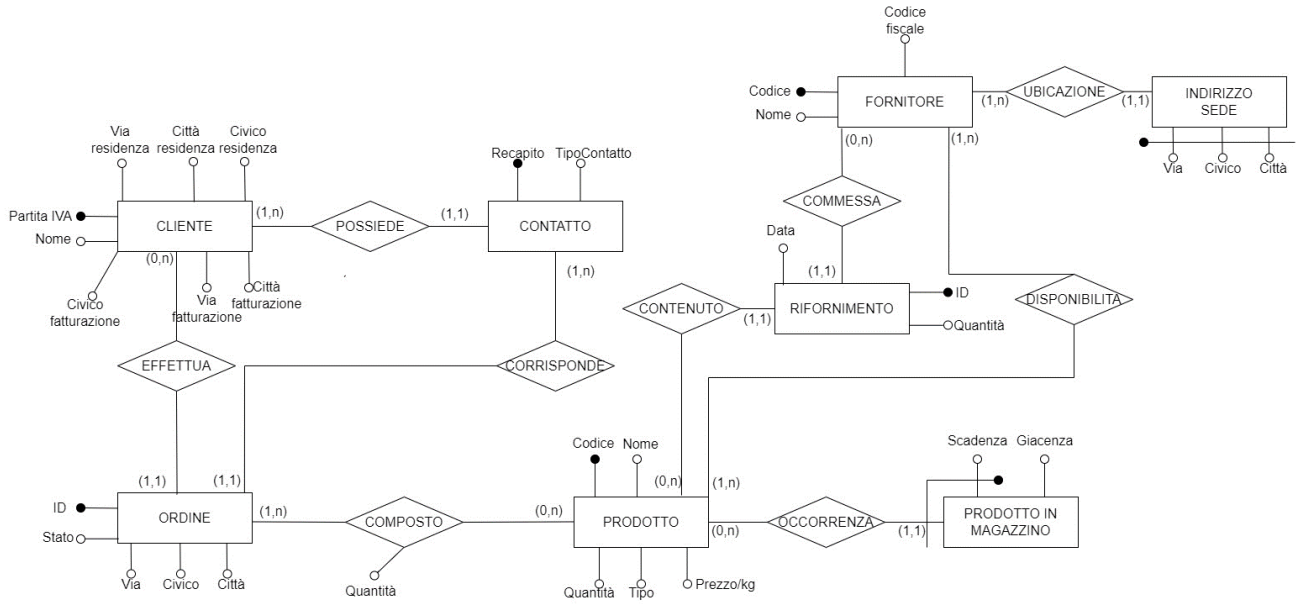
Nello schema ristrutturato ho deciso di inserire l’attributo “Quantità” nell’entità “Prodotto” l’attributo può essere derivato in quanto è dato dalla somma della giacenza di tutti quanti i prodotti di quel tipo nel magazzino. Ho deciso di inserire questo attributo per risparmiare costo computazionale.

2) Eliminazione delle generalizzazioni

Per quanto riguarda le generalizzazioni di "Prodotto” ho deciso di accorpare nell’ entità padre entrambe le entità figlie attraverso l’attributo “Tipo” in quanto non esistono relazioni che le legano ad altre entità. Stesso discorso vale per le generalizzazioni di “Contatto” che vengono accorpate al padre attraverso l’attributo “TipoContatto” e quelle di “Ordine” sostituite dall’attributo “Stato”.

Trasformazione di attributi e identificatori

Nello schema è presente un attributo multivalore, ovvero “Indirizzo sede” che indica le diverse sedi che un fornitore può avere. In questo caso ho deciso di creare un'altra entità chiamata “Indirizzo Sede”, questa nuova entità ha come identificatore tutti gli attributi componenti di “indirizzo sede”.
Per quanto riguarda gli attributi composti li ho semplicemente scomposti nelle componenti.



Traduzione di entità e associazioni

CLIENTE(PartitaIVA, Nome, ViaResidenza, CittàResidenza, CivicoResidenza, ViaFatturazione, CittàFatturazione, CivicoFatturazione)

CONTATTO(Recapito, Cliente, TipoContatto)

ORDINE(ID, Stato, Via, Città, Civico, Cliente, Contatto)

COMPOSTO(Ordine, Prodotto, Quantità)

PRODOTTO(Codice, Nome, Quantità, Tipo, Prezzo/kg)

PRODOTTO-IN-MAGAZZINO(Prodotto, Scadenza, Giacenza)

RIFORNIMENTO(ID, Prodotto, Fornitore, Quantità, Data)

FORNITORE(Codice, Nome, CodiceFiscale)

INDIRIZZO-SEDE(Città, Via, Civico, Fornitore)

DISPONIBILITA(Prodotto, Fornitore)

CONTATTO(Cliente) ⊆ CLIENTE(PartitaIVA)

ORDINE(Cliente) ⊆ CLIENTE(PartitaIVA)

ORDINE(Contatto) ⊆ CONTATTO(Recapito)

COMPOSTO(Ordine) ⊆ ORDINE(ID)

COMPOSTO(Prodotto) ⊆ PRODOTTO(Codice)

PRODOTTO-IN-MAGAZZINO(Prodotto) ⊆ PRODOTTO(Codice)

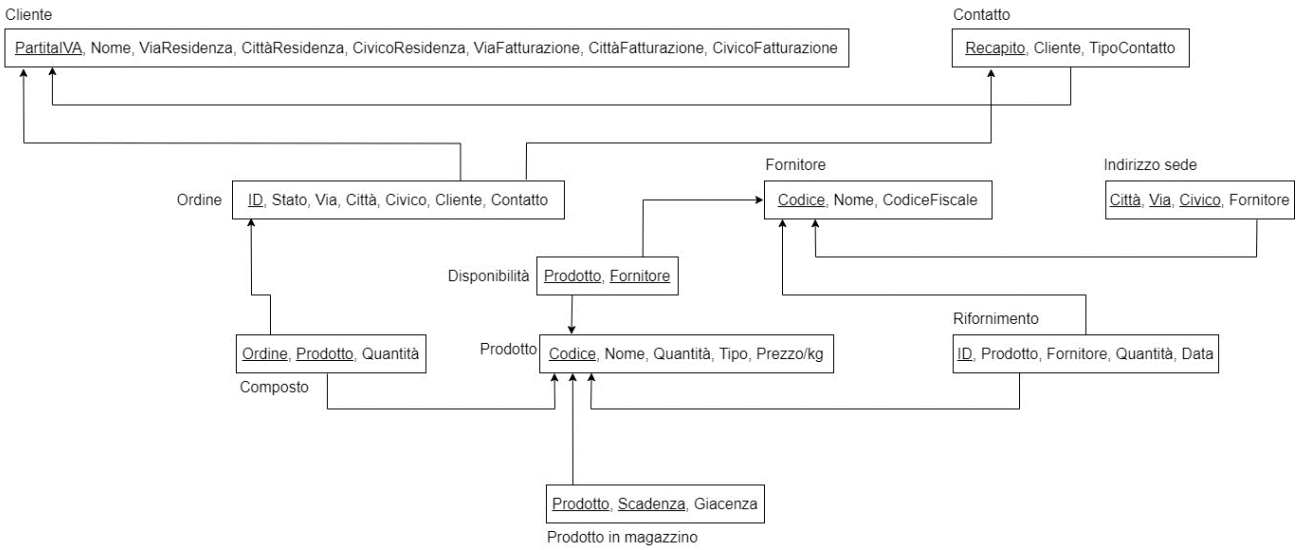
RIFORNIMENTO(Prodotto) ⊆ PRODOTTO(Codice)

RIFORNIMENTO(Fornitore) ⊆ FORNITORE(Codice)

INDIRIZZO-SEDE(Fornitore) ⊆ FORNITORE(Codice)

DISPONIBILITA(Prodotto) ⊆ PRODOTTO(Codice)

DISPONIBILITA(Fornitore) ⊆ FORNITORE(Codice)



Normalizzazione del modello relazionale

Una relazione r è in forma normale di Boyce e Codd se per ogni dipendenza funzionale (non banale) X→A definita su di essa, X contiene una chiave K di r, cioè X è una superchiave per r.
Iniziamo a verificare se le relazioni sono in BCNF.

-La relazione CLIENTE ha solo dipendenze banali in quanto, per esempio, dal nome non posso dedurre altri attributi in quanto due clienti potrebbero essere omonimi. Non posso neanche supporre che l'insieme degli attributi Nome, ViaResidenza, CittàResidenza, CivicoResidenza mi permetta di identificare gli altri attributi in quanto due clienti omonimi potrebbero abitare nella stessa palazzina.

-La relazione CONTATTO ha solo dipendenze banali. Da "Cliente" non posso risalire a "Recapito" in quanto un cliente potrebbe avere più recapiti.

-La relazione ORDINE non è in BCNF in quanto esiste una relazione che non contiene al primo membro una chiave per la relazione. Nella fattispecie la relazione è Contatto→Cliente. Si decide di separare questi concetti e, in quanto esiste già una relazione CONTATTO, che lega questi due attributi, si decide di eliminare l'attributo cliente. La relazione si decompone senza perdite in quanto

l'intersezione tra gli attributi di ORDINE e CONTATTO costituisce una chiave per CONTATTO, inoltre si conservano tutte le dipendenze.

-La relazione FORNITORE è in BCNF in quanto, nonostante esista la dipendenza CodiceFiscale→Nome, CodiceFiscale è una chiave per la relazione.

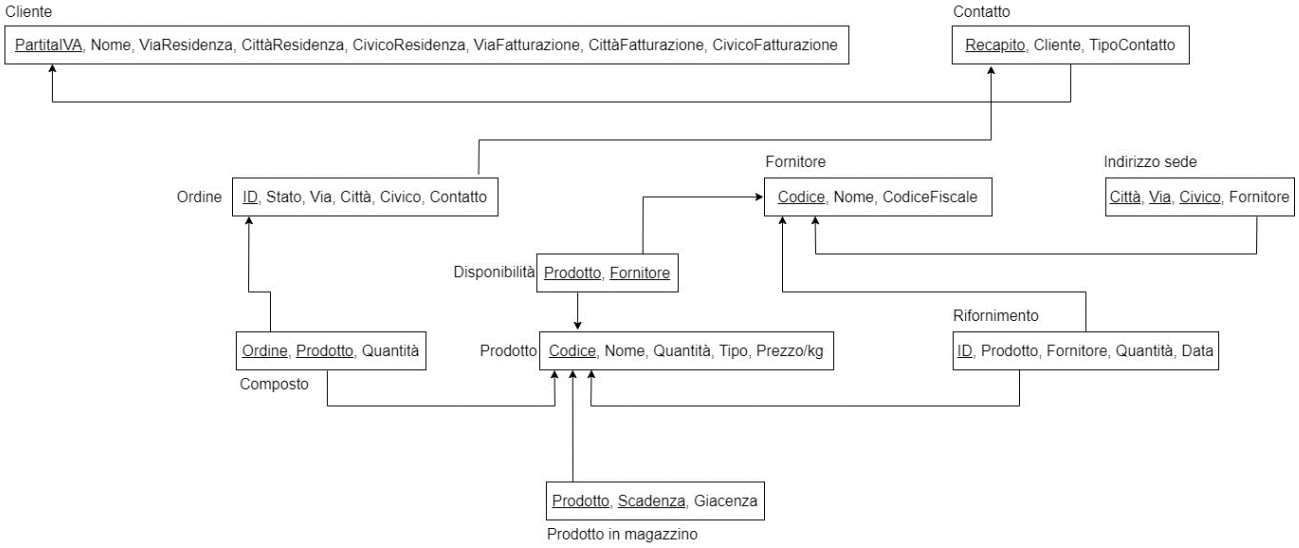
-La relazione INDIRIZZO-SEDE ha solo dipendenze banali.

-La relazione COMPOSTO ha solo dipendenze banali.

-La relazione PRODOTTO è in BCNF in quanto oltre alle dipendenze banale ha la dipendenza Nome→Codice, Quantità, Tipo, Prezzo/kg e Nome è chiave per la relazione.

-La relazione PRODOTTO-IN-MAGAZZINO ha solo dipendenze banali.

-La relazione RIFORNIMENTO ha solo dipendenze banali.



5. Progettazione fisica

Utenti e privilegi

```
SET SQL_MODE = '';
DROP USER IF EXISTS login;
SET SQL_MODE='ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZERO_DATE,ERROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION';
CREATE USER 'login' IDENTIFIED BY 'login';
GRANT EXECUTE ON PROCEDURE `fruttasrldb`.`login` TO login;


SET SQL_MODE = '';
DROP USER IF EXISTS operatore;
SET SQL_MODE='ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZERO_DATE,ERROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION';
CREATE USER 'operatore' IDENTIFIED BY 'operatore';
GRANT EXECUTE ON PROCEDURE `fruttasrldb`.`visualizza_prodotti_disponibili` to operatore;
GRANT EXECUTE ON PROCEDURE `fruttasrldb`.`apri_ordine` to operatore;
GRANT EXECUTE ON PROCEDURE `fruttasrldb`.`aggiungi_prodotto_a_ordine` to operatore;
GRANT EXECUTE ON PROCEDURE `fruttasrldb`.`rimuovi_prodotto_a_ordine` to operatore;
GRANT EXECUTE ON PROCEDURE `fruttasrldb`.`invia_ordine` to operatore;
GRANT EXECUTE ON PROCEDURE `fruttasrldb`.`report_ordine` to operatore;
GRANT EXECUTE ON PROCEDURE `fruttasrldb`.`registra_cliente` to operatore;
GRANT EXECUTE ON PROCEDURE `fruttasrldb`.`aggiungi_contatto` to operatore;
GRANT EXECUTE ON PROCEDURE `fruttasrldb`.`visualizza dettagli cliente` to operatore;

SET SQL_MODE = '';
DROP USER IF EXISTS manager;
SET SQL_MODE='ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZERO_DATE,ERROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION';
CREATE USER 'manager' IDENTIFIED BY 'manager';
GRANT EXECUTE ON PROCEDURE `fruttasrldb`.`modifica_prezzoKg` to manager;
GRANT EXECUTE ON PROCEDURE `fruttasrldb`.`aggiungi_prodotto` to manager;
GRANT EXECUTE ON PROCEDURE `fruttasrldb`.`aggiungi_fornitore` to manager;
GRANT EXECUTE ON PROCEDURE `fruttasrldb`.`aggiungi_indirizzo_fornitore` to manager;
GRANT EXECUTE ON PROCEDURE `fruttasrldb`.`aggiungi_prodotto_fornitore` to manager;

SET SQL_MODE = '';
DROP USER IF EXISTS magazziniere;
SET SQL_MODE='ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZERO_DATE,ERROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION';
CREATE USER 'magazziniere' IDENTIFIED BY 'magazziniere';
GRANT EXECUTE ON PROCEDURE `fruttasrldb`.`visualizza_ordini_da_spedire` to magazziniere;
GRANT EXECUTE ON PROCEDURE `fruttasrldb`.`mostra_prodotti_ordine` to magazziniere;
GRANT EXECUTE ON PROCEDURE `fruttasrldb`.`finalizza_ordine` to magazziniere;
GRANT EXECUTE ON PROCEDURE `fruttasrldb`.`effettua_richiesta_fornitore` to magazziniere;
GRANT EXECUTE ON PROCEDURE `fruttasrldb`.`report_prodotti_in_scadenza` to magazziniere;
GRANT EXECUTE ON PROCEDURE `fruttasrldb`.`registra_rifornimento` to magazziniere;
GRANT EXECUTE ON PROCEDURE `fruttasrldb`.`visualizza_fornitori_che_hanno_prodotto` to magazziniere;
GRANT EXECUTE ON PROCEDURE `fruttasrldb`.`mostra_richieste_ai_fornitori_settimana` to magazziniere;
```

Strutture di memorizzazione

Cliente		
Colonna	**Tipo di dato**	**Attributi**[2]
Partita IVA	CHAR(11)	PK, NN
Nome	VARCHAR(45)	NN
Via Residenza	VARCHAR(45)	NN
Città Residenza	VARCHAR(45)	NN
Civico Residenza	SMALLINT	NN, UN
Via Fatturazione	VARCHAR(45)	NN
Città Fatturazione	VARCHAR(45)	NN
Civico Fatturazione	SMALLINT	NN, UN

Contatto		
Colonna	**Tipo di dato**	**Attributi**
Recapito	VARCHAR(45)	NN, PK
Cliente	CHAR(11)	NN
Tipo Contatto	ENUM(‘Cellulare’, ’Mail’, ‘Telefono’)	NN

Ordine		
Colonna	**Tipo di dato**	**Attributi**
ID	INT	NN, PK, AI, UN
Stato	ENUM(‘Aperto’, ‘Inviato’, ‘Finalizzato’)	NN
Via	VARCHAR(45)	NN
Città	VARCHAR(45)	NN
Civico	SMALLINT	NN, UN
Contatto	VARCHAR(45)	NN

Fornitore		
Colonna	**Tipo di dato**	**Attributi**
Codice	SMALLINT	PK, NN, AI, UN
Nome	VARCHAR(45)	NN
Codice Fiscale	CHAR(16)	NN, UQ

Indirizzo sede		
Colonna	**Tipo di dato**	**Attributi**
Città	VARCHAR(45)	PK, NN
Via	VARCHAR(45)	PK, NN
Civico	SMALLINT	PK, NN, UN
Fornitore	SMALLINT	NN, UN

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

Composto		
Colonna	**Tipo di dato**	**Attributi**
Ordine	INT	PK, NN, UN
Prodotto	SMALLINT	PK, NN, UN
Quantità	DECIMAL	NN, UN

Prodotto		
Colonna	**Tipo di dato**	**Attributi**
Codice	SMALLINT	PK, NN, AI, UN
Nome	VARCHAR(45)	NN, UQ
Quantità	DECIMAL	NN, UN
Tipo	ENUM(‘Frutta’, ‘Verdura’)	NN
Prezzo/kg	DECIMAL	NN, UN

Rifornimento		
Colonna	**Tipo di dato**	**Attributi**
ID	SMALLINT	PK, NN, AI, UN
Prodotto	SMALLINT	NN, UN
Fornitore	SMALLINT	NN, UN
Quantità	DECIMAL	NN, UN
Data	DATE	NN

Prodotto in magazzino		
Colonna	**Tipo di dato**	**Attributi**
Prodotto	SMALLINT	PK, NN, UN
Scadenza	DATE	PK, NN
Giacenza	DECIMAL	NN, UN

Disponibilità		
Colonna	**Tipo di dato**	**Attributi**
Prodotto	SMALLINT	PK, NN, UN
Fornitore	SMALLINT	PK, NN, UN

Utenti		
Colonna	**Tipo di dato**	**Attributi**
Username	VARCHAR(45)	PK, NN
Password	CHAR(32)	NN
Ruolo	ENUM(‘Operatore’, ‘Manager’, ‘Magazziniere’)	NN

Indici

Non ho usato indici

Trigger

Trigger che dopo rimozione di un prodotto dall'ordine ripristina la giacenza del prodotto.

```
DELIMITER $$

USE `fruttasrldb`$$
DROP TRIGGER IF EXISTS `fruttasrldb`.`composto_AFTER_DELETE` $$
USE `fruttasrldb`$$
CREATE
DEFINER=`root`@`localhost`
TRIGGER `fruttasrldb`.`composto_AFTER_DELETE`
AFTER DELETE ON `fruttasrldb`.`composto`
FOR EACH ROW
BEGIN
    UPDATE prodotto
    SET prodotto.Quantità = prodotto.Quantità + old.Quantità
    WHERE Codice = old.Prodotto;
END$$
```

Trigger che modifica la giacenza di un prodotto dopo aggiunta a un ordine.

```
USE `fruttasrldb`$$
DROP TRIGGER IF EXISTS `fruttasrldb`.`composto_AFTER_INSERT` $$
USE `fruttasrldb`$$
CREATE
DEFINER=`root`@`localhost`
TRIGGER `fruttasrldb`.`composto_AFTER_INSERT`
AFTER INSERT ON `fruttasrldb`.`composto`
FOR EACH ROW
BEGIN
    UPDATE prodotto
    SET prodotto.Quantità = prodotto.Quantità - New.Quantità
    WHERE Codice = New.Prodotto;
END$$
```

Trigger che verifica prima che un ordine sia aperto e poi che la quantità richiesta sia inferiore alla quantità disponibile.

```
USE `fruttasrldb`$$
DROP TRIGGER IF EXISTS `fruttasrldb`.`composto_BEFORE_INSERT` $$
USE `fruttasrldb`$$
CREATE
DEFINER=`root`@`localhost`
TRIGGER `fruttasrldb`.`composto_BEFORE_INSERT`
BEFORE INSERT ON `fruttasrldb`.`composto`
FOR EACH ROW
BEGIN
    declare var_Giacenza decimal(10,2);
    declare var_Stato varchar(45);
    select Stato from ordine where ID = New.Ordine into var_Stato;
    if var_Stato <> 'Aperto' then
    SIGNAL sqlstate '45000' set message_text = "Impossibile aggiungere prodotto perchè ordine già inviato o finalizzato";
    end if;

    select Quantità from Prodotto where Codice = New.prodotto into var_Giacenza;
    if var_Giacenza < New.Quantità then
    SIGNAL sqlstate '45000' set message_text = "La richiesta è superiore alla giacenza in magazzino";
    end if;
END$$
```

Trigger che impedisce la modifica di un ordine se inviato.

```
USE `fruttasrldb`$$
DROP TRIGGER IF EXISTS `fruttasrldb`.`composto_BEFORE_DELETE`$$
USE `fruttasrldb`$$
CREATE
DEFINER = `root`@`localhost`
TRIGGER `fruttasrldb`.`composto_BEFORE_DELETE`
BEFORE DELETE ON `composto` FOR EACH ROW
BEGIN
    declare var_stato varchar(45);
    select Stato from ordine
    where old.Ordine = `ordine`.`ID`
    into var_stato;
    if var_stato <> 'Aperto' then
    SIGNAL sqlstate '45000' set message_text = "Impossibile modificare l'ordine perchè inviato o finalizzato";
    end if;
END$$
```

Trigger per aggiornare la giacenza totale di un prodotto dopo la scadenza di una parte dei prodotti in magazzino.

```
USE `fruttasrldb`$$
DROP TRIGGER IF EXISTS `fruttasrldb`.`scaduto_AFTER_DELETE` $$
USE `fruttasrldb`$$
CREATE
DEFINER=`root`@`localhost`
TRIGGER `fruttasrldb`.`scaduto_AFTER_DELETE`
AFTER DELETE ON `fruttasrldb`.`prodotto in magazzino`
FOR EACH ROW
BEGIN
    UPDATE prodotto
    SET prodotto.Quantità = prodotto.Quantità - old.`Giacenza`
    WHERE Codice = old.`Prodotto`;
END$$
```

Trigger per decrementare la quantità dei prodotti in magazzino in base a ordine inviato.

```
USE `fruttasrldb`$$
DROP TRIGGER IF EXISTS `fruttasrldb`.`ordine_AFTER_UPDATE` $$
USE `fruttasrldb`$$
CREATE
DEFINER=`root`@`localhost`
TRIGGER `ordine_AFTER_UPDATE`
AFTER UPDATE ON `ordine`
FOR EACH ROW
BEGIN

    declare done int default false;
    declare var_richiesta decimal (10,2) unsigned;
    declare var_prodotto smallint unsigned;

    declare var_prodotto_magazzino smallint unsigned;
    declare var_giacenza decimal(10,2) unsigned;
    declare var_data date;

    declare cur cursor for
        select `composto`.`Prodotto`, `composto`.`Quantità`
        from ordine join composto on `ordine`.`ID` = `composto`.`Ordine`
        where old.Stato = 'Aperto' and new.Stato = 'Inviato' ;
```

```
declare curr cursor for
    select `prodotto in magazzino`.`Prodotto`, `prodotto in magazzino`.`Scadenza`, `prodotto in magazzino`.`Giacenza`
    from `prodotto in magazzino`
    order by `prodotto in magazzino`.`Scadenza`;

declare continue handler for not found set done = true;

open cur;
read_loop: loop
    fetch cur into var_prodotto, var_richiesta;
    if done then
        leave read_loop;
    end if;

open curr;
read__loop: loop
    fetch curr into var_prodotto_magazzino, var_data, var_giacenza;
    if var_richiesta = 0 then
        leave read__loop;
    end if;

    if var_prodotto = var_prodotto_magazzino then

                if var_giacenza > var_richiesta then
                 update `prodotto in magazzino`
                 set `prodotto in magazzino`.Giacenza = var_giacenza - var_richiesta
                 where Prodotto = var_prodotto and Scadenza = var_data;

                 set var_richiesta = 0;
                 end if;

                            if var_richiesta >= var_giacenza then
                 update `prodotto in magazzino`
                 set `prodotto in magazzino`.Giacenza = 0
                 where Prodotto = var_prodotto and Scadenza = var_data;

                 delete from `prodotto in magazzino`
                 where Prodotto = var_prodotto and Scadenza = var_data;

                 set var_richiesta = var_richiesta - var_giacenza;
                 end if;
        end if;


        end loop;
        close curr;

        end loop;
        close cur;

END$$
```

Eventi

Evento giornaliero per eliminazione dei prodotti in scadenza.

```
DELIMITER $$
CREATE EVENT IF NOT EXISTS elimina_scaduti_o_indisponibili
    On schedule EVERY 1 DAY
    Do BEGIN
    DELETE from `prodotto in magazzino` where Scadenza = curdate();
    END$$

DELIMITER ;
```

Evento giornaliero per eliminazione richieste a fornitori più vecchie di un mese.

```
DELIMITER $$
CREATE EVENT IF NOT EXISTS elimina_vecchi_rifornimenti
    On schedule EVERY 1 DAY
    Do BEGIN
    DELETE from `rifornimento` where Data < (curdate()- INTERVAL 1 MONTH);
    END$$

DELIMITER ;
```

Viste

Non ho usato viste.

Stored Procedures e transazioni

OP1 – Visualizza prodotti disponibili

Ho scelto come livello di isolamento Read Committed perché non voglio dirty reads sulla quantità di prodotti disponibili.

```
DELIMITER $$
USE `fruttasrldb`$$
CREATE DEFINER=`root`@`localhost` PROCEDURE `visualizza_prodotti_disponibili`()
BEGIN
        declare exit handler for sqlexception
    begin
        rollback;
        resignal;
    end;

set transaction isolation level read committed;
set transaction read only;
start transaction;

    select *
    from prodotto
    where Quantità > 0;
    commit;
END$$
```

OP2 – Apri Ordine

Ho scelto come livello d'isolamento Repeatble Read per via dei due trigger before insert e after insert su composto. Non voglio che nessuno sia in grado di modificare la quantità disponibile di un prodotto prima della fine di quest'operazione.

```
USE `fruttasrldb`;
DROP procedure IF EXISTS `fruttasrldb`.`apri_ordine`;

DELIMITER $$
USE `fruttasrldb`$$
CREATE DEFINER=`root`@`localhost` PROCEDURE `apri_ordine`(in var_viaConsegna varchar(45), in var_cittaConsegna varchar(45), in var_civicoConsegna smallint unsigned,
                                                          in var_contatto varchar(45), in var_prodotto smallint unsigned, in var_quantita float unsigned, out var_ID smallint unsigned)
BEGIN
    declare exit handler for sqlexception
    begin
        rollback;
        resignal;
    end;

    set transaction isolation level repeatable read; /*per il trigger*/
    start transaction;
        insert into ordine(Stato, Via, Città, Civico, Contatto) values('Aperto', var_viaConsegna, var_cittaConsegna, var_civicoConsegna, var_contatto);
        select max(ID) from ordine into var_ID;
        insert into composto(Ordine, Prodotto, Quantità) values (var_ID, var_prodotto, var_quantita);
    commit;
END$$
```

OP3 – Aggiungi prodotto all'ordine

Come OP2

```
USE `fruttasrldb`;
DROP procedure IF EXISTS `fruttasrldb`.`aggiungi_prodotto_a_ordine`;

DELIMITER $$
USE `fruttasrldb`$$
CREATE DEFINER=`root`@`localhost` PROCEDURE `aggiungi_prodotto_a_ordine`(in var_IDOrdine smallint unsigned, in var_prodotto smallint unsigned, in var_quantita float unsigned)
BEGIN
    declare exit handler for sqlexception
        begin
            rollback;
            resignal;
        end;

        set transaction isolation level repeatable read; /*per il trigger*/
        start transaction;
        insert into composto(Ordine, Prodotto, Quantità) values (var_IDOrdine, var_prodotto, var_quantita);
        commit;
END$$

DELIMITER ;
```

OP4 – Rimuovi prodotto dall'ordine

Ho scelto come livello d'isolamento Repeatble Read per via del trigger after delete su composto. Non voglio che nessuno sia in grado di modificare o leggere la quantità disponibile di un prodotto prima della fine di quest'operazione.

```
USE `fruttasrldb`;
DROP procedure IF EXISTS `fruttasrldb`.`rimuovi_prodotto_a_ordine`;

DELIMITER $$
USE `fruttasrldb`$$
CREATE DEFINER=`root`@`localhost` PROCEDURE `rimuovi_prodotto_a_ordine`(in var_IDOrdine smallint unsigned, in var_prodotto smallint unsigned)
BEGIN
    declare exit handler for sqlexception
        begin
            rollback;
            resignal;
        end;

        set transaction isolation level repeatable read;
        start transaction;
        delete
        from composto
        where Ordine = var_IDOrdine and Prodotto = var_prodotto;
        commit;
END$$

DELIMITER ;
```

OP5 – Invia Ordine

Ho scelto come livello d'isolamento Serializable per via del trigger after update su Ordine in quanto all'interno del trigger faccio uso di cursori.

```
USE `fruttasrldb`;
DROP procedure IF EXISTS `fruttasrldb`.`invia_ordine`;

DELIMITER $$
USE `fruttasrldb`$$
CREATE DEFINER=`root`@`localhost` PROCEDURE `invia_ordine`(in var_IDOrdine smallint unsigned)
BEGIN
declare var varchar(45);
declare exit handler for sqlexception
        begin
            rollback;
            resignal;
        end;

        set transaction isolation level serializable; /*per il trigger*/
        start transaction;

        select Stato
        from ordine
        where ID = var_IDOrdine
        into var;
        if var <> 'Aperto' then
            signal sqlstate "45000" set message_text = "Non si può inviare un ordine non aperto";
        else

        update ordine
        set Stato = 'Inviato'
        where ID = var_IDOrdine;
        end if;
        commit;
END$$
```

OP6 – Report Ordine

Ho scelto come livello d'isolamento Read Committed perché voglio evitare letture di dati non ancora andati in commit, evitando quindi letture sporche.

```
USE `fruttasrldb`;
DROP procedure IF EXISTS `fruttasrldb`.`report_ordine`;

DELIMITER $$
USE `fruttasrldb`$$
CREATE DEFINER=`root`@`localhost` PROCEDURE `report_ordine`(in var_id smallint unsigned)
BEGIN
    declare exit handler for sqlexception
        begin
            rollback;
            resignal;
        end;
        set transaction isolation level read committed;
        start transaction;

        select `Codice`, `Nome`, `Prezzo/kg`, `composto`.`Quantità`, (`Prezzo/kg` * `composto`.`Quantità`) as `Costo`
        from prodotto join composto on `composto`.`Prodotto` = `prodotto`.`Codice`
        where `composto`.`Ordine` = var_id;

        select sum(`Prezzo/kg` * `composto`.`Quantità`) as `Costo Totale`
        from prodotto join composto on `composto`.`Prodotto` = `prodotto`.`Codice`
        where `composto`.`Ordine` = var_id;
        commit;
END$$
```

OP7 – Registra Cliente

Ho scelto come livello d'isolamento il livello più basso(Read Uncommitted) per fare rollback in caso di problemi.

```
USE `fruttasrldb`;
DROP procedure IF EXISTS `fruttasrldb`.`registra_cliente`;

DELIMITER $$
USE `fruttasrldb`$$
CREATE DEFINER=`root`@`localhost` PROCEDURE `registra_cliente`(in var_partitaIVA char(11), in var_nome varchar(45), in var_viaResidenza varchar(45),
                                                               in var_cittaResidenza varchar(45), in var_civicoResidenza smallint unsigned, in var_viaFatturazione varchar(45), in var_cittaFatturazione varchar(45),
                                                               in var_civicoFatturazione smallint unsigned, in var_contatto varchar(45), in var_tipoContatto enum('Cellulare','Mail','Telefono'))
BEGIN
    declare exit handler for sqlexception
        begin
            rollback;
            resignal;
        end;

        set transaction isolation level read uncommitted;
        start transaction;
    insert into cliente(`Partita IVA`, `Nome`, `Via Residenza`, `Città Residenza`,`Civico Residenza` , `Via Fatturazione`, `Città Fatturazione`, `Civico Fatturazione`)
    values(var_partitaIVA, var_nome, var_viaResidenza, var_cittaResidenza, var_civicoResidenza, var_viaFatturazione, var_cittaFatturazione, var_civicoFatturazione);
    insert into contatto(`Recapito`, `Cliente`, `Tipo Contatto`) values(var_contatto, var_partitaIVA, var_tipoContatto);
    commit;
END$$

DELIMITER ;
```

OP8 – Aggiungi Contatto

```
USE `fruttasrldb`;
DROP procedure IF EXISTS `fruttasrldb`.`aggiungi_contatto`;

DELIMITER $$
USE `fruttasrldb`$$
CREATE DEFINER=`root`@`localhost` PROCEDURE `aggiungi_contatto`(in var_recapito varchar(45), in var_cliente char(11), in var_tipoContatto enum('Cellulare','Mail','Telefono'))
BEGIN
	insert into contatto(`Recapito`, `Cliente`, `Tipo Contatto`) values(var_recapito, var_cliente, var_tipoContatto);
END$$

DELIMITER ;
```

OP9 – Visualizza dettagli cliente

Ho scelto come livello di isolamento Read Committed perché non vogli dirty reads su dati non ancora andati in commit.

```
USE `fruttasrldb`;
DROP procedure IF EXISTS `fruttasrldb`.`visualizza_dettagli_cliente`;

DELIMITER $$
USE `fruttasrldb`$$
CREATE DEFINER=`root`@`localhost` PROCEDURE `visualizza_dettagli_cliente`(in var_partitaIVA char(11))
BEGIN
    declare exit handler for sqlexception
    begin
        rollback;
        resignal;
    end;

    set transaction isolation level read committed;
    start transaction;
    select *
    from cliente
    where `Partita IVA` = var_partitaIVA;

    select `Recapito`, `Tipo Contatto`
    from contatto
    where Cliente = var_partitaIVA;

    commit;

END$$
```

M1 – Modifica prezzo/kg di un prodotto

```
USE `fruttasrldb`;
DROP procedure IF EXISTS `fruttasrldb`.`modifica_prezzoKg`;

DELIMITER $$
USE `fruttasrldb`$$
CREATE DEFINER=`root`@`localhost` PROCEDURE `modifica_prezzoKg`(in var_IDProdotto smallint unsigned, in var_nuovoPrezzo float unsigned)
BEGIN
    update prodotto
    set `Prezzo/kg` = var_nuovoPrezzo
    where `Codice` = var_IDProdotto;

END$$

DELIMITER ;
```

M2 – Aggiungi prodotto al listino

```
USE `fruttasrldb`;
DROP procedure IF EXISTS `fruttasrldb`.`aggiungi_prodotto`;

DELIMITER $$
USE `fruttasrldb`$$
CREATE DEFINER=`root`@`localhost` PROCEDURE `aggiungi_prodotto`(in var_nome varchar(45), in var_tipo enum('Frutta','Verdura'), in var_prezzoKg float unsigned)
BEGIN

        insert into prodotto(`Nome`, `Quantità`, `Tipo`, `Prezzo/kg`)
        values(var_nome, 0, var_tipo, var_prezzoKg);
END$$

DELIMITER ;
```

M3 – Registra nuovo fornitore

Ho scelto come livello d'isolamento Read Committed perché voglio evitare letture sporche sul codice del fornitore.

```
USE `fruttasrldb`;
DROP procedure IF EXISTS `fruttasrldb`.`aggiungi_fornitore`;

DELIMITER $$
USE `fruttasrldb`$$
CREATE DEFINER=`root`@`localhost` PROCEDURE `aggiungi_fornitore`(in var_nome varchar(45), in var_CF char(16), in var_citta varchar(45), in var_via varchar(45), in var_civico smallint unsigned, out var_codice smallint unsigned)
BEGIN
    declare exit handler for sqlexception
    begin
        rollback;
        resignal;
    end;

    set transaction isolation level read committed;

    start transaction;
        insert into fornitore(`Nome`, `Codice Fiscale`)
        values(var_nome, var_CF);
        select max(Codice) from fornitore into var_codice;
        insert into `indirizzo sede`(Città, Via, Civico, Fornitore) values(var_citta, var_via, var_civico, var_codice);
    commit;
END$$

DELIMITER ;
```

M4 – Aggiungi indirizzo fornitore

```
USE `fruttasrldb`;
DROP procedure IF EXISTS `fruttasrldb`.`aggiungi_indirizzo_fornitore`;

DELIMITER $$
USE `fruttasrldb`$$
CREATE DEFINER=`root`@`localhost` PROCEDURE `aggiungi_indirizzo_fornitore`(in var_citta varchar(45), in var_via varchar(45), in var_civico smallint unsigned, in var_codice smallint unsigned)
BEGIN
        insert into `indirizzo sede`(Città, Via, Civico, Fornitore) values(var_citta, var_via, var_civico, var_codice);
END$$

DELIMITER ;
```

M5 – Aggiungi prodotto a fornitore

```
USE `fruttasrldb`;
DROP procedure IF EXISTS `fruttasrldb`.`aggiungi_prodotto_fornitore`;

DELIMITER $$
USE `fruttasrldb`$$
CREATE DEFINER=`root`@`localhost` PROCEDURE `aggiungi_prodotto_fornitore`(in var_IDFornitore smallint unsigned, in var_IDProdotto smallint unsigned)
BEGIN
    insert into `disponibilità`(Fornitore, Prodotto) values(var_IDFornitore, var_IDProdotto);
END$$

DELIMITER ;
```

MAG1 – Visualizza ordini da spedire

Non ho usato transazioni perché un ordine inviato non può essere modificato.

```
USE `fruttasrldb`;
DROP procedure IF EXISTS `fruttasrldb`.`visualizza_ordini_da_spedire`;

DELIMITER $$
USE `fruttasrldb`$$
CREATE DEFINER=`root`@`localhost` PROCEDURE `visualizza_ordini_da_spedire`()
BEGIN

    select *
    from ordine
    where Stato = 'Inviato';

END$$

DELIMITER ;
```

MAG2 – Mostra prodotti in un ordine

```
USE `fruttasrldb`;
DROP procedure IF EXISTS `fruttasrldb`.`mostra_prodotti_ordine`;

DELIMITER $$
USE `fruttasrldb`$$
CREATE DEFINER=`root`@`localhost` PROCEDURE `mostra_prodotti_ordine`(in var_IDOrdine smallint unsigned)
BEGIN
    Select Codice, Nome, `composto`.`Quantità` as QuantitaOrdinata
    from prodotto join composto on `prodotto`.`Codice` = `composto`.`Prodotto`
    join ordine on `composto`.`Ordine` = `ordine`.`ID`
    where `ordine`.`Stato` = 'Inviato' and `composto`.`Ordine` = var_IDOrdine;
END$$

DELIMITER ;
```

MAG3 – Finalizza ordine

```
USE `fruttasrldb`;
DROP procedure IF EXISTS `fruttasrldb`.`finalizza_ordine`;

DELIMITER $$
USE `fruttasrldb`$$
CREATE DEFINER=`root`@`localhost` PROCEDURE `finalizza_ordine`(in var_IDOrdine smallint unsigned)
BEGIN
        declare var varchar(45);
        select Stato
        from ordine
        where ID = var_IDOrdine
        into var;
        if var <> 'Inviato' then
            signal sqlstate "45000" set message_text = "Non si può finalizzare un ordine non inviato";
        else
        update ordine
        set Stato = 'Finalizzato'
        where ID = var_IDOrdine;
        end if;
END$$

DELIMITER ;
```

MAG4 – Effettua richiesta a fornitore

Ho scelto come livello d'isolamento Read Committed per evitare problemi sul valore ID(Auto increment) di 'rifornimento'.

```
USE `fruttasrldb`;
DROP procedure IF EXISTS `fruttasrldb`.`effettua_richiesta_fornitore`;

DELIMITER $$
USE `fruttasrldb`$$
CREATE DEFINER=`root`@`localhost` PROCEDURE `effettua_richiesta_fornitore`(in var_Fornitore smallint unsigned, in var_Prodotto smallint unsigned, in var_Quantita float unsigned)
BEGIN
    declare var int;
    declare exit handler for sqlexception
    begin
        rollback;
        resignal;
    end;
    set transaction isolation level read committed;
    start transaction;
        select count(*)
        from disponibilità
        where Prodotto = var_Prodotto and Fornitore = var_Fornitore
        into var;

        if var = 0 then
            signal sqlstate "45000" set message_text = "Il fornitore non dispone di quel prodotto";
        else
            insert into rifornimento(Prodotto, Fornitore, Quantità, Data) values(var_Prodotto, var_fornitore, var_Quantita, NOW());
        end if;
        commit;
END$$

DELIMITER ;
```

MAG5 – Report prodotti in scadenza

Ho usato come livello d'isolamento Read Committed perché non voglio avere letture sporche.

```
USE `fruttasrldb`;
DROP procedure IF EXISTS `fruttasrldb`.`report_prodotti_in_scadenza`;

DELIMITER $$
USE `fruttasrldb`$$
CREATE DEFINER=`root`@`localhost` PROCEDURE `report_prodotti_in_scadenza`()
BEGIN
        declare exit handler for sqlexception
        begin
            rollback;
            resignal;
        end;
        set transaction isolation level read committed;
        start transaction;

        select  Prodotto, Giacenza, Scadenza, Nome
        from `prodotto in magazzino` join `prodotto` on Prodotto = Codice
        where Scadenza > now() and Scadenza <= date_add(now(), interval 7 day)
        order by Scadenza;
        commit;
END$$

DELIMITER ;
```

MAG6 – Registra Rifornimento

Ho usato come livello d'isolamento Repeatable Read perché non voglio che nessuno sia in grado di modificare o leggere la quantità disponibile di un prodotto prima della fine di quest'operazione.

```
USE `fruttasrldb`;
DROP procedure IF EXISTS `fruttasrldb`.`registra_rifornimento`;

DELIMITER $$
USE `fruttasrldb`$$
CREATE DEFINER=`root`@`localhost` PROCEDURE `registra_rifornimento`(in var_Prodotto smallint unsigned, in var_Scadenza date, in var_Quantita float)
BEGIN
    declare var int;
    declare var_totale float unsigned;
    declare var_OldGiacenza float unsigned;

    declare exit handler for sqlexception
    begin
        rollback;
        resignal;
    end;
    set transaction isolation level repeatable read;
    start transaction;

        select count(*)
        from `prodotto in magazzino`
        where Prodotto = var_Prodotto and Scadenza = var_scadenza
        into var;
```

```
        if var = 1 then
        select Giacenza
        from `prodotto in magazzino`
        where Prodotto = var_Prodotto and Scadenza = var_scadenza
        into var_OldGiacenza;

        update `prodotto in magazzino`
        set `Giacenza` = var_Quantita + var_OldGiacenza
        where Prodotto = var_Prodotto and Scadenza = var_scadenza;

        else
            insert into `prodotto in magazzino`(Prodotto, Scadenza, Giacenza) values (var_Prodotto, var_Scadenza, var_Quantita);
        end if;

        select `Quantità`
        from `prodotto`
        where Codice = var_Prodotto
        into var_totale;

        update `prodotto`
        set Quantità = var_totale + var_Quantita
        where Codice = var_Prodotto;
        commit;

END$$

DELIMITER ;
```

MAG7 – Visualizza fornitori che hanno prodotto

```
USE `fruttasrldb`;
DROP procedure IF EXISTS `fruttasrldb`.`visualizza_fornitori_che_hanno_prodotto`;

DELIMITER $$
USE `fruttasrldb`$$
CREATE DEFINER=`root`@`localhost` PROCEDURE `visualizza_fornitori_che_hanno_prodotto`(in var_IDProdotto smallint unsigned)
BEGIN

        select `fornitore`.`Codice`, `fornitore`.`Nome`
        from fornitore join disponibilità on `fornitore`.`Codice` = `disponibilità`.`Fornitore`
        where `disponibilità`.`Prodotto` = var_IDProdotto;

END$$

DELIMITER ;
```

MAG8 - Mostra richieste ai fornitori della settimana

Ho usato come livello d'isolamento Read Committed perché non voglio avere letture sporche su richieste non ancora andate in commit.

```
USE `fruttasrldb`;
DROP procedure IF EXISTS `fruttasrldb`.`mostra_richieste_ai_fornitori_settimana`;

DELIMITER $$
USE `fruttasrldb`$$
CREATE DEFINER=`root`@`localhost` PROCEDURE `mostra_richieste_ai_fornitori_settimana`()
BEGIN
        declare exit handler for sqlexception
        begin
            rollback;
            resignal;
        end;
        set transaction isolation level read committed;
        start transaction;

        select  `rifornimento`.`ID`, Prodotto, Fornitore, `rifornimento`.`Quantità`, Data, `prodotto`.`Nome` as NomeProdotto, `fornitore`.`Nome` as NomeFornitore
        from `rifornimento` join `prodotto` on Prodotto = `prodotto`.`Codice` join `fornitore` on Fornitore = `fornitore`.`Codice`
        where Data > date_sub(now(), interval 7 day)
        order by Data;
        commit;
END$$

DELIMITER ;
```

L1 – Login

```
USE `fruttasrldb`;
DROP procedure IF EXISTS `fruttasrldb`.`login`;

DELIMITER $$
USE `fruttasrldb`$$
CREATE DEFINER=`root`@`localhost` PROCEDURE `login`(in var_username varchar(45), in var_pass varchar(45), out var_role INT)
BEGIN
    declare var_user_role ENUM('manager', 'operatore', 'magazziniere');

    select ruolo from utenti
        where username = var_username
        and password = md5(var_pass)
    into var_user_role;

    if var_user_role = 'manager' then set var_role = 1;
    elseif var_user_role = 'operatore' then set var_role = 2;
    elseif var_user_role = 'magazziniere' then set var_role = 3;
    else set var_role = 4;
    end if;
END$$
```

6. Appendice: Implementazione

```
SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0;
SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS,
FOREIGN_KEY_CHECKS=0;
SET @OLD_SQL_MODE=@@SQL_MODE,
SQL_MODE='ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO
_ZERO_DATE,ERROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION';

-- -----------------------------------------------------
-- Schema mydb
-- -----------------------------------------------------
-- -----------------------------------------------------
-- Schema fruttasrldb
-- -----------------------------------------------------
DROP SCHEMA IF EXISTS `fruttasrldb` ;

-- -----------------------------------------------------
-- Schema fruttasrldb
-- -----------------------------------------------------
CREATE SCHEMA IF NOT EXISTS `fruttasrldb` DEFAULT CHARACTER SET utf8mb4
COLLATE utf8mb4_0900_ai_ci ;
USE `fruttasrldb` ;

-- -----------------------------------------------------
-- Table `fruttasrldb`.`cliente`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `fruttasrldb`.`cliente` ;

CREATE TABLE IF NOT EXISTS `fruttasrldb`.`cliente` (
  `Partita IVA` CHAR(11) NOT NULL,
  `Nome` VARCHAR(45) NOT NULL,
  `Via Residenza` VARCHAR(45) NOT NULL,
  `Città Residenza` VARCHAR(45) NOT NULL,
```

```
  `Civico Residenza` SMALLINT UNSIGNED NOT NULL,
  `Via Fatturazione` VARCHAR(45) NOT NULL,
  `Città Fatturazione` VARCHAR(45) NOT NULL,
  `Civico Fatturazione` SMALLINT UNSIGNED NOT NULL,
  PRIMARY KEY (`Partita IVA`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;


-- -----------------------------------------------------
-- Table `fruttasrldb`.`contatto`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `fruttasrldb`.`contatto` ;

CREATE TABLE IF NOT EXISTS `fruttasrldb`.`contatto` (
  `Recapito` VARCHAR(45) NOT NULL,
  `Cliente` CHAR(11) NOT NULL,
  `Tipo Contatto` ENUM('Cellulare', 'Mail', 'Telefono') NOT NULL,
  PRIMARY KEY (`Recapito`),
  INDEX `Cliente_idx` (`Cliente` ASC) VISIBLE,
  CONSTRAINT `Cliente`
    FOREIGN KEY (`Cliente`)
    REFERENCES `fruttasrldb`.`cliente` (`Partita IVA`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;


-- -----------------------------------------------------
-- Table `fruttasrldb`.`ordine`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `fruttasrldb`.`ordine` ;
```

```
CREATE TABLE IF NOT EXISTS `fruttasrldb`.`ordine` (
  `ID` INT UNSIGNED NOT NULL AUTO_INCREMENT,
  `Stato` ENUM('Aperto', 'Inviato', 'Finalizzato') NOT NULL,
  `Via` VARCHAR(45) NOT NULL,
  `Città` VARCHAR(45) NOT NULL,
  `Civico` SMALLINT UNSIGNED NOT NULL,
  `Contatto` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`ID`),
  INDEX `Contatto_idx` (`Contatto` ASC) VISIBLE,
  CONSTRAINT `Contatto`
    FOREIGN KEY (`Contatto`)
    REFERENCES `fruttasrldb`.`contatto` (`Recapito`))
ENGINE = InnoDB
AUTO_INCREMENT = 1
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;


-- -----------------------------------------------------
-- Table `fruttasrldb`.`prodotto`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `fruttasrldb`.`prodotto` ;

CREATE TABLE IF NOT EXISTS `fruttasrldb`.`prodotto` (
  `Codice` SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT,
  `Nome` VARCHAR(45) NOT NULL,
  `Quantità` FLOAT UNSIGNED NOT NULL,
  `Tipo` ENUM('Frutta', 'Verdura') NOT NULL,
  `Prezzo/kg` DECIMAL(10,2) UNSIGNED NOT NULL,
  PRIMARY KEY (`Codice`),
  UNIQUE INDEX `Nome_UNIQUE` (`Nome` ASC) VISIBLE)
ENGINE = InnoDB
AUTO_INCREMENT = 10
DEFAULT CHARACTER SET = utf8mb4
```

```
COLLATE = utf8mb4_0900_ai_ci;


-- -----------------------------------------------------
-- Table `fruttasrldb`.`composto`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `fruttasrldb`.`composto` ;

CREATE TABLE IF NOT EXISTS `fruttasrldb`.`composto` (
  `Ordine` INT UNSIGNED NOT NULL,
  `Prodotto` SMALLINT UNSIGNED NOT NULL,
  `Quantità` DECIMAL(10,2) UNSIGNED NOT NULL,
  PRIMARY KEY (`Ordine`, `Prodotto`),
  INDEX `Prodotto_idx` (`Prodotto` ASC) VISIBLE,
  CONSTRAINT `Ordine`
    FOREIGN KEY (`Ordine`)
    REFERENCES `fruttasrldb`.`ordine` (`ID`),
  CONSTRAINT `ProdottoOrdinato`
    FOREIGN KEY (`Prodotto`)
    REFERENCES `fruttasrldb`.`prodotto` (`Codice`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;


-- -----------------------------------------------------
-- Table `fruttasrldb`.`fornitore`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `fruttasrldb`.`fornitore` ;

CREATE TABLE IF NOT EXISTS `fruttasrldb`.`fornitore` (
  `Codice` SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT,
  `Nome` VARCHAR(45) NOT NULL,
  `Codice Fiscale` CHAR(16) NOT NULL,
```

```
  PRIMARY KEY (`Codice`),
  UNIQUE INDEX `Codice Fiscale_UNIQUE` (`Codice Fiscale` ASC) VISIBLE)
ENGINE = InnoDB
AUTO_INCREMENT = 6
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;


-- -----------------------------------------------------
-- Table `fruttasrldb`.`disponibilità`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `fruttasrldb`.`disponibilità` ;

CREATE TABLE IF NOT EXISTS `fruttasrldb`.`disponibilità` (
  `Prodotto` SMALLINT UNSIGNED NOT NULL,
  `Fornitore` SMALLINT UNSIGNED NOT NULL,
  PRIMARY KEY (`Prodotto`, `Fornitore`),
  INDEX `fornitoreprodotto_idx` (`Fornitore` ASC) VISIBLE,
  CONSTRAINT `fornitoreprodotto`
    FOREIGN KEY (`Fornitore`)
    REFERENCES `fruttasrldb`.`fornitore` (`Codice`),
  CONSTRAINT `prodottofornito`
    FOREIGN KEY (`Prodotto`)
    REFERENCES `fruttasrldb`.`prodotto` (`Codice`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;


-- -----------------------------------------------------
-- Table `fruttasrldb`.`indirizzo sede`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `fruttasrldb`.`indirizzo sede` ;
```

```
CREATE TABLE IF NOT EXISTS `fruttasrldb`.`indirizzo sede` (
  `Città` VARCHAR(45) NOT NULL,
  `Via` VARCHAR(45) NOT NULL,
  `Civico` SMALLINT UNSIGNED NOT NULL,
  `Fornitore` SMALLINT UNSIGNED NOT NULL,
  PRIMARY KEY (`Città`, `Via`, `Civico`),
  INDEX `Fornitore_idx` (`Fornitore` ASC) VISIBLE,
  CONSTRAINT `Fornitore`
    FOREIGN KEY (`Fornitore`)
    REFERENCES `fruttasrldb`.`fornitore` (`Codice`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;


-- -----------------------------------------------------
-- Table `fruttasrldb`.`prodotto in magazzino`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `fruttasrldb`.`prodotto in magazzino` ;

CREATE TABLE IF NOT EXISTS `fruttasrldb`.`prodotto in magazzino` (
  `Prodotto` SMALLINT UNSIGNED NOT NULL,
  `Scadenza` DATE NOT NULL,
  `Giacenza` DECIMAL(10,2) UNSIGNED NOT NULL,
  PRIMARY KEY (`Prodotto`, `Scadenza`),
  CONSTRAINT `Prodottomagazzino`
    FOREIGN KEY (`Prodotto`)
    REFERENCES `fruttasrldb`.`prodotto` (`Codice`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;


-- -----------------------------------------------------
```

```
-- Table `fruttasrldb`.`rifornimento`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `fruttasrldb`.`rifornimento` ;

CREATE TABLE IF NOT EXISTS `fruttasrldb`.`rifornimento` (
  `ID` SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT,
  `Prodotto` SMALLINT UNSIGNED NOT NULL,
  `Fornitore` SMALLINT UNSIGNED NOT NULL,
  `Quantità` DECIMAL(10,2) UNSIGNED NOT NULL,
  `Data` DATE NOT NULL,
  PRIMARY KEY (`ID`),
  INDEX `ProdottoRifornito_idx` (`Prodotto` ASC) VISIBLE,
  CONSTRAINT `ProdottoRifornito`
    FOREIGN KEY (`Prodotto`)
    REFERENCES `fruttasrldb`.`prodotto` (`Codice`))
ENGINE = InnoDB
AUTO_INCREMENT = 1
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;
-- -----------------------------------------------------
-- Table `fruttasrldb`.`utenti`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `fruttasrldb`.`utenti` ;

CREATE TABLE IF NOT EXISTS `fruttasrldb`.`utenti` (
  `username` VARCHAR(45) NOT NULL,
  `password` CHAR(32) NOT NULL,
  `ruolo` ENUM('operatore', 'manager', 'magazziniere') NOT NULL,
  PRIMARY KEY (`username`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;

USE `fruttasrldb` ;
```