Basi di Dati e Conoscenza

Progetto A.A. 2020/2021

Sistema per la gestione della vendita all'ingrosso di piante

Matricola 0240675

Davide Salerno

Indice

Non modificare il formato del documento:

- Carattere: Times New Roman, 12pt
- Dimensione pagina: A4
- Margini: superiore/inferiore 2,5cm, sinistro/destro: 1,9cm

1. Descrizione del Minimondo

Si intende realizzare un'applicazione per la gestione della vendita all'ingrosso di piante.
L'azienda Verde S.r.l. gestisce la vendita all'ingrosso di piante da interni ed esterni.
L'azienda tratta diverse specie di piante, ciascuna caratterizzata sia dal nome latino che dal nome comune, e da un codice univoco alfanumerico attraverso cui la specie viene identificata. Per ciascuna specie è inoltre noto se sia tipicamente da giardino o da appartamento e se sia una specie esotica o meno. Le piante possono essere verdi oppure fiorite. Nel caso di specie di piante fiorite, sono note tutte le colorazioni in cui una specie è disponibile.
L'azienda gestisce ordini massivi ed ha un parco clienti sia di rivendite che di privati. Per ciascun privato sono noti il codice fiscale, il nome e l'indirizzo della persona, mentre per ogni rivendita sono noti la partita iva, il nome e l'indirizzo della rivendita. In entrambi i casi, è possibile mantenere un numero arbitrario di contatti, ad esempio numeri di telefono, di cellulare, di indirizzi email. Per ciascun cliente è possibile indicare qual è il mezzo di comunicazione preferito per essere contattati. Nel caso di una rivendita, è necessario mantenere anche il nome/cognome di un referente, eventualmente associato ad altri contatti (con la possibilità, sempre, di indicarne uno preferito). Sia i clienti privati che le rivendite devono avere un indirizzo di fatturazione, che può essere differente dall'indirizzo di residenza o dall'indirizzo di spedizione.
I fornitori di Verde 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 diverse specie di piante.
Verde 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 tutte le specie di piante trattate.
Le specie di piante trattate sono gestite dai manager di Verde S.r.l.
Si vuole tener traccia di tutti gli acquisti eseguiti da ciascun cliente. Un acquisto, effettuato in una data specifica, è relativo a una certa quantità di piante appartenenti ad un certo numero di specie. Nell'ambito di un ordine è di interesse sapere a quale indirizzo questo deve essere inviato, e quale referente (se presente) 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.
Il listino prezzi, in cui si vuole tener traccia dei prezzi assunti nel tempo da ciascuna specie

di piante.

Una variazione di prezzo non deve avere effetto su un ordine già aperto ma non ancora finalizzato. I prezzi sono gestiti dai manager di Verde S.r.l.

Gli ordini vengono evasi in pacchi. Un ordine è associato ad un numero arbitrario di pacchi ed è di interesse di Verde S.r.l. tenere traccia di quali piante sono contenute all'interno di un pacco. Per motivi di ottimizzazione degli spazi, un pacco può contenere un insieme differente di specie di piante. Quando si prepara un pacco, è di interesse per l'operatore sapere quali piante devono essere ancora inserite nei pacchi, al fine di evadere correttamente l'ordine.

2. Analisi dei Requisiti

Identificazione dei termini ambigui e correzioni possibili

Linea	Termine	Nuovo termine	Motivo correzione
6	Giardino	Esterno	Ci si è già riferiti allo stesso concetto in precedenza
7	Appartamento	Interno	Ci si è già riferiti allo stesso concetto in precedenza
12, 15, 29	Contatti	Recapiti	Parola che può avere più significati e nel testo viene usata con diverso significato. La sostituisco con un termine più specifico.
13	Mezzo di comunicazione	Recapito	Ci si era riferiti precedentemente nel testo allo stesso concetto con termine differente.
13	Indirizzo email	email	Indirizzo viene usato nel testo per indicare luoghi fisici. Mentre in questo caso si sta specificando una tipologia di recapito.
15	Referente	Referente della rivendita	Specifico il termine che viene riutilizzato in seguito per indicare un concetto diverso
23	Ordine	Richiesta di rifornimento	Si confonde con l'ordine emesso dei clienti
26	Acquisto eseguito	Ordine finalizzato	Ci si riferisce con due termini allo stesso concetto, teniamo ordine finalizzato che è più specifico.
29	Referente	Referente dell'ordine	Specifico il termine che viene riutilizzato in precedenza per indicare un concetto diverso
31	Disponibilità	Giacenza	Entrambi i termini si riferiscono al numero di piante di una specie in magazzino.

Specifica disambiguata

Si intende realizzare un'applicazione per la gestione della vendita all'ingrosso di piante. L'azienda Verde S.r.l. gestisce la vendita all'ingrosso di piante da interni ed esterni. L'azienda tratta diverse specie di piante, ciascuna caratterizzata sia dal nome latino che dal nome comune, e da un codice univoco alfanumerico attraverso cui la specie viene identificata. Per ciascuna specie è inoltre noto se sia tipicamente da interno o da esterno e se sia una specie esotica o meno. Le piante possono essere verdi oppure fiorite. Nel caso di specie di piante fiorite, sono note tutte le colorazioni in cui una specie è disponibile.

L'azienda gestisce ordini massivi ed ha un parco clienti sia di rivendite che di privati. Per ciascun privato sono noti il codice fiscale, il nome e l'indirizzo della persona, mentre per ogni rivendita sono

noti la partita iva, il nome e l'indirizzo della rivendita. In entrambi i casi, è possibile mantenere un numero arbitrario di recapiti, ad esempio numeri di telefono, di cellulare ed email. Per ciascun cliente è possibile indicare quale recapito preferisce per essere contattato. Nel caso di una rivendita, è necessario mantenere anche il nome/cognome di un referente della rivendita, eventualmente associato ad altri recapiti (con la possibilità, sempre, di indicarne uno preferito). Sia i clienti privati che le rivendite devono avere un indirizzo di fatturazione, che può essere differente dall'indirizzo di residenza o dall'indirizzo di spedizione.

I fornitori di Verde 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 diverse specie di piante.

Verde S.r.l. ha un dipartimento di gestione di magazzino che tiene traccia delle giacenze ed effettua, periodicamente, richiesta di rifornimento ai fornitori per mantenere una giacenza di tutte le specie di piante trattate.

Le specie di piante trattate sono gestite dai manager di Verde S.r.l.

Si vuole tener traccia di tutti gli ordini finalizzati da ciascun cliente. Un ordine finalizzato, effettuato in una data specifica, è relativo a una certa quantità di piante appartenenti ad un certo numero di specie. È di interesse sapere a quale indirizzo un ordine finalizzato deve essere inviato, qual è il referente dell'ordine (se presente) e quale recapito 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 la giacenza in magazzino.

Il listino prezzi, in cui si vuole tener traccia dei prezzi assunti nel tempo da ciascuna specie di piante. Una variazione di prezzo non deve avere effetto su un ordine già aperto ma non ancora finalizzato. I prezzi sono gestiti dai manager di Verde S.r.l.

Gli ordini vengono evasi in pacchi. Un ordine è associato ad un numero arbitrario di pacchi ed è di interesse di Verde S.r.l. tenere traccia di quali piante sono contenute all'interno di un pacco. Per motivi di ottimizzazione degli spazi, un pacco può contenere un insieme differente di specie di piante. Quando si prepara un pacco, è di interesse per l'operatore sapere quali piante devono essere ancora inserite nei pacchi, al fine di evadere correttamente l'ordine.

Glossario dei Termini

Termine	Descrizione	Sinonimi	Collegamenti
Ordine	Un cliente effettua un ordine di un certo numero di piante. Un ordine può proseguire	Acquisto.	Cliente, referente dell'ordine, specie.

	solo se vi è disponibilità in magazzino.		
Recapito	Recapito per mettersi in contatto con le diverse figure con la quale l'azienda interagisce. Possono essere più di uno e possono essere di diverso tipo.	Contatto, mezzo di comunicazione.	Referente di rivendita, referente dell'ordine, cliente.
Cliente	Acquirente della azienda. Può essere un privato o una rivendita.	Nessuno.	Recapito, ordine.
Specie di piante	Oggetto della compravendita. L'istanza specifica è la pianta.	Nessuno.	Ordine.
Pacchi	Contenitore che raggruppa delle piante relative a un ordine	Nessuno.	Piante, specie di piante, ordine.
Listino prezzi	Costo specie di pianta	Nessuno	Specie di piante
Fornitori	Azienda da cui si rifornisce la Verde S.r.l	Nessuno	Specie di piante

Raggruppamento dei requisiti in insiemi omogenei

Frasi relative a Specie di piante
L'azienda tratta diverse specie di piante, ciascuna caratterizzata sia dal nome latino che dal nome comune, e da un codice univoco alfanumerico attraverso cui la specie viene identificata. Per ciascuna specie è inoltre noto se sia tipicamente da giardino o da appartamento e se sia una specie esotica o meno. Nel caso di specie di piante fiorite, sono note tutte le colorazioni in cui una specie è disponibile. … Il listino prezzi, in cui si vuole tener traccia dei prezzi assunti nel tempo da ciascuna specie di piante.

Frasi relative al Cliente
Per ciascun privato sono noti il codice fiscale, il nome e l'indirizzo della persona, mentre per ogni rivendita sono noti la partita iva, il nome e l'indirizzo della rivendita. In entrambi i casi, è possibile mantenere un numero arbitrario di recapiti, ad esempio numeri di telefono, di cellulare, di email. Per ciascun cliente è possibile indicare quale recapito preferisce per essere contattato. Nel caso di una

rivendita, è necessario mantenere anche il nome/cognome di un referente della rivendita, eventualmente associato ad altri recapiti (con la possibilità, sempre, di indicarne uno preferito). Sia i clienti privati che le rivendite devono avere un indirizzo di fatturazione, che può essere differente dall'indirizzo di residenza o dall'indirizzo di spedizione.

Frasi riferite ai Fornitori

I fornitori di Verde 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 diverse specie di piante.

…richiesta di rifornimento ai fornitori per mantenere una giacenza di tutte le specie di piante trattate.

Frasi relative agli ordini

Si vuole tener traccia di tutti gli ordini finalizzati da ciascun cliente. Un ordine, effettuato in una data specifica, è relativo a una certa quantità di piante appartenenti ad un certo numero di specie. E' di interesse sapere a quale indirizzo un ordine deve essere inviato, qual è il referente dell'ordine (se presente) e quale recapito 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 la giacenza in magazzino.

Frasi riferite ai Pacchi

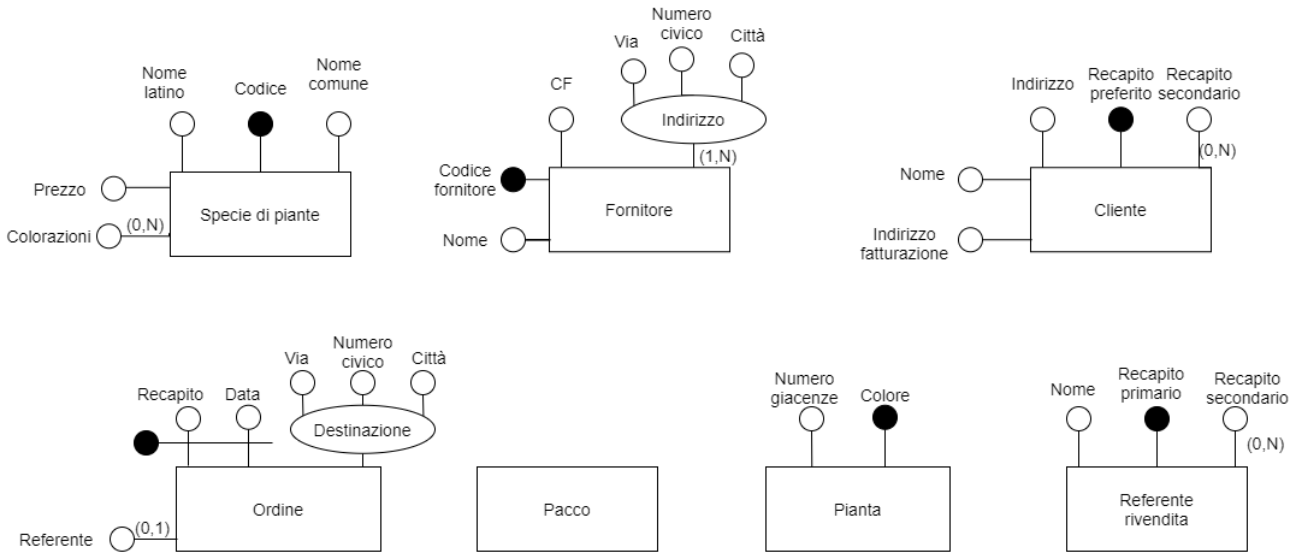
Gli ordini vengono evasi in pacchi. Un ordine è associato ad un numero arbitrario di pacchi ed è di interesse di Verde S.r.l. tenere traccia di quali piante sono contenute all'interno di un pacco. Per motivi di ottimizzazione degli spazi, un pacco può contenere un insieme differente di specie di piante. Quando si prepara un pacco, è di interesse per l'operatore sapere quali piante devono essere ancora inserite nei pacchi, al fine di evadere correttamente l'ordine.

3. Progettazione concettuale

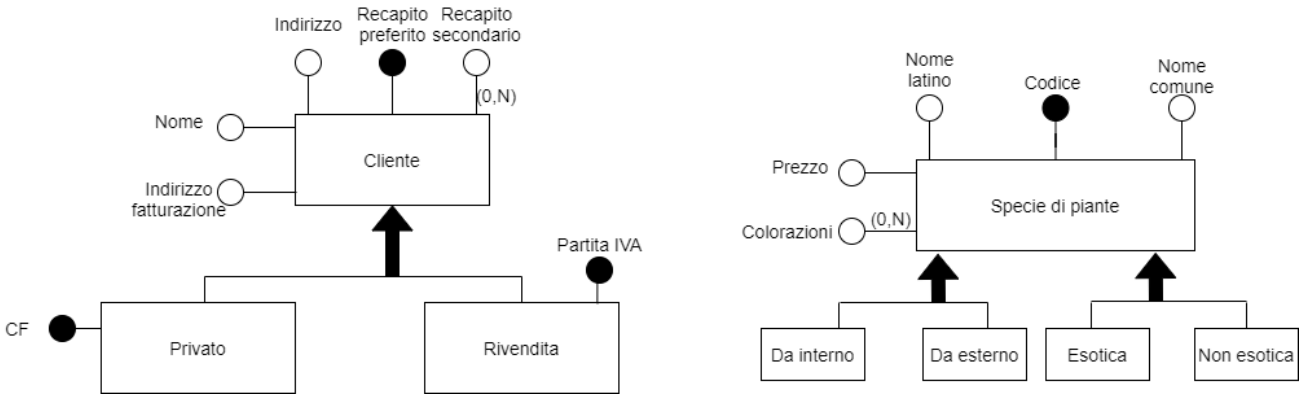
Costruzione dello schema E-R

Verrà applicata una strategia bottom-up per costruire lo schema E-R a partire dalle specifiche precedentemente elaborate.

Nella prima fase individuiamo i componenti "importanti" della nostra base di dati. Le entità individuate vengono riportate con i relativi attributi individuati a un primo studio delle specifiche. Iniziamo anche a individuare degli identificatori.

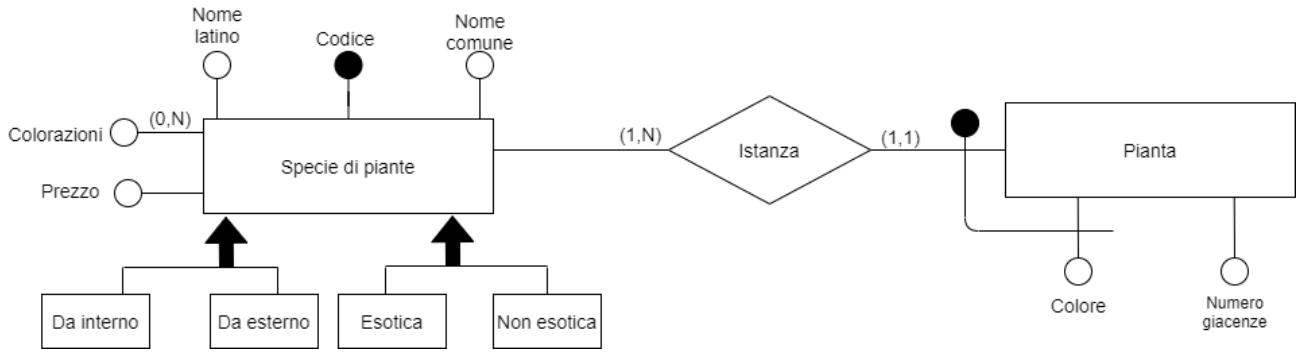


Specializziamo ora ulteriormente i concetti individuati fino ad arrivare ad avere tutti frammenti elementari.

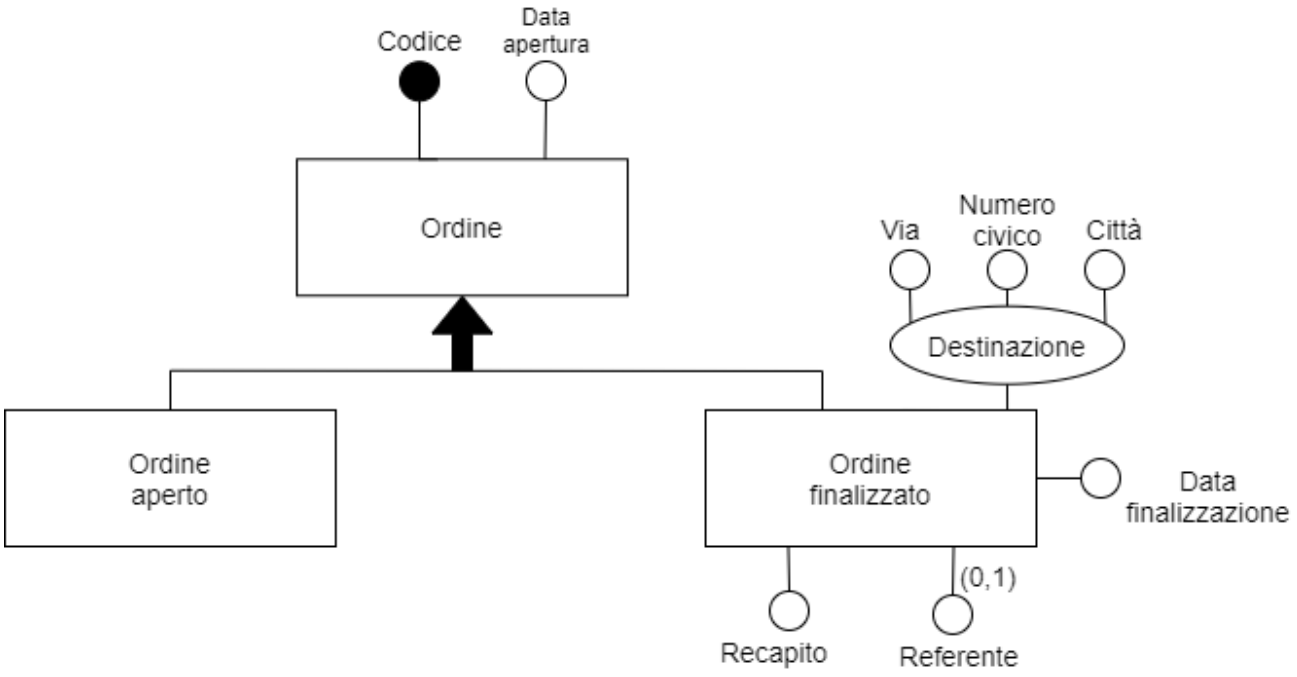


Inizio ad integrare i vari elementi del diagramma creando le prime relazioni.

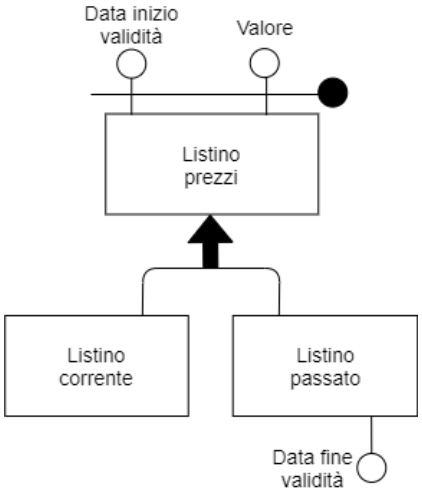
Utilizzo il design pattern istance-of:



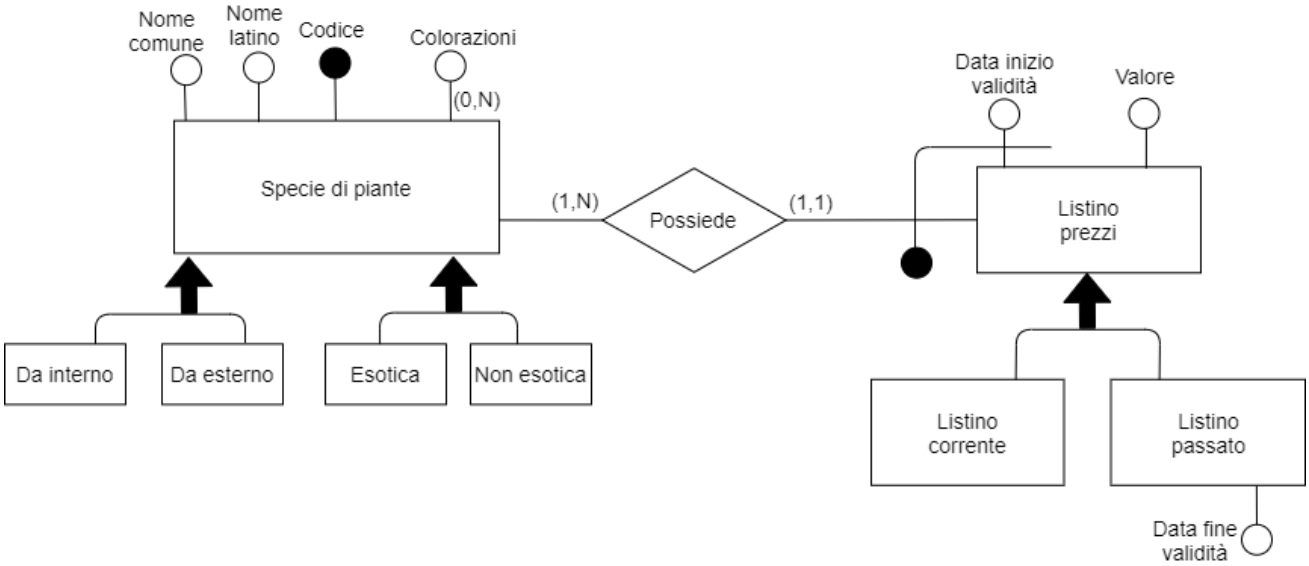
Introduco una generalizzazione sugli ordini. Introduco anche l'attributo "Codice" all'entità "Ordine" data l'assenza di ulteriori metodi per identificare un ordine.



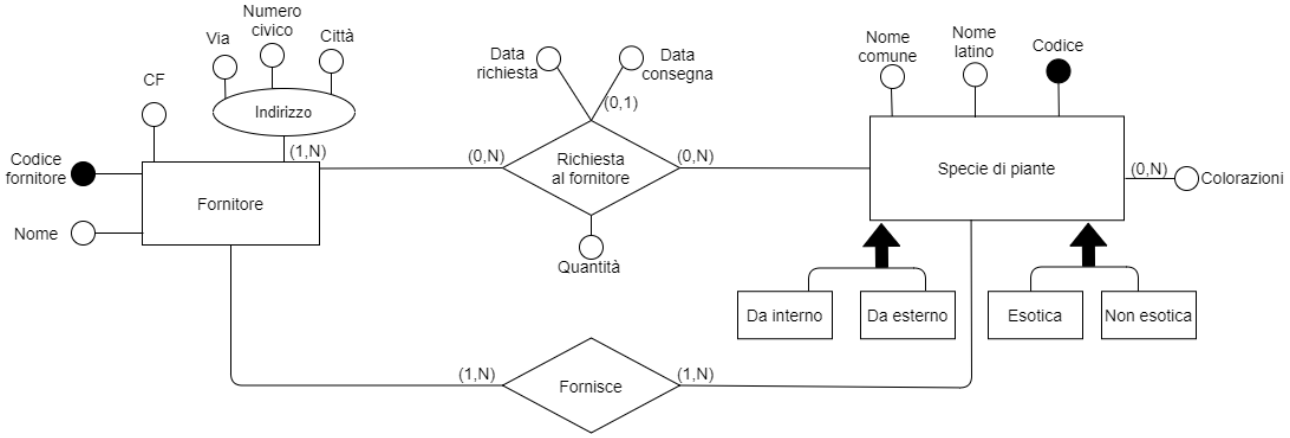
Reifico l'attributo "prezzo" di "specie di pianta" al fine di realizzare una storicizzazione riguardante i listini posseduti da una specie nel tempo.



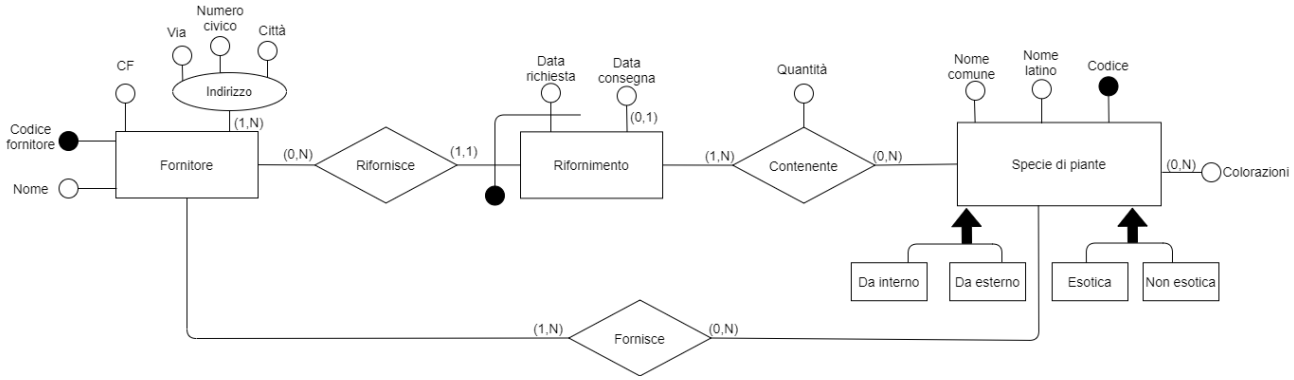
L'identificatore di "Listino prezzi" mostra dei problemi in caso di un listino con stesso valore e data inizio validità ma data fine validità diversa. Lavoriamo su come legare questa entità con la specie a cui è associato il listino. Introduciamo a questo punto un identificatore esterno che ci permette data una specie e una data inizio validità un unico listino.



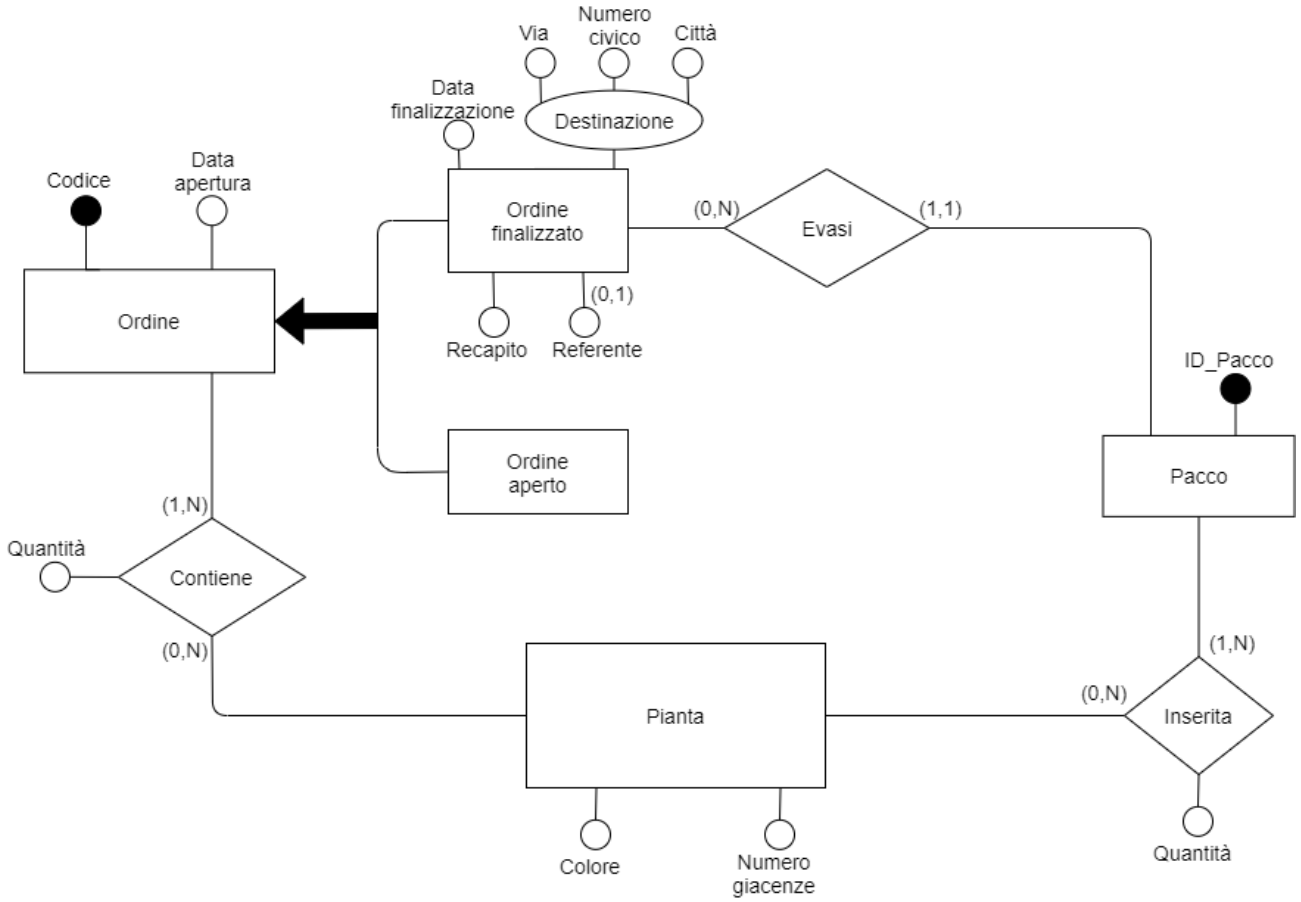
Introduciamo ora le relazioni che legano i fornitori alle specie da rifornire. La relazione "Fornisce" serve per indicare quali specie fornisce un fornitore. La relazione "Richiesta al fornitore" mantiene le effettive richieste elaborate. Questa relazione inoltre sarà utile alla gestione del magazzino per tenere traccia di rifornimenti già richiesti ma non ancora ricevuti.



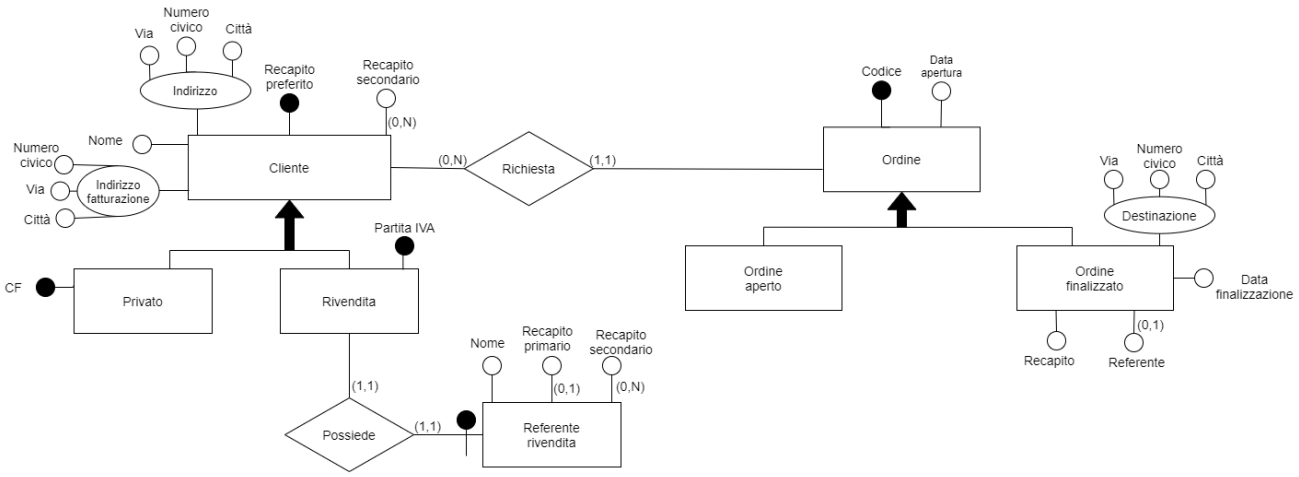
A questo punto ci rendiamo conto che l'associazione "Richiesta al fornitore" non permette di identificare due richieste allo stesso fornitore riguardanti stessa specie ma effettuate in data diverse. Reifichiamo quindi l'associazione così da poter inserire l'attributo "Data richiesta" tra gli identificatori. Così facendo la nuova entità creata viene chiamata "Rifornimento" e va a indicare un rifornimento eseguito in una determinata data riguardante un insieme di specie con le relative quantità richieste.



Continuiamo con l'integrazione: consideriamo le relazioni tra Ordine, Pacco e Pianta. Introduco un attributo aggiuntivo all'entità "Pacco" al fine di poterla identificare.



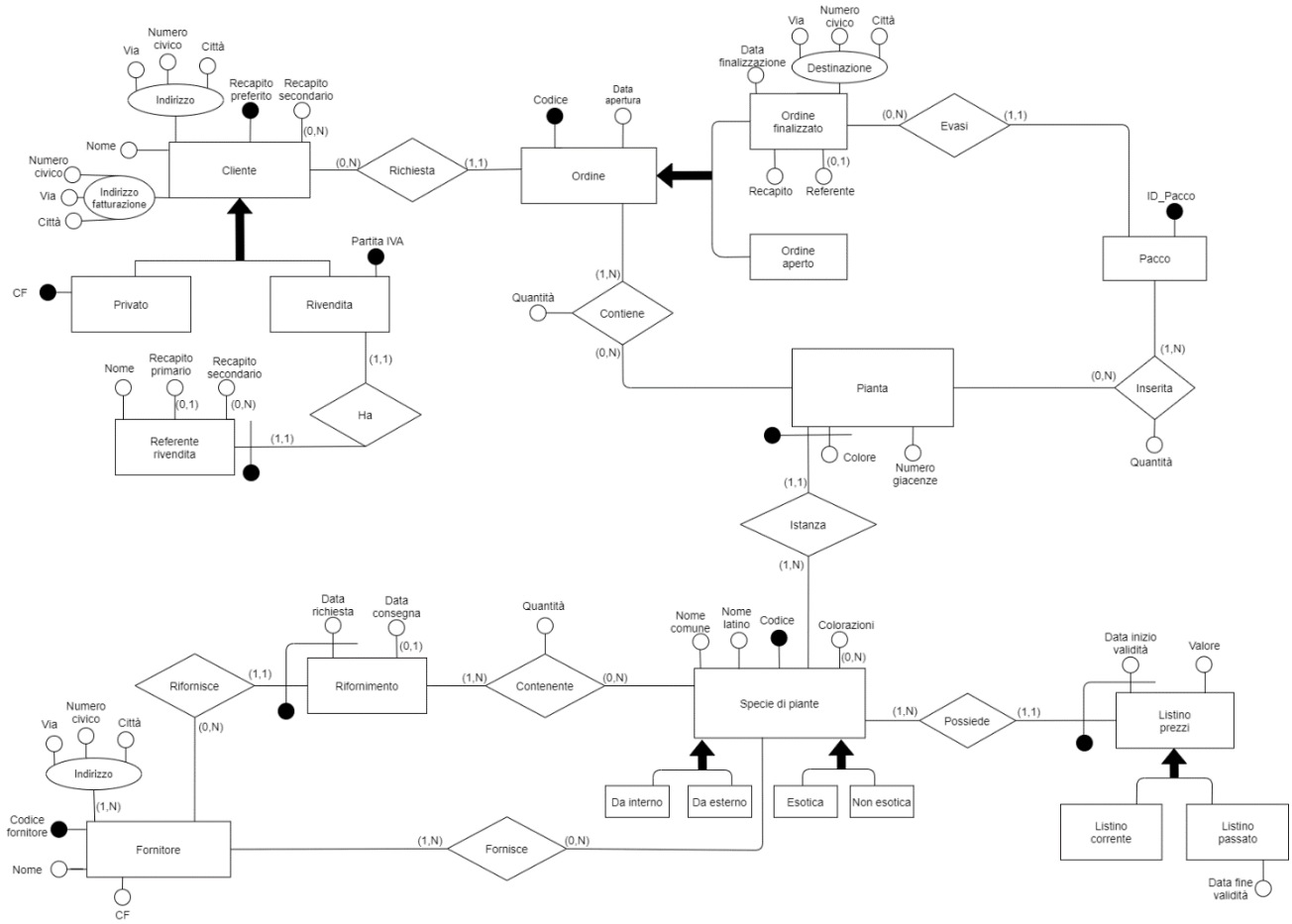
Considero le relazioni con cliente:



Integrazione finale

In questa fase vengono unite tutte le varie parti sviluppate precedentemente. Si risolvono quindi eventuali conflitti strutturali o conflitti sull'uso di stessi nomi in due concetti diversi.

Modifico il nominativo "Possiede" relativo alla relazione tra le entità "Rivendita" e "Referente rivendita" in quanto è in conflitto con quello tra "Specie di pianta" e "Listino prezzi".



Regole aziendali

Regole di vincolo

1. Un ordine non deve poter essere aperto se non vi è disponibilità in magazzino.
2. Una variazione di prezzo non deve avere effetto su un ordine già aperto ma non ancora finalizzato.

Regole di derivazione

1. Il numero di giacenze in magazzino di una specie si ottiene sommando le giacenze delle istanze di una specie.
2. Le specie da rifornire si ottengono delle giacenze di una specie meno le richieste di rifornimento non ancora consegnate.
3. Le piante da inserire si ottengono sottraendo alle piante contenute in un ordine finalizzato le piante inserite nei pacchi relative agli ordini.

Dizionario dei dati

Entità	Descrizione	Attributi	Identificatori
Cliente	Clienti dell'azienda	Recapito preferito, Recapito secondario, Nome, Indirizzo (Via, Numero civico, Città), Indirizzo fatturazione (Via, Numero civico, Città).	Recapito preferito
Privato	Specializzazione dell'entità Cliente: cliente che acquista in proprio	CF (attributi padre).	CF oppure Recapito preferito
Rivendita	Specializzazione di Cliente: società di rivendita piante.	Partita IVA (attributi padre).	Partita IVA oppure Recapito preferito
Referente rivendita	Ad ogni rivendita viene associato un referente	Nome, Recapito primario, recapito secondario.	Rivendita (identificatore esterno)
Ordine	Insieme di prodotti acquistati da un cliente	Codice, Data apertura.	Codice
Ordine aperto	Specificazione di Ordine: Ordine in corso di compilazione	(attributi padre).	Codice
Ordine finalizzato	Specificazione di Ordine: Ordine di cui è stata terminata la compilazione	Data finalizzazione, Destinazione (Via, Numero civico, Città), Recapito, Referente (attributi padre).	Codice

Pacco	Contiene parte delle piante relative a un ordine in fase di spedizione	ID_Pacco	ID_Pacco
Pianta	Istanza di Specie di piante	Colore, Numero giacenze	Colore, Specie di piante (identificatore esterno)
Specie di piante	Insieme di piante della stessa specie	Codice, Colorazioni, Nome comune, Nome latino	Codice
Listino prezzi	Tiene traccia dei prezzi della specie di piante nel tempo	Valore, Data inizio validità	Valore, Specie di piante (identificatore esterno)
Listino corrente	Listino attualmente associato a una specie	(attributi padre)	Valore, Specie di piante (identificatore esterno)
Listino passato	Listino associato in un periodo di tempo passato a una specie	Data fine validità, (attributi padre).	Valore, Specie di piante (identificatore esterno)
Fornitore	Rifornisce l'azienda	Nome, Codice fornitore, CF, Indirizzo (Via, Numero civico, Città)	Codice fornitore
Rifornimento	Richiesta effettuata di un rifornimento di specie di piante terminate	Data richiesta, data consegna	Data consegna, Fornitore (identificatore esterno)

4. Progettazione logica

Volume dei dati

Concetto nello schema	Tipo[1]	Volume atteso
Cliente	E	10.000
Privato	E	9.000
Rivendita	E	1.000
Referente rivendita	E	1.000
Ordine	E	36.500 **
Ordine aperto	E	500
Ordine finalizzato	E	36.000 **
Pacco	E	108.000 **
Pianta	E	3.000
Specie di piante	E	1.000
Da interno	E	200
Da esterno	E	800
Esotica	E	100
Non esotica	E	900
Fornitore	E	10
Rifornimento	E	360 **
Listino prezzi	E	10.000
Listino corrente	E	1.000
Listino passato	E	9.000
Richiesta	R	36.500 **
Evasi	R	108.000 **
Ha	R	1.000
Contiene	R	109.500 **
Inserita	R	216.000 **
Istanza	R	3.000

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

Possiede	R	1.000
Rifornisce	R	360 **
Fornisce	R	3.000
Contenente	R	1.800 **

Considerazioni

** Data la presenza di concetti di volumi che crescono nel tempo senza raggiungere un valore atteso, optiamo per la scelta di storicizzazione della base di dati. Struttureremo la nostra base in modo tale da avere una sezione "operativa" e una di "archivio", ogni periodo di tempo utilizzeremo operazioni per spostare i dati più vecchi nella sezione "archivio". Per l'ipotesi sui volumi viene prevista questa operazione ogni 6 mesi.

Assunzioni e stime

Pianta = Specie * media colorazioni = 1.000 * 3 = 3.000

Ordine finalizzato = 6 mesi * (ordini al mese) = 6 * (30 * 200) = 36.000

Pacco = Ordine finalizzato * media pacchi per ordine = 36.000 * 3 = 108.000

Fornisce = Fornitori * media specie fornite da ogni fornitore = 10 * 300 = 3.000

Inseriti = Pacco * media piante diverse per pacco (se sono uguali contano uno perché c'è l'attributo quantità) = 108.000 * 2 = 216.000

Rifornimento = 6 mesi * numero rifornimenti al mese = (6 * 30)giorni * 2 rifornimenti al giorno = 360

Contiene = Ordine * 3 = 109.500

Contenente = Rifornimenti * numero medio specie in un rifornimento = 360 * 5 = 1800

Tavola delle operazioni

Cod.	Descrizione	Frequenza attesa
OP1	Aggiungere una nuova specie	1/mese
OP2	Aggiungere un nuovo cliente privato	9/g
OP3	Aggiungere una nuova rivendita alla clientela	1/g
OP4	Aggiungere un nuovo fornitore	1/mese
OP5	Richiesta di rifornimento	2/g
OP6	Finalizzazione ordine	200/g
OP7	Visualizza ordini finalizzati relativi a un cliente	10/g

OP8	Visualizza info di un ordine finalizzato (piante e prezzo)	50/g
OP9	Visualizza pacchi di un ordine	20/g
OP10	Visualizza piante da inserire nei pacchi relative a un ordine finalizzato	2/g
OP11	Inserisci pianta in un nuovo pacco relativo a un ordine finalizzato	600/g
OP12	Modifica piante inserite in un pacco	300/g
OP13	Visualizza specie da rifornire	1/g
OP14	Crea ordine aperto	250/g
OP15	Visualizza un ordine aperto	200/g
OP16	Inserisci o elimina una pianta da un ordine aperto	1.000/g
OP17	Elimina ordine aperto	50/g
OP18	Visualizzare una specie in vendita con relative informazioni	5.000/g
OP19	Visualizzare listini passati di una specie	10/mese
OP20	Modifica dati cliente privato	1/g
OP21	Modifica dati rivendita	1/mese
OP22	Modifica dati referente rivendita	1/mese
OP23	Modifica pianta di una specie	1/mese
OP24	Notifica arrivo rifornimento	2/g
OP25	Modifica dati specie (esclusa modificazione numero giacenza considerato precedentemente)	1/g
OP26	Crea listino	5/g
OP27	Modifica dati fornitore	1/mese
OP28	Archiviazione dati	2/anno

Costo delle operazioni

Operazione 1: aggiungere una nuova specie			
Concetto	**Costrutto**	**Accesso**	**Tipo**

Specie di piante	E	1	S
Istanza	R	3	S
Pianta	E	3	S
Possiede	R	1	S
Listino prezzi	E	1	S
Fornisce	R	1	S

Costo (considero 3 colorazioni in media per specie) = 2 * 10 * 1/mese = 20/mese

Operazione 2: aggiungere un nuovo cliente privato			
Concetto	**Costrutto**	**Accesso**	**Tipo**
Cliente	E	1	S
Privato	E	1	S

Costo = 2 * 2 * 9/g = 36/g

Operazione 3: aggiungere una nuova rivendita alla clientela			
Concetto	**Costrutto**	**Accesso**	**Tipo**
Cliente	E	1	S
Rivendita	E	1	S
Ha	R	1	S
Referente rivendita	E	1	S

Costo = 2 * 4 * 1/g = 8/g

Operazione 4: aggiungere un nuovo fornitore			
Concetto	**Costrutto**	**Accesso**	**Tipo**
Fornitore	E	1	S
Fornisce	R	300	S

Costo (in media un fornitore fornisce 300 specie) = 301 * 2 * 1/mese = 602/mese ~ 20/g

Operazione 5: richiesta di rifornimento			
Concetto	**Costrutto**	**Accesso**	**Tipo**
Rifornisce	R	1	S
Rifornimento	E	1	S
Contenente	R	5	S

Costo = 2 * 7 * 2/g = 28/g

Operazione 6: finalizzazione ordine			
Concetto	**Costrutto**	**Accesso**	**Tipo**
Ordine aperto	E	1	L
Contiene	R	3	L
Ordine aperto (eliminazione)	E	1	S
Contiene (eliminazione)	R	3	S
Ordine finalizzato	E	1	S
Contiene	R	3	S

Costo = (4+2*8) * 200/g = 4.000/g

Operazione 7: visualizza ordini relativi a un cliente (negli ultimi 6 mesi)			
Concetto	**Costrutto**	**Accesso**	**Tipo**
Richiesta	R	4	L
Ordine	E	4	L

La stima del numero degli accessi di richiesta è ottenuta: numero ordini/numero clienti
Costo = 8*10/g = 80/g

Operazione 8: Visualizza info di un ordine (richiedente, piante e prezzo)			
Concetto	**Costrutto**	**Accesso**	**Tipo**
Ordine	E	1	L
Richiesta	R	1	L
Cliente	E	1	L
Contiene	R	3	L
Pianta	E	3	L
Istanza	R	3	L
Specie di piante	E	2	L
Possiede	R	2	L
Listino prezzi	E	2	L

Gli accessi a Specie sono minori di quelli di Pianta perché stesse piante potrebbero appartenere alla stessa specie.
Costo = 18*50/g = 900/g

Operazione 9: visualizza pacchi di un ordine			
Concetto	**Costrutto**	**Accesso**	**Tipo**

Evasi	R	3	L
Pacco	E	3	L
Inserita	R	5	L
Pianta	E	3	L

Inserita ha 5 accessi e non 3 perché per un pacco potrebbero esserci due piante differenti (esempio 2 rose rosse e 3 rose bianche).
Costo = 14*20/g = 280/g

Operazione 10: visualizza piante da inserire nei pacchi			
Concetto	**Costrutto**	**Accesso**	**Tipo**
Ordine finalizzato	E	36.000	L
Contiene	R	108.000	L
Pianta	E	108.000	L
Evasi	R	108.000	L
Pacco	E	108.000	L
Inserita	R	180.000	L
Pianta	E	108.000	L

Leggo le piante contenute in un ordine e sottraggo quelle già inserite nei pacchi. Questo procedimento va ripetuto per ogni Ordine finalizzato. Cerchiamo nella sezione della ristrutturazione delle generalizzazioni di ridurre il costo di questa operazione incredibilmente alto rispetto agli altri.

Costo = 756.000*2/g = 1.512.000/g

Operazione 11: inserisci pianta in un nuovo pacco relativo a un ordine			
Concetto	**Costrutto**	**Accesso**	**Tipo**
Evasi	R	1	S
Pacco	E	1	S
Inserita	R	1	S

Costo = 2 * 3 * 600/g = 3.600

Operazione 12: modifica piante inserite in un pacco			
Concetto	**Costrutto**	**Accesso**	**Tipo**
Inserita	R	1	S

Costo = 2 * 300/g = 600/g

Operazione 13: visualizza specie da rifornire			
Concetto	**Costrutto**	**Accesso**	**Tipo**
Specie	E	1.000	L
Rifornimento	E	3	L
Contenente	R	15	L
Istanza	R	3.000	L
Pianta	E	3.000	L

Ho bisogno di visualizzare i rifornimenti senza una data di consegna con il relativo contenuto per sommare alle giacenze di una specie, le giacenze già ordinate ma non ancora arrivate.
Costo = 7.018*1/g = 7.018/g

Operazione 14: crea ordine aperto			
Concetto	**Costrutto**	**Accesso**	**Tipo**
Ordine	E	1	S
Ordine aperto	E	1	S
Contiene	R	1	S
Pianta	E	1	S

Per realizzare il vincolo aziendale sulla apertura di un ordine solo in caso di giacenza, vado a modificare il numero di giacenze di una pianta all'inserimento di una pianta in un ordine.

Costo = 2 * 4 * 250/g = 2.000/g

Operazione 15: visualizza un ordine aperto			
Concetto	**Costrutto**	**Accesso**	**Tipo**
Ordine aperto	E	1	L
Contiene	R	2	L
Pianta	E	2	L
Istanza	R	2	L
Specie di piante	E	2	L
Possiede	R	18	L
Listino prezzi	E	18	L
Listino passato	E	18	L

Costo = 63*200/g = 12.600/g

Operazione 16: Inserisci o elimina una pianta a un ordine aperto			
Concetto	**Costrutto**	**Accesso**	**Tipo**
Contiene	R	1	S
Pianta	E	1	S

Costo = 2*2*1.000/g = 4.000/g

Operazione 17: Elimina ordine aperto			
Concetto	**Costrutto**	**Accesso**	**Tipo**
Ordine aperto	E	1	S
Contiene	R	3	S
Pianta	E	3	S

Costo = 2*7*50/g = 700/g

Operazione 18: visualizzare una specie in vendita con relative informazioni			
Concetto	**Costrutto**	**Accesso**	**Tipo**
Pianta	E	3	L
Istanza	R	3	L
Specie	E	1	L
Possiede	R	1	L
Listino prezzi	E	1	L

Costo = 9*5.000/g = 45.000/g

Operazione 19: visualizzare listini passati di una specie			
Concetto	**Costrutto**	**Accesso**	**Tipo**
Specie	E	1	L
Possiede	R	9	L
Listino passato	E	9	L

Costo = 19*10/mese = 190/mese = 6.3/g

Operazione 20: modifica dati cliente privato			
Concetto	**Costrutto**	**Accesso**	**Tipo**
Cliente	E	1	S

Privato	E	1	S

Costo = 2 * 2 * 1/g = 4/g

Operazione 21: modifica dati rivendita			
Concetto	**Costrutto**	**Accesso**	**Tipo**
Cliente	E	1	S
Rivendita	E	1	S

Costo = 2 * 2 * 1/mese = 4/mese

Operazione 22: modifica dati referente rivendita			
Concetto	**Costrutto**	**Accesso**	**Tipo**
Ha	R	1	S
Referente rivendita	E	1	S

Costo = 2*2*1/mese = 4/mese

Operazione 23: modifica pianta di una specie			
Concetto	**Costrutto**	**Accesso**	**Tipo**
Pianta	E	1	S
Istanza	R	1	S

Costo = 2*2*1/mese = 4/mese

Operazione 24: notifica arrivo rifornimento			
Concetto	**Costrutto**	**Accesso**	**Tipo**
Rifornimento	E	1	S
Contenente	R	5	L
Specie	E	5	L
Istanza	R	15	L
Pianta	E	15	S

Costo = (2*16+25) * 2/g = 114/g (in media 5 specie in un rifornimento; per ogni specie aggiorniamo in media il numero di giacenze di 3 piante).

Operazione 25: modifica dati specie			
Concetto	**Costrutto**	**Accesso**	**Tipo**
Specie	E	1	S

Costo = 2*1/g = 2/g

Operazione 26: Crea nuovo listino			
Concetto	**Costrutto**	**Accesso**	**Tipo**
Possiede	R	1	S
Listino	E	1	S
Listino corrente	E	1	S

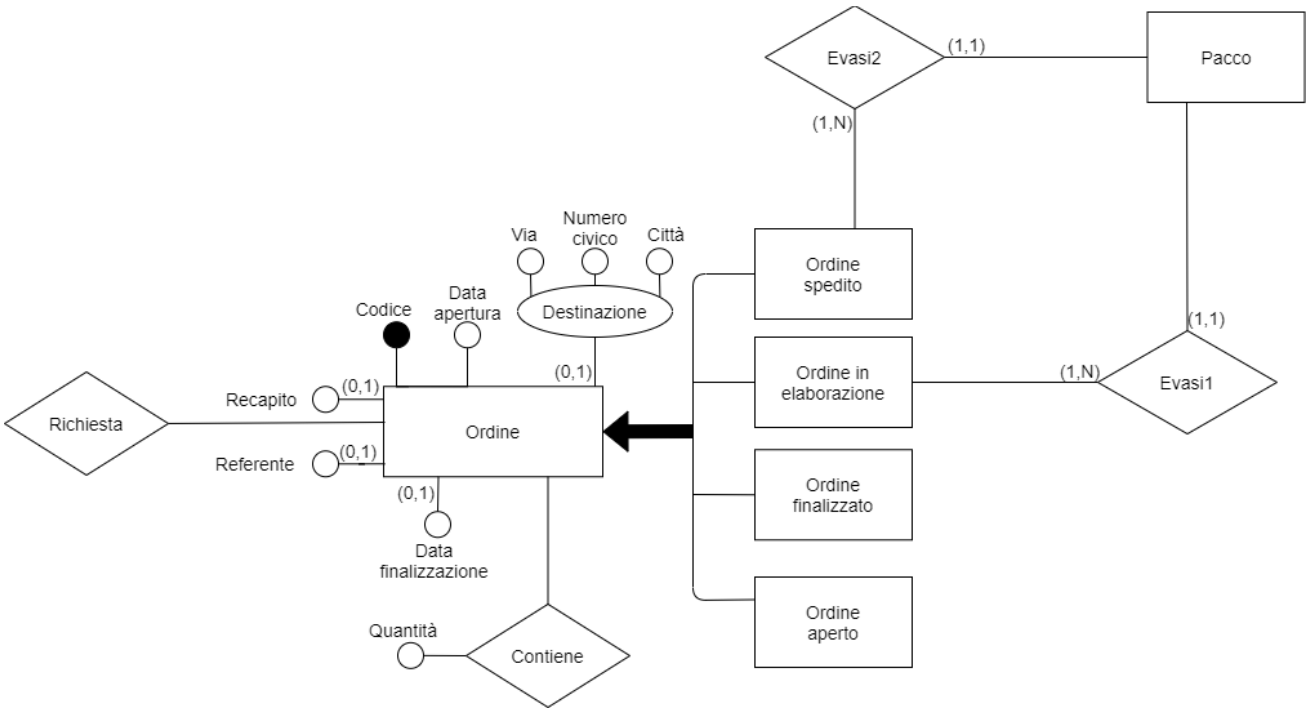
Costo = 2*3*5/g = 30/g

Operazione 27: modifica dati fornitore			
Concetto	**Costrutto**	**Accesso**	**Tipo**
Fornitore	E	1	S
Fornisce	R	1	S

Costo = 2*2*1/mese = 4/mese

Ristrutturazione dello schema E-R

Prima delle varie analisi introduciamo una ulteriore specializzazione di Ordine così da permetterci di ridurre fortemente la costosissima OP10:



Grazie a questa specificazione nella OP10 basta accedere agli Ordini in elaborazione che avranno un volume stimato di 300. Si è inoltre spostato gli attributi delle entità nel padre (avremo la presenza di valori nulli solo in caso di Ordini Aperti, che sono un numero trascurabile rispetto il numero totale di ordini). Riportiamo la tabella degli accessi aggiornata.

Operazione 10: visualizza piante da inserire nei pacchi			
Concetto	**Costrutto**	**Accesso**	**Tipo**
Ordine	E	36.000	L
Ordine in elaborazione	E	300	L
Contiene	R	900	L
Pianta	E	900	L
Evasi	R	300	L
Pacco	E	300	L
Inserita	R	300	L
Pianta	E	300	L

Considero in media un pacco già inserito nella base di dati per ogni Ordine in elaborazione. Il risparmio è enorme: costo = 39.300 * 2/g = 78.600/g

Ridefiniamo quindi ora i volumi cambiati con questa specificazione introdotta: Ordine, 37.000; Aperto, 500; Finalizzato 200; In elaborazione, 300; Spedito, 36.000;
Aggiungiamo un'operazione per spedire l'ordine una volta completata l'elaborazione:

Operazione 30: spedisci ordine			
Concetto	**Costrutto**	**Accesso**	**Tipo**
Ordine in elaborazione	E	1	L
Evasi	R	3	L
Ordine in elaborazione (eliminazione)	E	1	S
Evasi (eliminazione)	R	3	S
Ordine spedito	E	1	S
Evasi	R	3	S

Costo = (4 + 8*2) * 200/g = 4.000/g

Inseriamo anche un'operazione per il momento in cui un operatore inizia l'elaborazione di un ordine finalizzato:

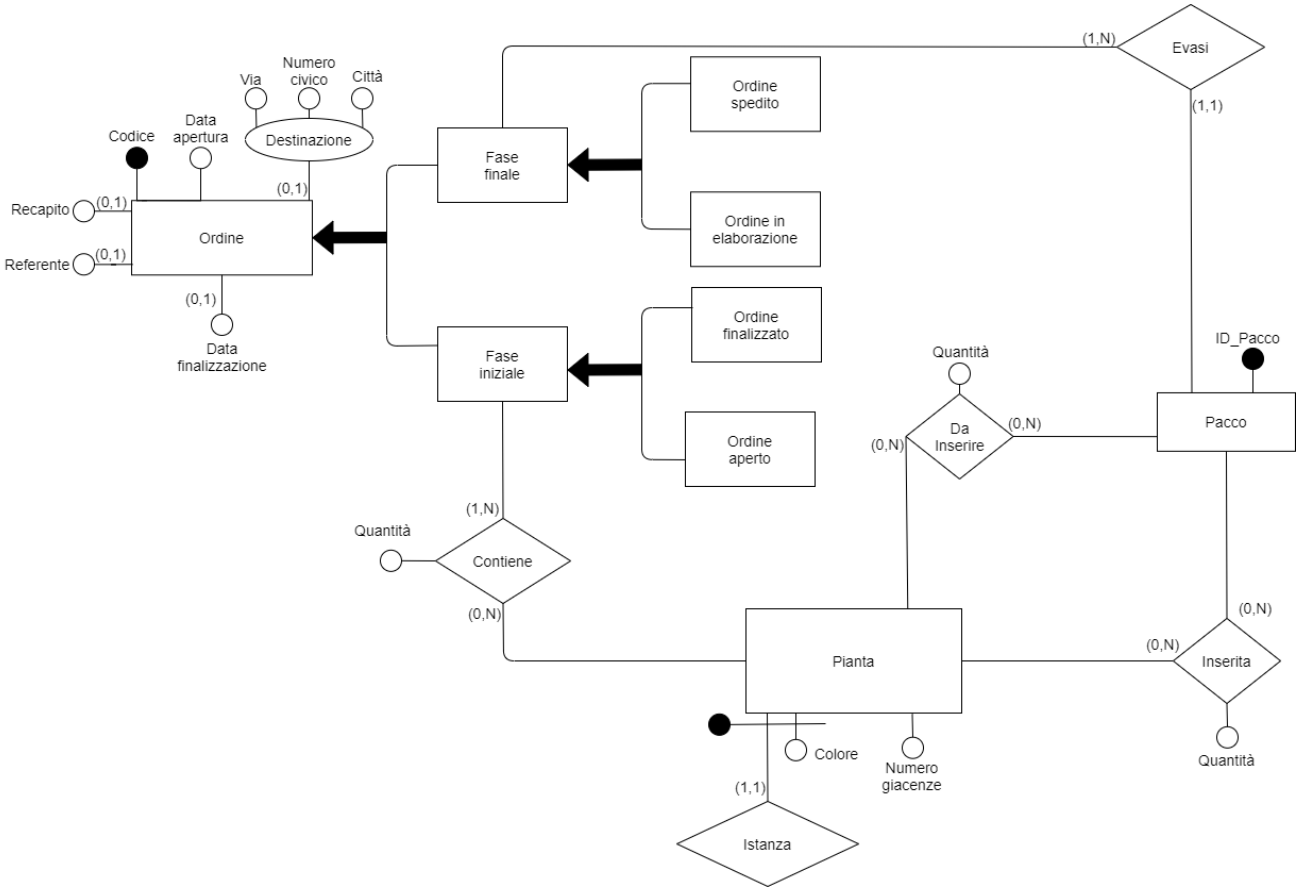
Operazione 31: prendi in elaborazione un ordine			
Concetto	**Costrutto**	**Accesso**	**Tipo**
Ordine finalizzato	E	1	L
Contiene	R	3	L
Ordine finalizzato (eliminazione)	E	1	S
Contiene (eliminazione)	R	3	S

Ordine in elaborazione	E	1	S
Contiene	R	3	S
Evasi	R	1	S
Pacco	E	1	S
Inserita	R	1	S

Costo = (4+11*2)*200/g = 5.200/g

Analisi delle ridondanze

- Le colorazioni di una specie possono essere derivate visualizzando il colore delle varie istanze di una specie (piante).
  L'attributo Colorazioni non migliora le prestazioni della base di dati perché per ogni operazione che richiede la colorazione di una specie si effettua anche l'accesso a pianta. Mantenere l'attributo inoltre costa: media colorazioni per specie * costo per salvare una colorazione * volume di Specie = 3 * 20 byte * 1000 = 60.000 byte = 60 kB
- Un Ordine spedito è evaso in pacchi che contengono le piante relative all'ordine. Sommando quindi le piante contenute nei pacchi otterremmo le piante contenute in un ordine, dato già presente tramite l'associazione "contiene" tra ordine e pianta.
  Data l'alta frequenza delle operazioni legate a questa sezione di schema proponiamo qui di seguito un'alternativa e studiamo quale sia la scelta migliore.



Consideriamo i costi delle operazioni nei due casi, calcoliamo quindi le operazioni nel caso di eliminazione della ridondanza.

Operazione 8: Visualizza info di un ordine (richiedente, piante e prezzo)			
Concetto	**Costrutto**	**Accesso**	**Tipo**
Ordine	E	1	L
Richiesta	R	1	L
Cliente	E	1	L
Evasi	R	3	L
Pacco	E	3	L
Inserita	R	5	L
Pianta	E	3	L
Istanza	R	3	L
Specie di piante	E	2	L
Possiede	R	2	L
Listino prezzi	E	2	L

Costo = 26 * 50/g = 1.300/g

Per il passaggio da Ordine finalizzato a Ordine in elaborazione siamo costretti a dover impostare il sistema in modo tale che automaticamente crei un numero di pacchi e imposti in quali inserire le piante relative all'odine. Dato che non vogliamo che l'operatore si trovi a dover eliminare pacchi che erano stati istanziati automaticamente ma poi non utilizzati, impostiamo che nell'operazione con la quale l'operatore comunica di prendere in elaborazione un ordine finalizzato vengano impostate come "da inserire" tutte le piante nello stesso pacco. Questo complica l'operazione di inserimento perché significa che ogni volta che ci sarà un inserimento dovremo andare a eliminare associazioni "da inserire".

Operazione 31: prendi in elaborazione un ordine			
Concetto	**Costrutto**	**Accesso**	**Tipo**
Ordine finalizzato	E	1	L
Contiene	R	3	L
Ordine finalizzato (eliminazione)	E	1	S
Contiene (eliminazione)	R	3	S
Ordine in elaborazione	E	1	S
Evasi	R	1	S

Pacco	E	1	S
Da inserire	R	3	S

Costo = (4+10*2) *200/g = 4.800/g

Operazione 10: visualizza piante da inserire nei pacchi			
Concetto	**Costrutto**	**Accesso**	**Tipo**
Ordine	E	36.000	L
Ordine in elaborazione	E	300	L
Evasi	R	300	L
Pacco	E	300	L
Da inserire	R	900	L
Pianta	E	900	L

Costo = 38700 * 2/g = 77.400/g

Operazione 11: inserisci pianta in un nuovo pacco relativo a un ordine			
Concetto	**Costrutto**	**Accesso**	**Tipo**
Evasi	R	1	S
Pacco	E	1	S
Inserita	R	1	S
Da inserire	R	1	S

Va eliminata la relazione “da inserire” quando vengono inserite delle piante in un pacco.
Costo 4*2*600/g = 4.800/g

Operazione 12: modifica piante inserite in un pacco			
Concetto	**Costrutto**	**Accesso**	**Tipo**
Inserita	R	1	S
Da inserire	R	1	S

Costo = 2*2*300/g = 1.200/g

Calcolo delle differenze:
OP8: 1.300/g – 900/g = + 400/g;
OP31: 4.800/g - 5.200/g = - 400/g
OP10: 77.400/g – 78.600/g = - 1.200/g;
OP11: 4.800/g – 3.600/g = + 1.200/g
OP12: 1.200/g – 600/g = + 600/g

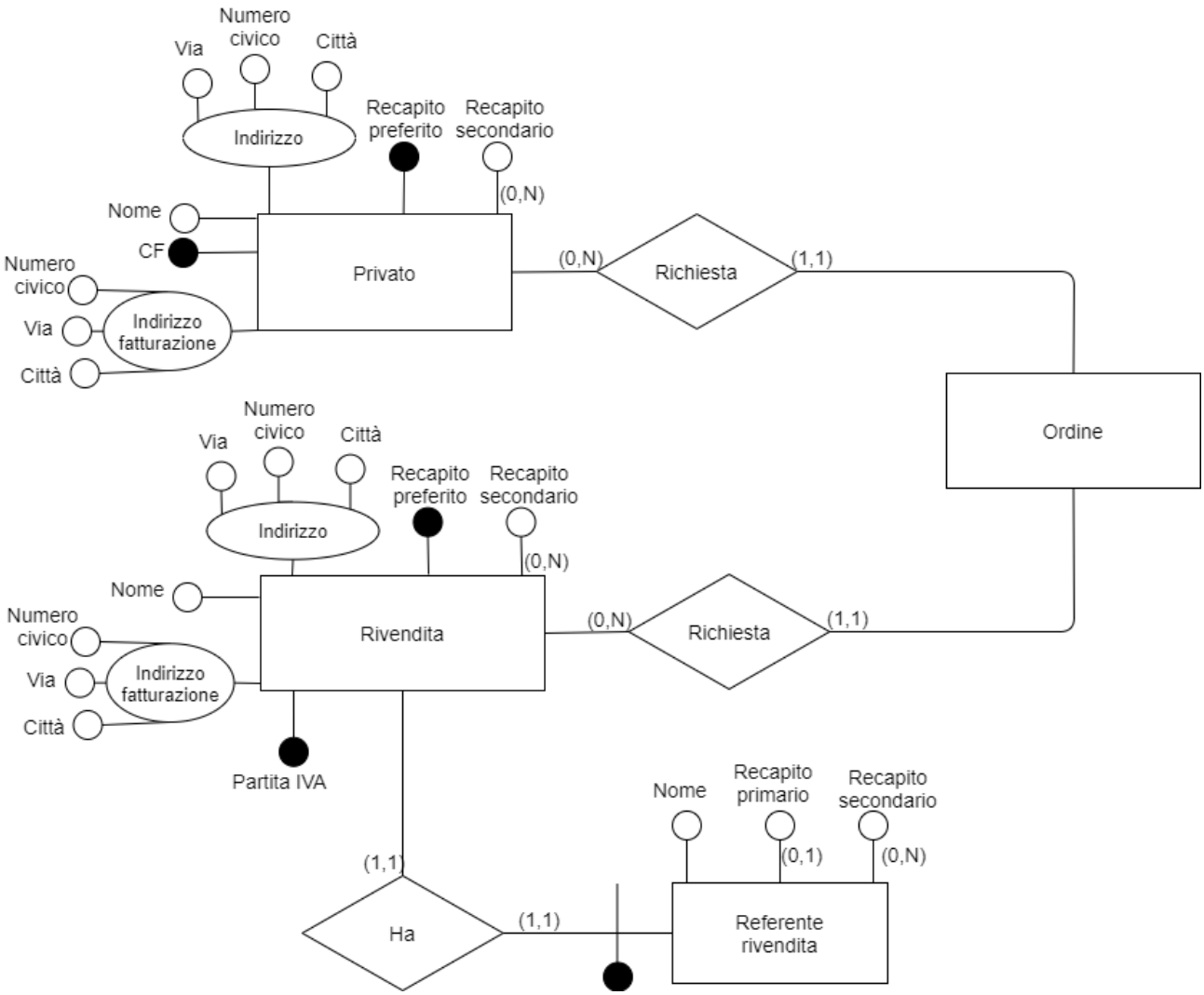
In conclusione, con la variazione di schema proposta si cercava di smovere di molto i costi invece risulta addirittura un lieve peggioramento. Inoltre, sembra che con questa proposta si sia complicata la struttura dello schema ER. Preferiamo quindi tenere la scelta iniziale.

Eliminazione delle generalizzazioni

Il diagramma E-R presenta diverse generalizzazioni che esaminiamo di seguito.

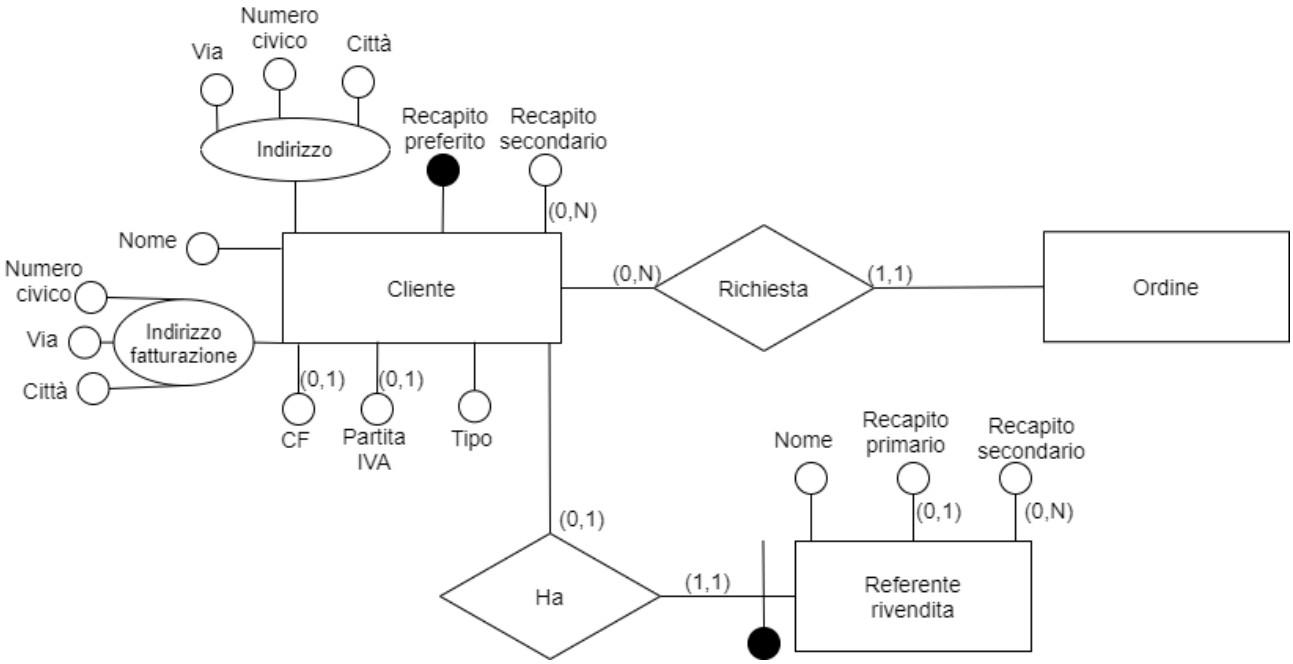
Ristrutturando Cliente con l'accorpamento del genitore della generalizzazione nelle figlie (figura che segue), non abbiamo spreco di memoria mentre se analizziamo gli accessi delle operazioni vediamo che:

- OP8 aumenta di costo perché per cercare il cliente che ha fatto l'ordine abbiamo bisogno di accedere sia a Rivendita che a Privato. Calcoliamo quindi un aumento di 100/g;
- OP2, OP3, OP20, OP21 hanno costo diminuito dalla riduzione degli accessi perché sono operazioni specifiche solo per Rivendita o solo per Privato. Risparmio tra le 4 stimato di 24/g;



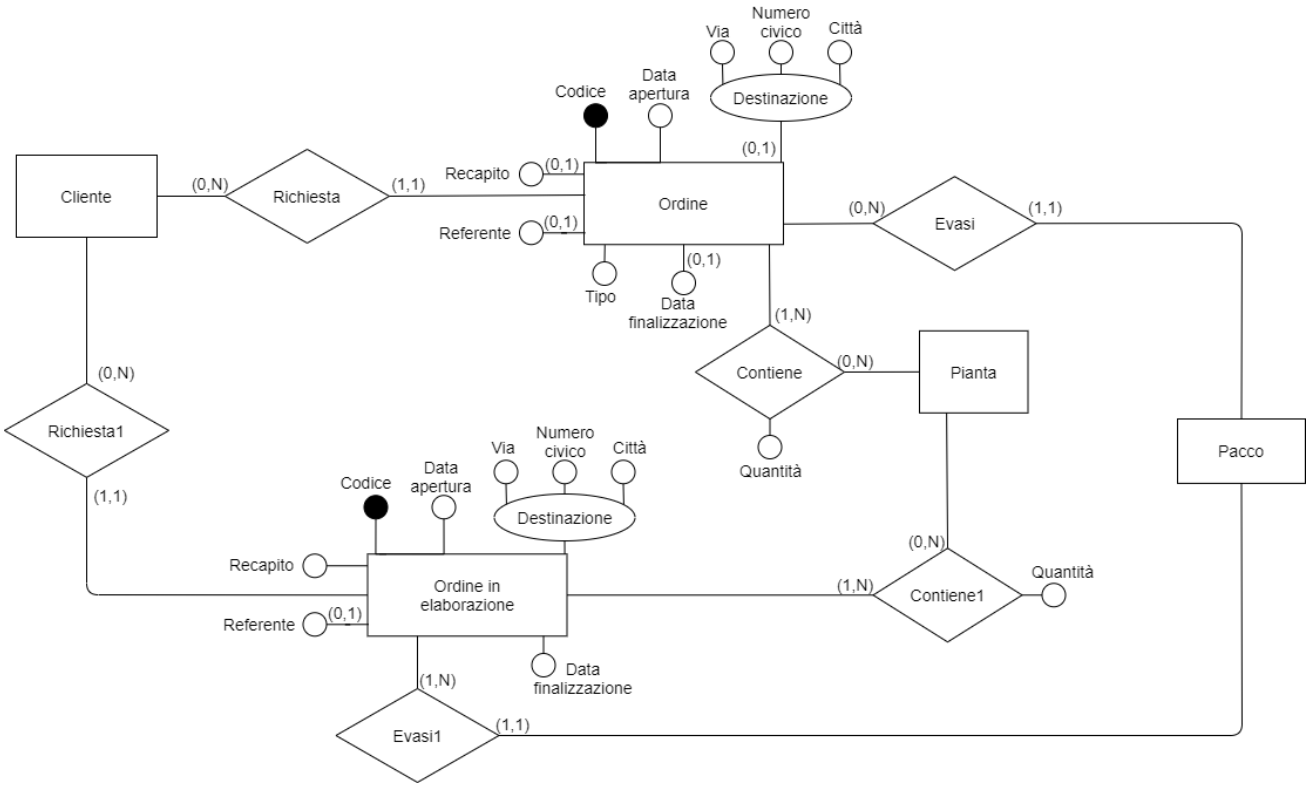
Ristrutturiamolo ora con l'accorpamento delle figlie nel genitore.
Aumenta l'utilizzo di memoria a causa della presenza di valori nulli (molto limitata) e per quanto riguarda il costo delle operazioni abbiamo:

- OP2, OP3, OP20, OP21 hanno costo diminuito dalla riduzione degli accessi perché ora Cliente contiene tutti gli attributi utili a queste operazioni. Risparmio tra le 4 stimato di 24/g;



Decidiamo quindi di utilizzare la seconda delle soluzioni analizzata per la ristrutturazione della generalizzazione. Il costo aggiuntivo di memoria sarà ridottissimo e al tempo stesso avremo un miglioramento sul costo delle operazioni.

Analizziamo ora la generalizzazione relativa a ordine. Anche in questo caso decidiamo di analizzare le differenze tra le due tipologie di ristrutturazione proposte in precedenza escludendo la sostituzione della generalizzazione con associazioni perché ritenuta poco efficiente data la generalizzazione totale.
Riportiamo di seguito la ristrutturazione che ci permette di abbattere i costi dell'operazione più costosa della nostra base di dati, la OP10.

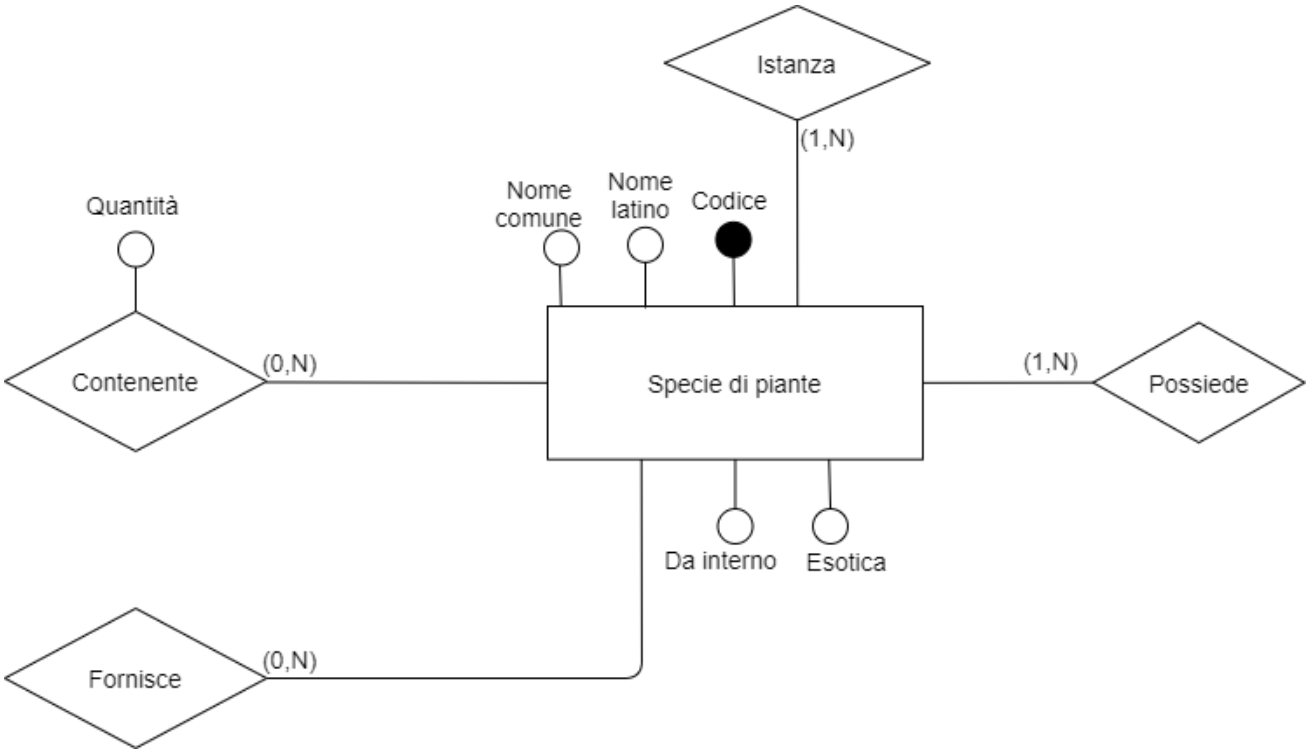


Analizziamo i costi:

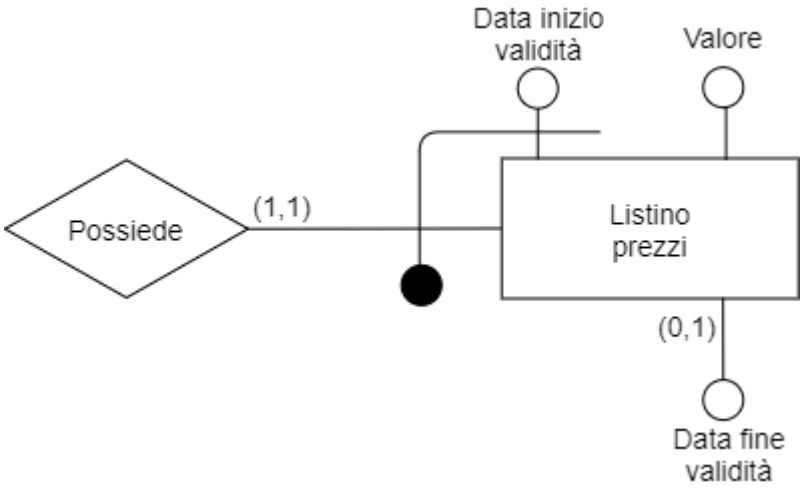
- OP10: avendo separato la specificazione Ordine in elaborazione il costo crolla a 3300/g quindi abbiamo un risparmio di 78.600-3300 = 75.300/g
- OP6: non dobbiamo più leggere ed eliminare, ora basta modificare l'attributo "tipo". Risparmiamo 4.000 – 400 = 3.600/g

Ristrutturare mettendo tutte le figlie nel genitore semplificherebbe lo schema e le operazioni OP30 e OP31 ma non permetterebbe di abbattere i costi della OP10. Mettendo a confronto preferiamo abbattere i costi dell'operazione più costosa della nostra base di dati, sacrificando altre operazioni meno costose.

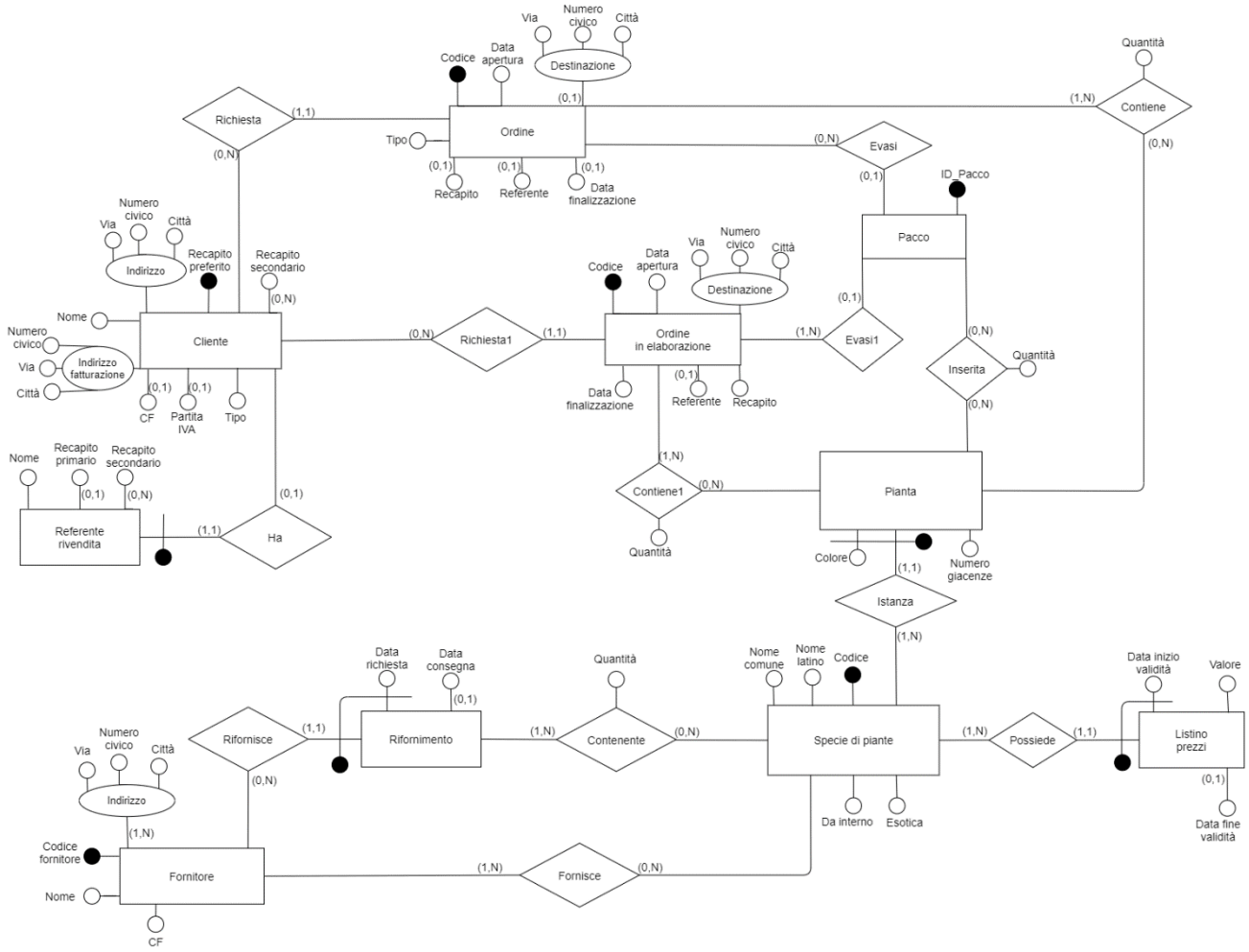
Arriviamo ora alla generalizzazione da ristrutturare che riguarda "Specie di piante". Possiamo risolvere il tutto semplicemente inserendo due attributi. È sicuramente la scelta più vantaggiosa perché le figlie di "Specie di piante" non hanno attributi e non hanno associazioni private.



Infine, anche per la generalizzazione che coinvolge “Listino prezzi” optiamo per ristrutturare le figlie nel genitore portando la cardinalità dell’attributo “Data fine validità” a (0,1).

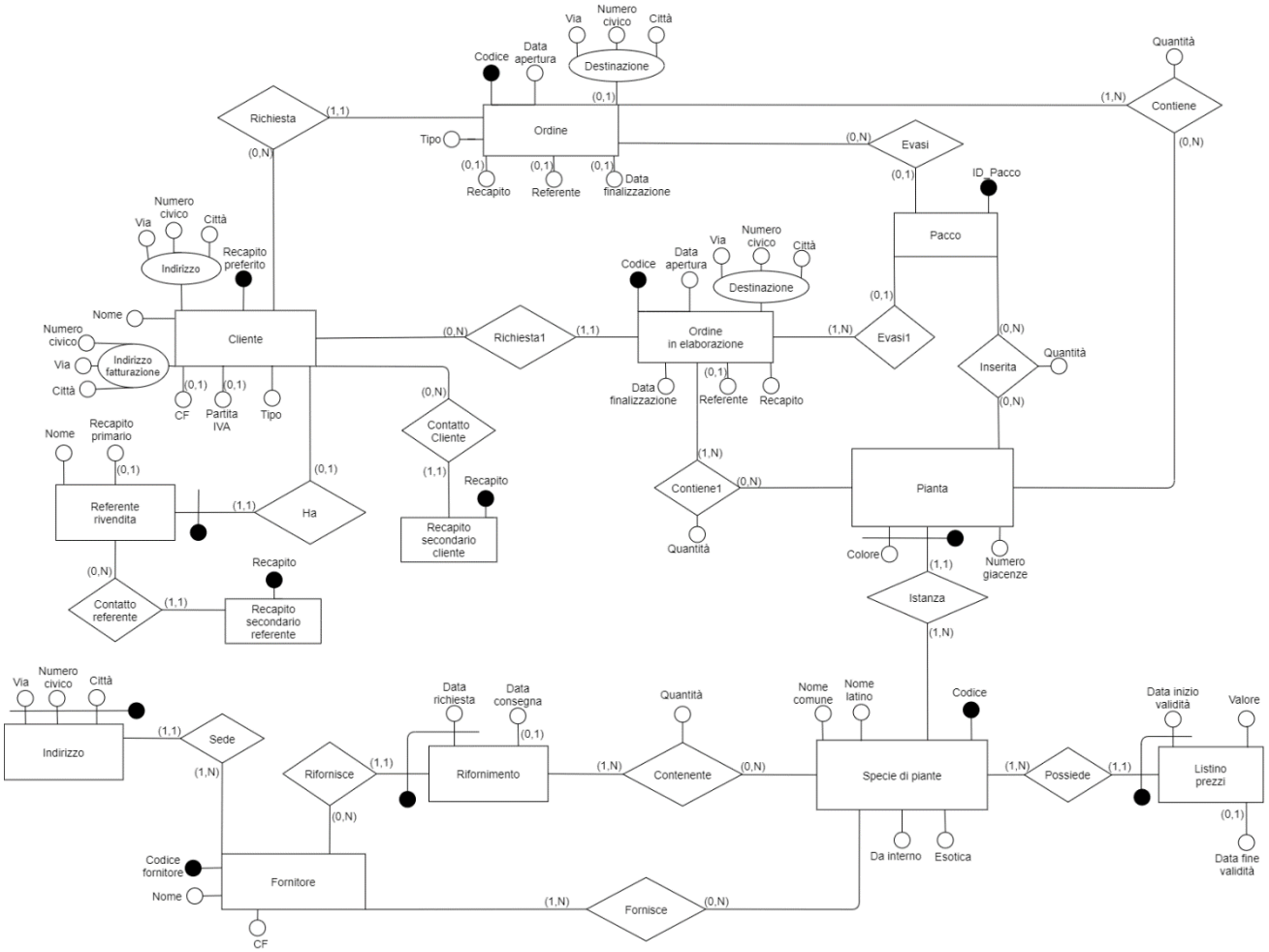


Riportiamo di seguito lo schema con le ristrutturazioni apportate fino a questo punto.



Eliminazione di attributi multivalore

Ristrutturazione necessaria perché, come per la generalizzazione, il modello relazionale non permette di rappresentare in modo diretto questo tipo di attributo.
Non ci sono analisi da fare a riguardo, la ristrutturazione di attributi multivalore è semplice e va bene in ogni caso. Viene quindi eseguita sugli attributi multivalore appartenenti a “Cliente”, “Referente rivendita” e “Fornitore”. Di seguito riporto lo schema ER con le modifiche apportate.



Traduzione di entità e associazioni

INDIRIZZO (Via, Numero civico, Città, Fornitore), vincolo di integrità referenziale tra Fornitore e la relazione FORNITORE.

FORNITORE (Codice fornitore, Nome, CF).

RIFORNIMENTO (Fornitore, Data richiesta, Data consegna*), vincolo di integrità referenziale tra Fornitore e la relazione FORNITORE.

FORNISCE (Fornitore, Specie di piante), vincolo di integrità referenziale tra Fornitore e la relazione FORNITORE e tra Specie di piante e la relazione SPECIE DI PIANTE.

CONTENENTE (Data richiesta rifornimento, Fornitore, Specie di piante, Quantità), vincolo di integrità referenziale tra Data richiesta rifornimento, Fornitore e la relazione RIFORNIMENTO e tra Specie di piante e la relazione SPECIE DI PIANTE.

SPECIE DI PIANTE (Codice, Nome comune, Nome latino, Da interno, Esotica).

LISTINO PREZZI (Data inizio validità, Codice specie, valore, Data fine validità*), vincolo di integrità referenziale tra Codice specie e la relazione SPECIE DI PIANTE.

PIANTA (Colore, Codice specie, Numero giacenze) con vincolo di integrità referenziale tra Codice specie e la relazione SPECIE DI PIANTE.

CONTIENE (Ordine, Colore pianta, Specie, Quantità), con vincolo di integrità referenziale tra Colore pianta, Specie e la relazione PIANTA e tra Ordine e la relazione ORDINE.

INSERITA (ID_Pacco, Colore pianta, Specie, Quantità), vincoli di integrità referenziale tra Colore pianta, Specie e la relazione PIANTA.

CONTIENE1 (Ordine in elaborazione, Colore pianta, Specie, Quantità), con vincoli di integrità referenziali tra Ordine in elaborazione e la relazione ORDINE IN ELABORAZIONE e tra Colore pianta, Specie e la relazione PIANTA.

ORDINE (Codice, Data apertura, Data finalizzazione*, Via*, Numero civico*, Città*, Tipo, Recapito*, Referente*, Richiesta) con vincolo di integrità referenziale tra Richiesta e la relazione CLIENTE.

EVASI (Pacco, Ordine), vincolo di integrità referenziale tra Ordine e la relazione ORDINE.

EVASI1 (Pacco, Ordine in elaborazione), vincoli di integrità referenziale tra Ordine in elaborazione e la relazione ORDINE IN ELABORAZIONE.

ORDINE IN ELABORAZIONE (Codice, Data apertura, Data finalizzazione, Via, Numero civico, Città, Recapito, Referente*, Richiesta1), vincolo di integrità referenziale tra Richiesta1 e la relazione CLIENTE.

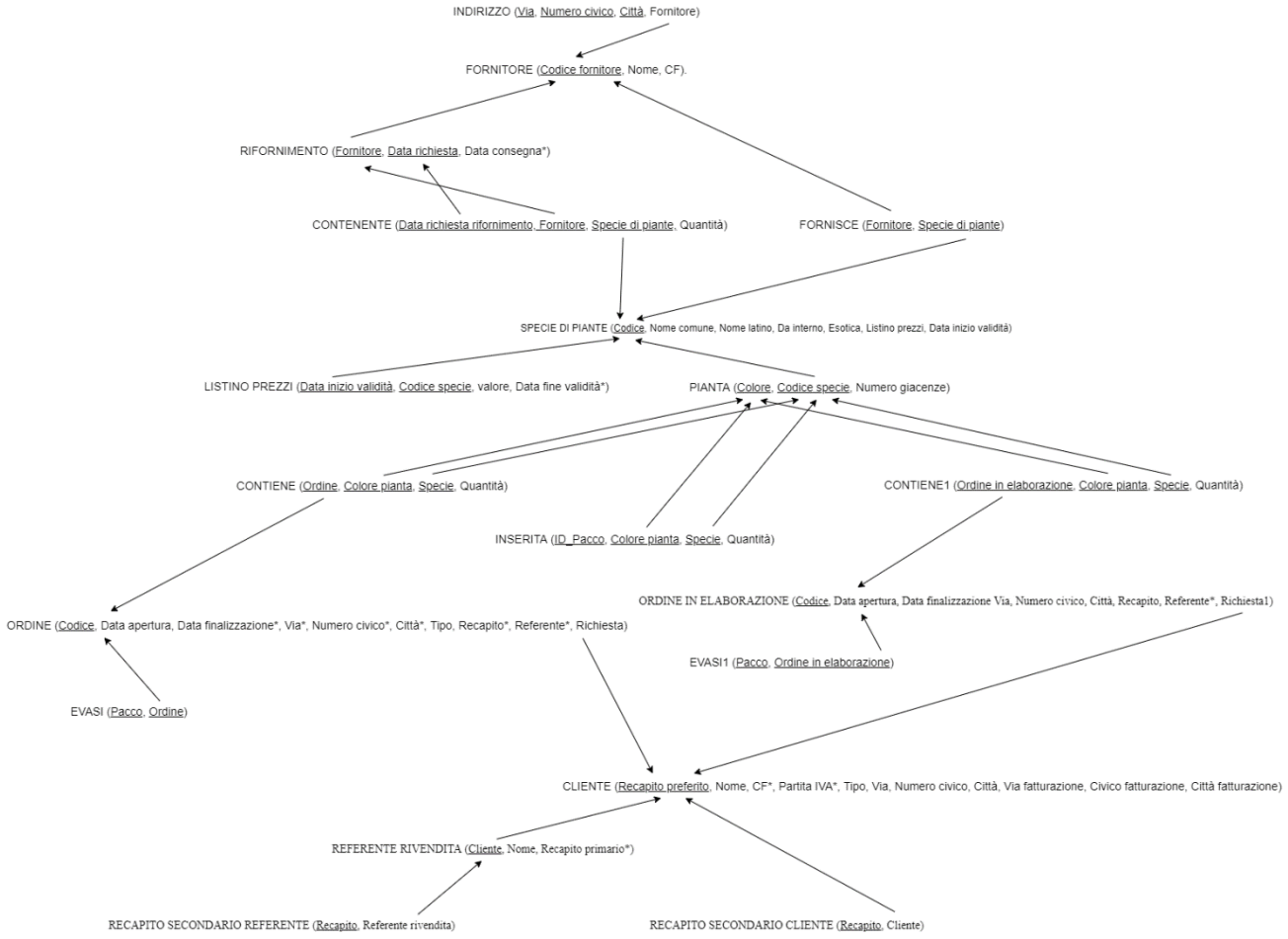
CLIENTE (Recapito preferito, Nome, CF*, Partita IVA*, Tipo, Via, Numero civico, Città, Via fatturazione, Civico fatturazione, Città fatturazione).

REFERENTE RIVENDITA (Cliente, Nome, Recapito primario*), vincolo di integrità referenziale tra Cliente e la relazione CLIENTE.

RECAPITO SECONDARIO REFERENTE (Recapito, Referente rivendita), vincolo di integrità referenziale tra Referente rivendita e la relazione REFERENTE RIVENDITA.

RECAPITO SECONDARIO CLIENTE (Recapito, Cliente), vincoli di integrità tra Cliente e la relazione CLIENTE.

Optiamo per non inserire EVASI1 ed EVASI in PACCO perché avremmo sempre uno dei due impostati a NULL e scegliamo di non tenere una tabella di nome Pacco contenente solo l'ID_Pacco perché si può ricavare dall'unione delle proiezioni sull'attributo ID_Pacco di Evasi, Evasi1 e Inserita. Non è infatti ritenuto utile inserire un pacco senza riferimento a una di queste tre relazioni.



Normalizzazione del modello relazionale

Prima forma normale

Definizione: una relazione r è in 1NF se:

- È presente una chiave primaria (non ci sono tuple duplicate);
- Non vi sono gruppi di attributi che si ripetono;
- Non vi sono attributi composti e che quindi potrebbero essere divisi.

- La base di dati è in prima forma Normale

Seconda forma normale

Definizione: una relazione è in 2NF se è in 1NF e tutti gli attributi non-chiave dipendono funzionalmente dall'intera chiave composta (ovvero la relazione non ha attributi che dipendono funzionalmente da una parte della chiave).

Questa forma mi garantisce l'assenza di numerose anomalie che imporrebbero critici vincoli di rappresentazione di dati su una determinata relazione. Quindi, nonostante la 1NF garantisce la consistenza dei dati di un modello relazionale, la 2NF è una condizione quasi indispensabile per l'applicabilità di una base di dati a situazioni della realtà.

Per verificare la 2NF bisogna prendere la relazione e osservare le anomalie che si potrebbero verificare sostituendo la chiave primaria con una sua parte.

- La base di dati è in seconda forma normale

Terza forma normale

Definizione: una relazione è in 3NF se è in 2NF e tutti gli attributi non-chiave dipendono dalla chiave soltanto (ovvero non vi sono dipendenze transitive tra gli attributi).

Per verificare la 3NF considero tutte le dipendenze funzionali e vedo se ci sono dipendenze transitive.

- La base di dati è in terza forma normale

Forma normale di Boyce e Codd

Definizione: una relazione è in BCNF se è in 3NF e se, per ogni dipendenza funzionale non banale $X \rightarrow Y$ (cioè con Y non in X), X è una superchiave.

- La base di dati è in forma normale di Boyce e Codd

5. Progettazione fisica

Utenti e privilegi

Sono previsti 6 utenti nell'applicazione: login, cliente, amministratore, gestore magazzino, manager, operatore. Ad ognuno di questi è vietato l'accesso alle tabelle del database e ognuno ha un numero di privilegi (consistenti nelle procedure che possono eseguire) ritagliati appositamente sul ruolo che svolgono.
Quando ci si connette al DB all'apertura dell'applicazione lo si fa con i dati dell'utente Login. Questo utente è stato implementato all'interno al DB con il solo privilegio di eseguire la procedura "login", che permette all'utente reale di inserire username e password per identificarsi ed essere indirizzato a un menù fatto apposta per il ruolo che gli compete. Di seguito riporto gli utenti del DB con relativi privilegi.
Cliente: gestisce gli ordini eseguendo le procedure
Amministratore: crea utenti
Gestore magazzino: gestisce le giacenze e i rifornimenti
Manager: gestisce le specie di piante trattate e il listino prezzi
Operatore: si occupa dell'inserimento delle piante nei pacchi

Strutture di memorizzazione

Tabella <Cliente>		
Attributo	**Tipo di dato**	**Attributi[2]**
Recapito preferito	VARCHAR(45)	PK, NN.
Nome	VARCHAR(45)	NN.
Codice fiscale	CHAR(16)	UQ.
Partita IVA	CHAR(11)	UQ.
Tipo	ENUM('privato','rivendita')	NN.
Via	VARCHAR(45)	NN.
Numero civico	INT	NN, UN.
Città	VARCHAR(45)	NN.
Via fatturazione	VARCHAR(45)	NN.
Civico fatturazione	INT	NN, UN.
Username	VARCHAR(45)	NN, UQ.

[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.

Tabella <Contenente>		
Attributo	**Tipo di dato**	**Attributi**
Fornitore	INT	PK, NN, UN.
Data richiesta rifornimento	DATETIME	PK, NN.
Specie di piante	VARCHAR(45)	PK, NN.
Quantità	INT	NN, UN.

Tabella <Contiene>		
Attributo	**Tipo di dato**	**Attributi**
Ordine	INT	PK, NN, UN.
Colore pianta	VARCHAR(45)	PK, NN.
Specie	VARCHAR(45)	PK, NN.
Quantità	INT	NN, UN.

Tabella <Contiene1>		
Attributo	**Tipo di dato**	**Attributi**
Ordine in elaborazione	INT	PK, NN, UN.
Colore pianta	VARCHAR(45)	PK, NN.
Specie	VARCHAR(45)	PK, NN.
Quantità	INT	NN, UN.

Tabella <Evasi>		
Attributo	**Tipo di dato**	**Attributi**
Pacco	INT	PK, NN, UN.
Ordine	INT	PK, NN, UN.

Tabella <Evasi1>		
Attributo	**Tipo di dato**	**Attributi**
Pacco	INT	PK, NN, UN.
Ordine in elaborazione	INT	PK, NN, UN.

Tabella <Fornisce>		
Attributo	**Tipo di dato**	**Attributi**
Fornitore	INT	PK, NN, UN.
Specie di piante	VARCHAR(45)	PK, NN.

Tabella <Fornitore>		
Attributo	**Tipo di dato**	**Attributi**
Codice fornitore	INT	PK, NN, UN.
Nome	VARCHAR(45)	NN.
Codice fiscale	CHAR(16)	NN, UQ.
Quantità	INT	NN, UN.

Tabella <Indirizzo>		
Attributo	**Tipo di dato**	**Attributi**
Via	VARCHAR(45)	PK, NN.
Numero civico	INT	PK, NN, UN.
Città	VARCHAR(45)	PK, NN.
Fornitore	INT	NN, UN.

Tabella <Inserita>		
Attributo	**Tipo di dato**	**Attributi**
Id_pacco	INT	PK, NN, UN.
Colore pianta	VARCHAR(45)	PK, NN.
Specie	VARCHAR(45)	PK, NN.
Quantità	INT	NN, UN.

Tabella <Listino prezzi>

Attributo	Tipo di dato	Attributi
Data inizio validità	DATETIME	PK, NN.
Codice specie	VARCHAR(45)	PK, NN.
Valore	FLOAT	NN, UN.
Data fine validità	DATETIME	

Tabella <Ordine>		
Attributo	**Tipo di dato**	**Attributi**
Codice	INT	PK, NN, UN, AI.
Data apertura	DATETIME	NN.
Data finalizzazione	DATETIME	
Via	VARCHAR(45)	
Numero civico	INT	UN
Città	VARCHAR(45)	
Recapito	VARCHAR(45)	
Referente	VARCHAR(45)	
Tipo	ENUM('aperto','finalizzato','spedito')	NN.
Richiesta	VARCHAR(45)	NN.

Tabella <Ordine in elaborazione>		
Attributo	**Tipo di dato**	**Attributi**
Codice	INT	PK, NN, UN, AI.
Data apertura	DATETIME	NN.
Data finalizzazione	DATETIME	NN.
Via	VARCHAR(45)	NN.
Numero civico	INT	NN, UN.
Città	VARCHAR(45)	NN.
Recapito	VARCHAR(45)	NN.
Referente	VARCHAR(45)	
Richiesta1	VARCHAR(45)	NN.

Tabella <Pianta>		
Attributo	**Tipo di dato**	**Attributi**
Colore	VARCHAR(45)	PK, NN.
Codice specie	VARCHAR(45)	PK, NN.
Numero giacenze	INT	NN, UN.

Tabella <Recapito secondario cliente>		
Attributo	**Tipo di dato**	**Attributi**
Recapito	VARCHAR(45)	PK, NN.
Cliente	VARCHAR(45)	NN.

Tabella <Recapito secondario referente>		
Attributo	**Tipo di dato**	**Attributi**
Recapito	VARCHAR(45)	PK, NN.
Referente rivendita	VARCHAR(45)	NN.

Tabella <Referente rivendita>		
Attributo	**Tipo di dato**	**Attributi**
Cliente	VARCHAR(45)	PK, NN.
Nome	VARCHAR(45)	NN.
Recapito primario	VARCHAR(45)	

Tabella <Rifornimento>		
Attributo	**Tipo di dato**	**Attributi**
Fornitore	INT	PK, NN, UN.
Data richiesta	DATETIME	PK, NN.
Data consegna	DATETIME	

Tabella <Specie di piante>		
Attributo	**Tipo di dato**	**Attributi**
Codice	VARCHAR(45)	PK, NN.
Nome comune	VARCHAR(45)	NN, UQ.
Nome latino	VARCHAR(45)	NN, UQ.
Da interno	TINYINT	NN.
Esotica	TINYINT	NN.

Tabella <Utenti>		
Attributo	**Tipo di dato**	**Attributi**
Username	VARCHAR(45)	PK, NN.
Password	CHAR(32)	NN.
Ruolo	ENUM('amministratore', 'cliente', 'gestore_magazzino', 'manager', 'operatore')	NN.

Indici

Indici presenti nel modello:

- Indici PRIMARY vengono automaticamente generati da mysql e sono detti "clustered index" e che fanno rispettare l'ordine delle righe nella tabella. Questo tipo di indice consente di identificare univocamente e direttamente ogni singolo record all'interno di una tabella.
- Indici UNIQUE come i precedenti consentono di identificare univocamente e direttamente ogni singolo record all'interno di una tabella, ma ammettono anche valori NULL.
- Indici di tipo Foreign Key vengono creati per ogni foreign key per migliorare le performance nel caso in cui vengano fatte le operazioni di join.

Non sono stati aggiunti indici oltre a quelli autogenerati da Workbench all'inserimento di queste marcature sugli attributi di una tabella.

Trigger

Il seguente trigger viene utilizzato per modificare il numero di piante disponibili ogni volta che viene eliminato un ordine aperto che conteneva piante momentaneamente riservate.

```
CREATE DEFINER = CURRENT_USER TRIGGER `ingrosso-piante`.`contiene_AFTER_DELETE` AFTER DELETE ON `ingrosso-piante`.`contiene` FOR EACH ROW
BEGIN
	update pianta
		set numero_giacenze = numero_giacenze + OLD.quantità
	where codice_specie = OLD.specie  and colore = OLD.colore_pianta;
END
```

Eventi

Il seguente evento è istanziato in fase di configurazione del sistema. Fa' si che un ordine aperto venga automaticamente chiuso dopo 12 ore. Ciò perché una variazione di prezzo non interferisce su un ordine aperto e si evita così di mantenere ordini aperti per lungo tempo e conservare un prezzo ormai cambiato da tempo.

```
set global event_scheduler = on;
create event if not exists `clean_order`
	on schedule every 1 hour on completion preserve
  comment 'Remove order open for more than 12 hours'
  do
		delete from `ordine`
			where `tipo` = 'aperto' and `data_apertura` < (now() - interval 12 hour)
```

Viste

Vista che ritorna gli ordini aperti. Viene utilizzata per semplificare la lettura delle procedure che la utilizzano.

```
CREATE VIEW `ordini_aperti` AS select `codice`, `data_apertura`, `richiesta` as 'recapito cliente'
	from `ordine`
  where `tipo` = 'aperto';
```

Stored Procedures e transazioni

Le seguenti procedure sono l'unico modo di interazione tra il client e il database. A seconda del tipo di utente connesso si ha la possibilità di eseguire un numero ristretto di procedure.

In generale, in questa implementazione non ho utilizzato transazioni in caso di un'unica query di insert o update o delete.

Ho utilizzato un livello di isolamento READ UNCOMMITTED quando ho voluto utilizzare le proprietà ACID delle transazioni senza avere problematiche derivanti dalle anomalie. Ciò è avvenuto nel caso di gruppi di query che non includevano select.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `aggiungi_cliente`(in var_username varchar(45), in var_tipo varchar(20), in var_recapito_preferito varchar(45), in var_nome varchar(45), in var_codice_fiscale char(16), in var_partita_IVA char(11), in var_via varchar(45), in var_numero_civico int, in var_città varchar(45), in var_via_fatturazione varchar(45), in var_civico_fatturazione int, in var_città_fatturazione varchar(45), in var_referente varchar(45), in var_recapito_referente varchar(45))
BEGIN
	declare exit handler for sqlexception
  begin
    resignal;  -- raise again the sql exception to the caller
  end;

  -- verifico se il recapito corrisponde a un telefono o a un'email
	if (not `valid_email` (var_recapito_preferito) and not `valid_phone` (var_recapito_preferito)) then
		signal sqlstate '45001'
    set message_text = "Il recapito inserito non sembra corrispondere né a un'email né a un numreo di telefono";
	end if;

  set transaction isolation level read uncommitted;
	start transaction;
		if var_tipo = 'rivendita' then
			insert into `cliente` (`recapito_preferito`, `nome`, `partita_IVA`, `tipo`, `via`, `numero_civico`, `città`, `via_fatturazione`, `civico_fatturazione`, `città_fatturazione`, `username`) values (var_recapito_preferito, var_nome, var_partita_IVA, 'rivendita', var_via, var_numero_civico, var_città, var_via_fatturazione, var_civico_fatturazione, var_città_fatturazione, var_username);
			if var_recapito_referente = '-' then
				insert into `referente_rivendita` (`cliente`, `nome`) values (var_recapito_preferito, var_referente);
			else
				insert into `referente_rivendita` values (var_recapito_preferito, var_referente, var_recapito_referente);
			end if;
		else
```

```
                    insert into `cliente` (`recapito_preferito`, `nome`, `codice_fiscale`, `tipo`, `via`, `numero_civico`, `città`, `via_fatturazione`, `civico_fatturazione`, `città_fatturazione`, `username`) values (var_recapito_preferito, var_nome, var_codice_fiscale, 'privato', var_via, var_numero_civico, var_città, var_via_fatturazione, var_civico_fatturazione, var_città_fatturazione, var_username);
                end if;
    commit;
END

CREATE DEFINER=`root`@`localhost` PROCEDURE `aggiungi_contatto_cliente`(in var_username varchar(45), in var_contatto varchar(45))
BEGIN
        declare var_cliente varchar(45);
        declare exit handler for sqlexception
    begin
      resignal;  -- raise again the sql exception to the caller
    end;

    -- verifico se il recapito corrisponde a un telefono o a un'email
        if (not `valid_email` (var_contatto) and not `valid_phone` (var_contatto)) then
            signal sqlstate '45001'
      set message_text = "Il recapito inserito non sembra corrispondere né a un'email né a un numreo di telefono";
        end if;

    set transaction isolation level read committed;
    start transaction;
    select `recapito_preferito` from `cliente` where var_username = `username` into var_cliente;
    insert into `recapito_secondario_cliente` values (var_contatto, var_cliente);
    commit;
END

CREATE DEFINER=`root`@`localhost` PROCEDURE `aggiungi_fornitore`(in var_codice_fornitore int, in var_nome varchar(45),in var_codice_fiscale char(16), in var_via varchar(45), in var_numero_civico int, in var_città varchar(45), in var_specie_fornita varchar(45))
BEGIN
        set transaction isolation level read uncommitted;
    start transaction;
        insert into `fornitore` values (var_codice_fornitore, var_nome, var_codice_fiscale);
    insert into `indirizzo` values (var_via, var_numero_civico, var_città, var_codice_fornitore);
```

```
    insert into `fornisce` values (var_codice_fornitore, var_specie_fornita);
    commit;
END

CREATE DEFINER=`root`@`localhost` PROCEDURE `aggiungi_indirizzo_fornitore`(in var_via varchar(45), in var_numero_civico int, in var_città varchar(45), in var_codice_fornitore int)
BEGIN
        insert into `indirizzo` values (var_via, var_numero_civico, var_città, var_codice_fornitore);
END

CREATE DEFINER=`root`@`localhost` PROCEDURE `aggiungi_pianta_ordine`(in var_ordine int, in var_nome varchar(45), in var_colore varchar(45), in var_quantità int)
BEGIN
        declare var_specie varchar(45);
        declare exit handler for sqlstate '22003'   -- catturo segnale generato dall'inserimento di un numero negativo in un UNSIGNED
    begin
      rollback;  -- rollback any changes made in the transaction
      resignal;  -- raise again the sql exception to the caller
    end;

    set transaction isolation level read committed;
        start transaction;
                select `codice`
                        from `specie_di_piante`
        where `nome_comune` = var_nome
        into var_specie;

                update `pianta`
                        set `numero_giacenze` = `numero_giacenze` - var_quantità
                        where `colore` = var_colore and `codice_specie` = var_specie;  -- verifica il codice di errore tornato al client in caso di sottrazione con risultato negativo e scrivi "prodotto non disponibile"

                insert into `contiene` values (var_ordine, var_colore, var_specie, var_quantità);
        commit;
END
```

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `aggiungi_specie`(in var_codice varchar(45), in var_nome_comune varchar(45), in var_nome_latino varchar(45), in var_da_interno tinyint, in var_esotica tinyint, in var_prezzo float, in var_array_colore varchar(255))  -- corrent usage var_array_colore: rosso.bianco.blu.

BEGIN

	declare var_num_specie int;

	declare var_start_index int default 1;

   declare var_end_index int;

	set transaction isolation level read uncommitted;

   start transaction;

		insert into `specie_di_piante` values (var_codice, var_nome_comune, var_nome_latino, var_da_interno, var_esotica);

     insert into `listino_prezzi` (`data_inizio_validità`, `codice_specie`, `valore`) values (now(), var_codice, var_prezzo);


     if (var_array_colore = '-') then     -- correct usage: se non ha colorazioni immetti var_array_colore = '-'

			insert into `pianta` (`codice_specie`) values (var_codice);

		else

			select locate('.', var_array_colore) into var_end_index;


			while var_end_index <> 0 do

				insert into `pianta` (`colore`, `codice_specie`) values (substring(var_array_colore from var_start_index for (var_end_index - var_start_index)), var_codice);

				set var_start_index = var_end_index + 1;

				select locate('.', var_array_colore, var_start_index) into var_end_index;

			end while;

		end if;

	commit;

END



CREATE DEFINER=`root`@`localhost` PROCEDURE `aggiungi_specie_fornita`(in var_fornitore int, in var_specie varchar(45))

BEGIN

	insert into `fornisce` values (var_fornitore, var_specie);

END
```

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `crea_ordine`(in var_username varchar(45), in var_colore_pianta varchar(45), in var_nome_specie varchar(45), in var_quantità int, out var_codice_ordine int)
BEGIN
	declare var_cliente varchar(45);
	declare var_specie varchar(45);
	declare exit handler for sqlstate '22003'
  begin
    rollback;  -- rollback any changes made in the transaction
    resignal;  -- raise again the sql exception to the caller
  end;

  set transaction isolation level read committed;
	start transaction;
		select `codice`
    from `specie_di_piante`
    where `nome_comune` = var_nome_specie
    into var_specie;

		select `recapito_preferito`
			from `cliente`
      where `username` = var_username
      into var_cliente;

		update `pianta`
			set `numero_giacenze` = `numero_giacenze` - var_quantità
			where `colore` = var_colore_pianta and `codice_specie` = var_specie;

		insert into `ordine` (`data_apertura`, `tipo`, `richiesta`) values (now(), 'aperto', var_cliente);
		set var_codice_ordine = last_insert_id();

		insert into `contiene` values (var_codice_ordine, var_colore_pianta, var_specie, var_quantità);

  commit;
END

CREATE DEFINER=`root`@`localhost` PROCEDURE `crea_utente`(IN username VARCHAR(45), IN pass VARCHAR(45), IN ruolo varchar(45))
```

```
BEGIN
	insert into utenti VALUES(username, MD5(pass), ruolo);
END


CREATE DEFINER=`root`@`localhost` PROCEDURE `elimina_ordine_aperto` (in var_ordine int, in var_username varchar(45))
BEGIN
	declare var_cliente varchar(45);


	declare exit handler for sqlexception
  begin
    rollback;  -- rollback any changes made in the transaction
    resignal;  -- raise again the sql exception to the caller
  end;


	set transaction isolation level read committed;
  start transaction;
		select `recapito_preferito`
			from `cliente`
			where `username` = var_username
	    into var_cliente;


		if ((select `tipo` from `ordine` where `codice` = var_ordine) <> 'aperto') then
			signal sqlstate '45002' set message_text = "Non puoi eliminare un ordine già confermato";
		end if;


    if (var_ordine not in (select `codice` from `ordini_aperti` where `recapito cliente` = var_cliente)) then
			signal sqlstate '45003' set message_text = "Non puoi eliminare un ordine non tuo";
		end if;


    delete from `contiene` where `ordine` = var_ordine;  -- attiva il trigger che libera le quantità di piante momentaneamente riservate
		delete from `ordine` where `codice` = var_ordine;
  commit;
END
```

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `finalizza_ordine`(in var_ordine int, in var_via varchar(45), in var_numero_civico int, in var_città varchar(45), in var_recapito varchar(45), in var_referente varchar(45), in var_username varchar(45))

BEGIN

	declare exit handler for sqlexception

  begin

    rollback;  -- rollback any changes made in the transaction

    resignal;  -- raise again the sql exception to the caller

  end;


	set transaction isolation level read committed;

  start transaction;

	if ((select `username` from `cliente` where `recapito_preferito` in (select `richiesta` from `ordine` where `codice` = var_ordine)) <> var_username) then

		signal sqlstate '45003' set message_text = "Non ci provare: puoi finalizzare un ordine non tuo!";

	end if;

  if (var_referente = '-') then

		update `ordine` set `data_finalizzazione` = now(), `via` = var_via, `numero_civico` = var_numero_civico, `città` = var_città, `recapito` = var_recapito, `tipo` = 'finalizzato' where `codice` = var_ordine;

	else

		update `ordine` set `data_finalizzazione` = now(), `via` = var_via, `numero_civico` = var_numero_civico, `città` = var_città, `recapito` = var_recapito, `referente` = var_referente, `tipo` = 'finalizzato' where `codice` = var_ordine;

	end if;

	commit;


END


CREATE PROCEDURE `lista_speci_ordinate` ()

BEGIN

	set transaction isolation level read committed;

  start transaction;

		select `codice`, `nome_comune`, `nome_latino`

			from `specie_di_piante`

			order by `nome_comune`;

	commit;

END
```

```
CREATE PROCEDURE `listini_passati` (in var_nome varchar(45))
BEGIN
	declare var_specie varchar(45);

  select `codice`
		from `specie_di_piante`
    where `nome_comune` = var_nome
    into var_specie;

	select `valore`, `data_inizio_validità`, `data_fine_validità`
		from `listino_prezzi`
    where `codice_specie` = var_specie and `data_fine_validità` is not NULL;

END

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('amministratore', 'cliente', 'gestore_magazzino', 'manager', 'operatore');

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

		if var_user_role = 'amministratore' then
			set var_role = 1;
		elseif var_user_role = 'cliente' then
			set var_role = 2;
		elseif var_user_role = 'gestore_magazzino' then
			set var_role = 3;
		elseif var_user_role = 'manager' then
			set var_role = 4;
		elseif var_user_role = 'operatore' then
			set var_role = 5;
		else
```

```
                set var_role = 6;
            end if;
END

CREATE PROCEDURE `nuovo_listino` (in var_specie varchar(45), in var_nuovo_valore float)
BEGIN
    set transaction isolation level read uncommitted;
  start transaction;
            update `listino_prezzi`
                set `data_fine_validità` = now()
                where `codice_specie` = var_specie and `data_fine_validità` is NULL;

            insert into `listino_prezzi` (`data_inizio_validità`, `codice_specie`, `valore`) values (now(), var_specie, var_nuovo_valore);
    commit;
END

CREATE DEFINER=`root`@`localhost` PROCEDURE `piante_specie`(in var_nome varchar(45), out var_costo float)
BEGIN
    declare var_specie varchar(45);
    set transaction isolation level read committed;
  start transaction;
            select `codice`
                from `specie_di_piante`
        where `nome_comune` = var_nome
        into var_specie;

            select `valore`
                from `listino_prezzi`
        where `codice_specie` = var_specie and `data_fine_validità` is NULL
        into var_costo;

            select `colore`, `numero_giacenze`
                from `pianta`
        where `codice_specie` = var_specie;
    commit;
```

```
END


CREATE DEFINER=`root`@`localhost` PROCEDURE `richiesta_rifornimento`(in var_fornitore int, in var_array_specie varchar(255), in var_array_quantità varchar(255))  -- es. array rosa.ciclamino
BEGIN
	declare var_start_index int default 1;
    declare var_end_index int;
    declare var_start_index_num int default 1;
    declare var_end_index_num int;


	set transaction isolation level read uncommitted;
    start transaction;
		insert into `rifornimento` (`fornitore`, `data_richiesta`) values (var_fornitore, now());


      select locate('.', var_array_specie) into var_end_index;
      select locate('.', var_array_quantità) into var_end_index_num;
      while var_end_index <> 0 do
			insert into `contenente` values (var_fornitore, now(), substring(var_array_specie from var_start_index for (var_end_index - var_start_index)), cast(substring(var_array_quantità from var_start_index_num for (var_end_index_num - var_start_index_num)) as unsigned));
		set var_start_index = var_end_index + 1;
        set var_start_index_num = var_end_index_num + 1;
        select locate('.', var_array_specie, var_start_index) into var_end_index;
        select locate('.', var_array_quantità, var_start_index_num) into var_end_index_num;
		end while;
      commit;
END


CREATE PROCEDURE `specie_da_rifornire` ()
BEGIN


    drop temporary table if exists `giacenza_specie`;
	create temporary table `giacenza_specie` (
		`codice_specie` varchar(45),
      `giacenze` int
	);
```

```
        set transaction isolation level read committed;
        start transaction;
                insert into `giacenza_specie`
                        select `codice_specie`, sum(numero_giacenze)
                        from `pianta`
                        group by `codice_specie`;

                delete from `giacenza_specie`
                        where `codice_specie` in (select `specie_di_piante`
                                                  from `rifornimento` join `contenente` on `rifornimento`.`fornitore` = `contenente`.`fornitore`
                                                  where `data_consegna` is null);

                select `codice_specie`
                        from `giacenza_specie`
                        where `giacenze` = 0;

        commit;
END

CREATE DEFINER=`root`@`localhost` PROCEDURE `visualizza_info_ordine`(in var_ordine int)
BEGIN
        set transaction read only;
    set transaction isolation level read committed;
    start transaction;
        select `data_finalizzazione`, `colore_pianta`, `nome_comune`, `valore`
                from `ordine` join `contiene` on `ordine`.`codice` = `ordine`
                        join `pianta` on `colore_pianta` = `colore` and `specie` = `codice_specie`
          join `specie_di_piante` on `codice_specie` = `specie_di_piante`.`codice`
          join `listino_prezzi` on `specie_di_piante`.`codice` = `listino_prezzi`.`codice_specie` and ((timestampdiff(second, `data_inizio_validità`, `data_apertura`) > 0 and timestampdiff(second, `data_apertura`, `data_fine_validità`) > 0 ) or (timestampdiff(second, `data_inizio_validità`, `data_apertura`) > 0 and `data_fine_validità` is NULL))
        where var_ordine = `ordine`.`codice`;
        commit;
END

CREATE DEFINER=`root`@`localhost` PROCEDURE `visualizza_ordini`(in var_username varchar(45))
```

```
BEGIN
        declare var_cliente varchar(45);
        set transaction read only;
  set transaction isolation level read committed;
  start transaction;
            select `recapito_preferito`
                from `cliente`
                where `username` = var_username
      into var_cliente;

            select `codice`, `data_apertura`, `tipo`
                from `ordine`
      where `richiesta` = var_cliente;
        commit;
END

CREATE DEFINER=`root`@`localhost` PROCEDURE `visualizza_ordini_aperti`(in var_username varchar(45))
BEGIN
        declare var_cliente varchar(45);
        set transaction read only;
  set transaction isolation level read committed;
  start transaction;
            select `recapito_preferito`
                from `cliente`
      where `username` = var_username
      into var_cliente;
            select `codice`, `data_apertura` from ordini_aperti;
        commit;
END
```

Funzioni

Funzioni chiamate all'interno delle procedure per verificare l'inserimento di recapiti.

```
delimiter !
create function `valid_email`(email varchar(45))
returns bool
```

```
deterministic
begin
        if email regexp '^[a-zA-Z0-9][a-zA-Z0-9._-]*[a-zA-Z0-9._-]@[a-zA-Z0-9][a-zA-Z0-9._-]*[a-zA-Z0-9]\\.[a-zA-Z]{2,63}$' then
                return true;
   end if;
   return false;
end!
delimiter ;

delimiter !
create function `valid_phone`(telefono varchar(45))
returns boolean
deterministic
begin
        if telefono regexp '[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]'   then
                return true;
   end if;
   return false;
end!
delimiter ;
```

Appendice: Implementazione

Codice SQL per instanziare il database

```
-- MySQL Workbench Forward Engineering

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 ingrosso-piante
-- -----------------------------------------------------
DROP SCHEMA IF EXISTS `ingrosso-piante` ;

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

-- -----------------------------------------------------
-- Table `ingrosso-piante`.`utenti`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `ingrosso-piante`.`utenti` ;

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

```
-- -----------------------------------------------------
-- Table `ingrosso-piante`.`cliente`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `ingrosso-piante`.`cliente` ;

CREATE TABLE IF NOT EXISTS `ingrosso-piante`.`cliente` (
  `recapito_preferito` VARCHAR(45) NOT NULL,
  `nome` VARCHAR(45) NOT NULL,
  `codice_fiscale` CHAR(16) NULL DEFAULT NULL,
  `partita_IVA` CHAR(11) NULL DEFAULT NULL,
  `tipo` ENUM('privato', 'rivendita') NOT NULL,
  `via` VARCHAR(45) NOT NULL,
  `numero_civico` INT UNSIGNED NOT NULL,
  `città` VARCHAR(45) NOT NULL,
  `via_fatturazione` VARCHAR(45) NOT NULL,
  `civico_fatturazione` INT UNSIGNED NOT NULL,
  `città_fatturazione` VARCHAR(45) NOT NULL,
  `username` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`recapito_preferito`),
  CONSTRAINT `fk_cliente_1`
    FOREIGN KEY (`username`)
    REFERENCES `ingrosso-piante`.`utenti` (`username`)
    ON DELETE RESTRICT
    ON UPDATE RESTRICT)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;

CREATE UNIQUE INDEX `username_UNIQUE` ON `ingrosso-piante`.`cliente` (`username` ASC) VISIBLE;

CREATE UNIQUE INDEX `codice_fiscale_UNIQUE` ON `ingrosso-piante`.`cliente` (`codice_fiscale` ASC) VISIBLE;

CREATE UNIQUE INDEX `partita_IVA_UNIQUE` ON `ingrosso-piante`.`cliente` (`partita_IVA` ASC) VISIBLE;

CREATE INDEX `fk_cliente_1_idx` ON `ingrosso-piante`.`cliente` (`username` ASC) VISIBLE;
```

```
-- -----------------------------------------------------
-- Table `ingrosso-piante`.`specie_di_piante`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `ingrosso-piante`.`specie_di_piante` ;

CREATE TABLE IF NOT EXISTS `ingrosso-piante`.`specie_di_piante` (
  `codice` VARCHAR(45) NOT NULL,
  `nome_comune` VARCHAR(45) NOT NULL,
  `nome_latino` VARCHAR(45) NOT NULL,
  `da_interno` TINYINT NOT NULL,
  `esotica` TINYINT NOT NULL,
  PRIMARY KEY (`codice`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;

CREATE UNIQUE INDEX `nome_comune_UNIQUE` ON `ingrosso-piante`.`specie_di_piante` (`nome_comune` ASC) VISIBLE;

CREATE UNIQUE INDEX `nome_latino_UNIQUE` ON `ingrosso-piante`.`specie_di_piante` (`nome_latino` ASC) VISIBLE;


-- -----------------------------------------------------
-- Table `ingrosso-piante`.`fornitore`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `ingrosso-piante`.`fornitore` ;

CREATE TABLE IF NOT EXISTS `ingrosso-piante`.`fornitore` (
  `codice_fornitore` INT UNSIGNED NOT NULL,
  `nome` VARCHAR(45) NOT NULL,
  `codice_fiscale` CHAR(16) NOT NULL,
  PRIMARY KEY (`codice_fornitore`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
```

```
COLLATE = utf8mb4_0900_ai_ci;


CREATE UNIQUE INDEX `codice_fiscale_UNIQUE` ON `ingrosso-piante`.`fornitore` (`codice_fiscale` ASC) VISIBLE;



-- -----------------------------------------------------
-- Table `ingrosso-piante`.`rifornimento`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `ingrosso-piante`.`rifornimento` ;


CREATE TABLE IF NOT EXISTS `ingrosso-piante`.`rifornimento` (
  `fornitore` INT UNSIGNED NOT NULL,
  `data_richiesta` DATETIME NOT NULL,
  `data_consegna` DATETIME NULL DEFAULT NULL,
  PRIMARY KEY (`fornitore`, `data_richiesta`),
  CONSTRAINT `fk_rifornimento_1`
    FOREIGN KEY (`fornitore`)
    REFERENCES `ingrosso-piante`.`fornitore` (`codice_fornitore`)
    ON DELETE RESTRICT
    ON UPDATE RESTRICT)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;



-- -----------------------------------------------------
-- Table `ingrosso-piante`.`contenente`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `ingrosso-piante`.`contenente` ;


CREATE TABLE IF NOT EXISTS `ingrosso-piante`.`contenente` (
  `fornitore` INT UNSIGNED NOT NULL,
  `data_richiesta_rifornimento` DATETIME NOT NULL,
  `specie_di_piante` VARCHAR(45) NOT NULL,
  `quantità` INT UNSIGNED NOT NULL DEFAULT '1',
```

```
  PRIMARY KEY (`fornitore`, `data_richiesta_rifornimento`, `specie_di_piante`),
  CONSTRAINT `fk_contenente_1`
    FOREIGN KEY (`specie_di_piante`)
    REFERENCES `ingrosso-piante`.`specie_di_piante` (`codice`)
    ON DELETE RESTRICT
    ON UPDATE RESTRICT,
  CONSTRAINT `fk_contenente_2`
    FOREIGN KEY (`fornitore` , `data_richiesta_rifornimento`)
    REFERENCES `ingrosso-piante`.`rifornimento` (`fornitore` , `data_richiesta`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;

CREATE INDEX `fk_contenente_1_idx` ON `ingrosso-piante`.`contenente` (`specie_di_piante` ASC) VISIBLE;



-- -----------------------------------------------------
-- Table `ingrosso-piante`.`ordine`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `ingrosso-piante`.`ordine` ;

CREATE TABLE IF NOT EXISTS `ingrosso-piante`.`ordine` (
  `codice` INT UNSIGNED NOT NULL AUTO_INCREMENT,
  `data_apertura` DATETIME NOT NULL,
  `data_finalizzazione` DATETIME NULL DEFAULT NULL,
  `via` VARCHAR(45) NULL DEFAULT NULL,
  `numero_civico` INT UNSIGNED NULL DEFAULT NULL,
  `città` VARCHAR(45) NULL DEFAULT NULL COMMENT '          ',
  `recapito` VARCHAR(45) NULL DEFAULT NULL,
  `referente` VARCHAR(45) NULL DEFAULT NULL,
  `tipo` ENUM('aperto', 'finalizzato', 'spedito') NOT NULL,
  `richiesta` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`codice`),
  CONSTRAINT `fk_ordine_1`
    FOREIGN KEY (`richiesta`)
    REFERENCES `ingrosso-piante`.`cliente` (`recapito_preferito`))
```

```
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;

CREATE INDEX `fk_ordine_1_idx` ON `ingrosso-piante`.`ordine` (`richiesta` ASC) VISIBLE;



-- -----------------------------------------------------
-- Table `ingrosso-piante`.`pianta`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `ingrosso-piante`.`pianta` ;

CREATE TABLE IF NOT EXISTS `ingrosso-piante`.`pianta` (
  `colore` VARCHAR(45) NOT NULL DEFAULT 'Senza fiori',
  `codice_specie` VARCHAR(45) NOT NULL,
  `numero_giacenze` INT UNSIGNED NOT NULL DEFAULT '0',
  PRIMARY KEY (`colore`, `codice_specie`),
  CONSTRAINT `fk_pianta_specie`
    FOREIGN KEY (`codice_specie`)
    REFERENCES `ingrosso-piante`.`specie_di_piante` (`codice`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;

CREATE INDEX `fk_pianta_specie_idx` ON `ingrosso-piante`.`pianta` (`codice_specie` ASC) VISIBLE;



-- -----------------------------------------------------
-- Table `ingrosso-piante`.`contiene`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `ingrosso-piante`.`contiene` ;

CREATE TABLE IF NOT EXISTS `ingrosso-piante`.`contiene` (
  `ordine` INT UNSIGNED NOT NULL,
  `colore_pianta` VARCHAR(45) NOT NULL,
  `specie` VARCHAR(45) NOT NULL,
```

```
  `quantità` INT UNSIGNED NOT NULL DEFAULT '1',
  PRIMARY KEY (`ordine`, `colore_pianta`, `specie`),
  CONSTRAINT `fk_contiene_ordine`
    FOREIGN KEY (`ordine`)
    REFERENCES `ingrosso-piante`.`ordine` (`codice`)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  CONSTRAINT `fk_contiene_pianta`
    FOREIGN KEY (`colore_pianta` , `specie`)
    REFERENCES `ingrosso-piante`.`pianta` (`colore` , `codice_specie`)
    ON DELETE CASCADE
    ON UPDATE CASCADE)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;


CREATE INDEX `fk_contiene_pianta_idx` ON `ingrosso-piante`.`contiene` (`colore_pianta` ASC, `specie` ASC) VISIBLE;



-- -----------------------------------------------------
-- Table `ingrosso-piante`.`ordine_in_elaborazione`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `ingrosso-piante`.`ordine_in_elaborazione` ;


CREATE TABLE IF NOT EXISTS `ingrosso-piante`.`ordine_in_elaborazione` (
  `codice` INT UNSIGNED NOT NULL AUTO_INCREMENT,
  `data_apertura` DATETIME NOT NULL,
  `data_finalizzazione` DATETIME NOT NULL,
  `via` VARCHAR(45) NOT NULL,
  `numero_civico` INT UNSIGNED NOT NULL,
  `città` VARCHAR(45) NOT NULL,
  `recapito` VARCHAR(45) NOT NULL,
  `referente` VARCHAR(45) NULL DEFAULT NULL,
  `richiesta1` DATETIME NOT NULL,
  PRIMARY KEY (`codice`))
```

```
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;



-- -----------------------------------------------------
-- Table `ingrosso-piante`.`contiene1`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `ingrosso-piante`.`contiene1` ;

CREATE TABLE IF NOT EXISTS `ingrosso-piante`.`contiene1` (
  `ordine_in_elaborazione` INT UNSIGNED NOT NULL,
  `colore_pianta` VARCHAR(45) NOT NULL,
  `specie` VARCHAR(45) NOT NULL,
  `quantità` INT UNSIGNED NOT NULL,
  PRIMARY KEY (`ordine_in_elaborazione`, `colore_pianta`, `specie`),
  CONSTRAINT `fk_contiene1_ordine_elaborazione`
    FOREIGN KEY (`ordine_in_elaborazione`)
    REFERENCES `ingrosso-piante`.`ordine_in_elaborazione` (`codice`),
  CONSTRAINT `fk_contiene1_pianta`
    FOREIGN KEY (`colore_pianta` , `specie`)
    REFERENCES `ingrosso-piante`.`pianta` (`colore` , `codice_specie`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;

CREATE INDEX `fk_contiene1_pianta_idx` ON `ingrosso-piante`.`contiene1` (`colore_pianta` ASC, `specie` ASC) VISIBLE;



-- -----------------------------------------------------
-- Table `ingrosso-piante`.`evasi`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `ingrosso-piante`.`evasi` ;

CREATE TABLE IF NOT EXISTS `ingrosso-piante`.`evasi` (
```

```
  `pacco` INT UNSIGNED NOT NULL,
  `ordine` INT UNSIGNED NOT NULL,
  PRIMARY KEY (`pacco`, `ordine`),
  CONSTRAINT `fk_evasi_ordine`
    FOREIGN KEY (`ordine`)
    REFERENCES `ingrosso-piante`.`ordine` (`codice`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;

CREATE INDEX `fk_evasi_ordine_idx` ON `ingrosso-piante`.`evasi` (`ordine` ASC) VISIBLE;



-- -----------------------------------------------------
-- Table `ingrosso-piante`.`evasi1`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `ingrosso-piante`.`evasi1` ;

CREATE TABLE IF NOT EXISTS `ingrosso-piante`.`evasi1` (
  `pacco` INT UNSIGNED NOT NULL,
  `ordine_in_elaborazione` INT UNSIGNED NOT NULL,
  PRIMARY KEY (`pacco`, `ordine_in_elaborazione`),
  CONSTRAINT `fk_evasi1_ordine_elaborazione`
    FOREIGN KEY (`ordine_in_elaborazione`)
    REFERENCES `ingrosso-piante`.`ordine_in_elaborazione` (`codice`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;

CREATE INDEX `fk_evasi1_ordine_elaborazione_idx` ON `ingrosso-piante`.`evasi1` (`ordine_in_elaborazione` ASC)
VISIBLE;



-- -----------------------------------------------------
-- Table `ingrosso-piante`.`fornisce`
-- -----------------------------------------------------
```

```
DROP TABLE IF EXISTS `ingrosso-piante`.`fornisce` ;

CREATE TABLE IF NOT EXISTS `ingrosso-piante`.`fornisce` (
  `fornitore` INT UNSIGNED NOT NULL,
  `specie_di_piante` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`fornitore`, `specie_di_piante`),
  CONSTRAINT `fk_fornisce_fornitore`
    FOREIGN KEY (`fornitore`)
    REFERENCES `ingrosso-piante`.`fornitore` (`codice_fornitore`),
  CONSTRAINT `fk_fornisce_specie`
    FOREIGN KEY (`specie_di_piante`)
    REFERENCES `ingrosso-piante`.`specie_di_piante` (`codice`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;

CREATE INDEX `fk_fornisce_specie_idx` ON `ingrosso-piante`.`fornisce` (`specie_di_piante` ASC) VISIBLE;



-- -----------------------------------------------------
-- Table `ingrosso-piante`.`indirizzo`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `ingrosso-piante`.`indirizzo` ;

CREATE TABLE IF NOT EXISTS `ingrosso-piante`.`indirizzo` (
  `via` VARCHAR(45) NOT NULL,
  `numero_civico` INT UNSIGNED NOT NULL,
  `città` VARCHAR(45) NOT NULL,
  `fornitore` INT UNSIGNED NOT NULL,
  PRIMARY KEY (`via`, `numero_civico`, `città`),
  CONSTRAINT `fk_indirizzo_fornitore`
    FOREIGN KEY (`fornitore`)
    REFERENCES `ingrosso-piante`.`fornitore` (`codice_fornitore`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;
```

```
CREATE INDEX `fk_indirizzo_1_idx` ON `ingrosso-piante`.`indirizzo` (`fornitore` ASC) VISIBLE;



-- -----------------------------------------------------
-- Table `ingrosso-piante`.`inserita`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `ingrosso-piante`.`inserita` ;

CREATE TABLE IF NOT EXISTS `ingrosso-piante`.`inserita` (
  `id_pacco` INT UNSIGNED NOT NULL,
  `colore_pianta` VARCHAR(45) NOT NULL,
  `specie` VARCHAR(45) NOT NULL,
  `quantità` INT UNSIGNED NOT NULL,
  PRIMARY KEY (`id_pacco`, `colore_pianta`, `specie`),
  CONSTRAINT `fk_inserita_pianta`
    FOREIGN KEY (`colore_pianta` , `specie`)
    REFERENCES `ingrosso-piante`.`pianta` (`colore` , `codice_specie`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;

CREATE INDEX `fk_inserita_pianta_idx` ON `ingrosso-piante`.`inserita` (`colore_pianta` ASC, `specie` ASC) VISIBLE;



-- -----------------------------------------------------
-- Table `ingrosso-piante`.`listino_prezzi`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `ingrosso-piante`.`listino_prezzi` ;

CREATE TABLE IF NOT EXISTS `ingrosso-piante`.`listino_prezzi` (
  `data_inizio_validità` DATETIME NOT NULL,
  `codice_specie` VARCHAR(45) NOT NULL,
  `valore` FLOAT UNSIGNED NOT NULL,
  `data_fine_validità` DATETIME NULL DEFAULT NULL,
```

```
  PRIMARY KEY (`data_inizio_validità`, `codice_specie`),
  CONSTRAINT `fk_listino_prezzi_1`
    FOREIGN KEY (`codice_specie`)
    REFERENCES `ingrosso-piante`.`specie_di_piante` (`codice`)
    ON DELETE RESTRICT
    ON UPDATE RESTRICT)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;

CREATE INDEX `fk_listino_prezzi_1_idx` ON `ingrosso-piante`.`listino_prezzi` (`codice_specie` ASC) VISIBLE;



-- -----------------------------------------------------
-- Table `ingrosso-piante`.`recapito_secondario_cliente`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `ingrosso-piante`.`recapito_secondario_cliente` ;

CREATE TABLE IF NOT EXISTS `ingrosso-piante`.`recapito_secondario_cliente` (
  `recapito` VARCHAR(45) NOT NULL,
  `cliente` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`recapito`),
  CONSTRAINT `fk_recapito_secondario_cliente_cliente`
    FOREIGN KEY (`cliente`)
    REFERENCES `ingrosso-piante`.`cliente` (`recapito_preferito`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;

CREATE INDEX `fk_recapito_secondario_cliente_cliente_idx` ON `ingrosso-piante`.`recapito_secondario_cliente`
(`cliente` ASC) VISIBLE;



-- -----------------------------------------------------
-- Table `ingrosso-piante`.`referente_rivendita`
-- -----------------------------------------------------
```

```
DROP TABLE IF EXISTS `ingrosso-piante`.`referente_rivendita` ;


CREATE TABLE IF NOT EXISTS `ingrosso-piante`.`referente_rivendita` (
  `cliente` VARCHAR(45) NOT NULL,
  `nome` VARCHAR(45) NOT NULL,
  `recapito_primario` VARCHAR(45) NULL DEFAULT NULL,
  PRIMARY KEY (`cliente`),
  CONSTRAINT `fk_referente_rivendita_cliente`
    FOREIGN KEY (`cliente`)
    REFERENCES `ingrosso-piante`.`cliente` (`recapito_preferito`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;



-- -----------------------------------------------------
-- Table `ingrosso-piante`.`recapito_secondario_referente`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `ingrosso-piante`.`recapito_secondario_referente` ;


CREATE TABLE IF NOT EXISTS `ingrosso-piante`.`recapito_secondario_referente` (
  `recapito` VARCHAR(45) NOT NULL,
  `referente_rivendita` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`recapito`),
  CONSTRAINT `fk_recapito_secondario_referente_ref_rivendita`
    FOREIGN KEY (`referente_rivendita`)
    REFERENCES `ingrosso-piante`.`referente_rivendita` (`cliente`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;


CREATE INDEX `fk_recapito_secondario_referente_ref_rivendita_idx` ON `ingrosso-piante`.`recapito_secondario_referente` (`referente_rivendita` ASC) VISIBLE;


USE `ingrosso-piante` ;
```

```
DELIMITER ;

SET SQL_MODE = '';

DROP USER IF EXISTS amministratore;

SET
SQL_MODE='ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZERO_DATE,ER
ROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION';

CREATE USER 'amministratore' IDENTIFIED BY 'amministratore';


GRANT EXECUTE ON procedure `ingrosso-piante`.`crea_utente` TO 'amministratore';

GRANT EXECUTE ON procedure `ingrosso-piante`.`visualizza_ordini` TO 'amministratore';

GRANT EXECUTE ON procedure `ingrosso-piante`.`visualizza_info_ordine` TO 'amministratore';

GRANT EXECUTE ON procedure `ingrosso-piante`.`aggiungi_cliente` TO 'amministratore';

GRANT EXECUTE ON procedure `ingrosso-piante`.`aggiungi_contatto_cliente` TO 'amministratore';

SET SQL_MODE = '';

DROP USER IF EXISTS cliente;

SET
SQL_MODE='ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZERO_DATE,ER
ROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION';

CREATE USER 'cliente' IDENTIFIED BY 'cliente';


GRANT EXECUTE ON procedure `ingrosso-piante`.`aggiungi_contatto_cliente` TO 'cliente';

GRANT EXECUTE ON procedure `ingrosso-piante`.`aggiungi_pianta_ordine` TO 'cliente';

GRANT EXECUTE ON procedure `ingrosso-piante`.`crea_ordine` TO 'cliente';

GRANT EXECUTE ON procedure `ingrosso-piante`.`elimina_ordine_aperto` TO 'cliente';

GRANT EXECUTE ON procedure `ingrosso-piante`.`finalizza_ordine` TO 'cliente';

GRANT EXECUTE ON procedure `ingrosso-piante`.`piante_specie` TO 'cliente';

GRANT EXECUTE ON procedure `ingrosso-piante`.`visualizza_info_ordine` TO 'cliente';

GRANT EXECUTE ON procedure `ingrosso-piante`.`visualizza_ordini_aperti` TO 'cliente';

GRANT EXECUTE ON procedure `ingrosso-piante`.`visualizza_ordini` TO 'cliente';

SET SQL_MODE = '';

DROP USER IF EXISTS gestore_magazzino;

SET
SQL_MODE='ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZERO_DATE,ER
ROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION';

CREATE USER 'gestore_magazzino' IDENTIFIED BY 'gestore_magazzino';
```

```
GRANT EXECUTE ON procedure `ingrosso-piante`.`aggiungi_fornitore` TO 'gestore_magazzino';

GRANT EXECUTE ON procedure `ingrosso-piante`.`aggiungi_indirizzo_fornitore` TO 'gestore_magazzino';

GRANT EXECUTE ON procedure `ingrosso-piante`.`aggiungi_specie_fornita` TO 'gestore_magazzino';

GRANT EXECUTE ON procedure `ingrosso-piante`.`richiesta_rifornimento` TO 'gestore_magazzino';

GRANT EXECUTE ON procedure `ingrosso-piante`.`specie_da_rifornire` TO 'gestore_magazzino';

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,ER
ROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION';

CREATE USER 'manager' IDENTIFIED BY 'manager';


GRANT EXECUTE ON procedure `ingrosso-piante`.`aggiungi_specie` TO 'manager';

GRANT EXECUTE ON procedure `ingrosso-piante`.`piante_specie` TO 'manager';

GRANT EXECUTE ON procedure `ingrosso-piante`.`lista_speci_ordinate` TO 'manager';

GRANT EXECUTE ON procedure `ingrosso-piante`.`listini_passati` TO 'manager';

GRANT EXECUTE ON procedure `ingrosso-piante`.`nuovo_listino` TO 'manager';

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,ER
ROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION';

CREATE USER 'operatore' IDENTIFIED BY 'operatore';


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,ER
ROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION';

CREATE USER 'login' IDENTIFIED BY 'login';


GRANT EXECUTE ON procedure `ingrosso-piante`.`login` TO 'login';


SET SQL_MODE=@OLD_SQL_MODE;

SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS;

SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS;


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

```
-- Data for table `ingrosso-piante`.`utenti`

-- -----------------------------------------------------

START TRANSACTION;

USE `ingrosso-piante`;

INSERT INTO `ingrosso-piante`.`utenti` (`username`, `password`, `ruolo`) VALUES ('dada.cliente', '0c88028bf3aa6a6a143ed846f2be1ea4', 'cliente');

INSERT INTO `ingrosso-piante`.`utenti` (`username`, `password`, `ruolo`) VALUES ('dada.manager', '0c88028bf3aa6a6a143ed846f2be1ea4', 'manager');

INSERT INTO `ingrosso-piante`.`utenti` (`username`, `password`, `ruolo`) VALUES ('dada.operatore', '0c88028bf3aa6a6a143ed846f2be1ea4', 'operatore');

INSERT INTO `ingrosso-piante`.`utenti` (`username`, `password`, `ruolo`) VALUES ('dada.magazzino', '0c88028bf3aa6a6a143ed846f2be1ea4', 'gestore_magazzino');

INSERT INTO `ingrosso-piante`.`utenti` (`username`, `password`, `ruolo`) VALUES ('dada.amministratore', '0c88028bf3aa6a6a143ed846f2be1ea4', 'amministratore');

INSERT INTO `ingrosso-piante`.`utenti` (`username`, `password`, `ruolo`) VALUES ('pippo.riventita', '0c88028bf3aa6a6a143ed846f2be1ea4', 'cliente');

INSERT INTO `ingrosso-piante`.`utenti` (`username`, `password`, `ruolo`) VALUES ('dada.rivendita', '0c88028bf3aa6a6a143ed846f2be1ea4', 'cliente');


COMMIT;




-- -----------------------------------------------------

-- Data for table `ingrosso-piante`.`cliente`

-- -----------------------------------------------------

START TRANSACTION;

USE `ingrosso-piante`;

INSERT INTO `ingrosso-piante`.`cliente` (`recapito_preferito`, `nome`, `codice_fiscale`, `partita_IVA`, `tipo`, `via`, `numero_civico`, `città`, `via_fatturazione`, `civico_fatturazione`, `città_fatturazione`, `username`) VALUES ('3332547894', 'davide', 'dvdslr97fge5748d', NULL, 'privato', 'via roma', 58, 'roma', 'via appia', 5, 'roma', 'dada.cliente');

INSERT INTO `ingrosso-piante`.`cliente` (`recapito_preferito`, `nome`, `codice_fiscale`, `partita_IVA`, `tipo`, `via`, `numero_civico`, `città`, `via_fatturazione`, `civico_fatturazione`, `città_fatturazione`, `username`) VALUES ('6258529595', 'paolo', NULL, 'IT123456789', 'rivendita', 'via nord', 5, 'roma', 'via appia', 5, 'roma', 'dada.rivendita');


COMMIT;




-- -----------------------------------------------------

-- Data for table `ingrosso-piante`.`specie_di_piante`

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

```
START TRANSACTION;

USE `ingrosso-piante`;

INSERT INTO `ingrosso-piante`.`specie_di_piante` (`codice`, `nome_comune`, `nome_latino`, `da_interno`, `esotica`) VALUES ('a1', 'rosa', 'rosae', 0, 0);

INSERT INTO `ingrosso-piante`.`specie_di_piante` (`codice`, `nome_comune`, `nome_latino`, `da_interno`, `esotica`) VALUES ('a2', 'ciclamino', 'ciclaminae', 0, 0);

INSERT INTO `ingrosso-piante`.`specie_di_piante` (`codice`, `nome_comune`, `nome_latino`, `da_interno`, `esotica`) VALUES ('a3', 'girasole', 'girasolae', 0, 0);


COMMIT;




-- -----------------------------------------------------

-- Data for table `ingrosso-piante`.`ordine`

-- -----------------------------------------------------

START TRANSACTION;

USE `ingrosso-piante`;

INSERT INTO `ingrosso-piante`.`ordine` (`codice`, `data_apertura`, `data_finalizzazione`, `via`, `numero_civico`, `città`, `recapito`, `referente`, `tipo`, `richiesta`) VALUES (1, '2021-02-07 09:52:12', '2021-02-07 10:50:12', 'via roma', 4, 'napoli', '3366547896', NULL, 'finalizzato', '3332547894');

INSERT INTO `ingrosso-piante`.`ordine` (`codice`, `data_apertura`, `data_finalizzazione`, `via`, `numero_civico`, `città`, `recapito`, `referente`, `tipo`, `richiesta`) VALUES (2, '2021-02-07 10:49:14', '2021-02-07 10:58:54', 'via appia', 45, 'roma', '3365214568', NULL, 'finalizzato', '3332547894');


COMMIT;




-- -----------------------------------------------------

-- Data for table `ingrosso-piante`.`pianta`

-- -----------------------------------------------------

START TRANSACTION;

USE `ingrosso-piante`;

INSERT INTO `ingrosso-piante`.`pianta` (`colore`, `codice_specie`, `numero_giacenze`) VALUES ('blu', 'a1', 46);

INSERT INTO `ingrosso-piante`.`pianta` (`colore`, `codice_specie`, `numero_giacenze`) VALUES ('rosso', 'a2', 5);

INSERT INTO `ingrosso-piante`.`pianta` (`colore`, `codice_specie`, `numero_giacenze`) VALUES ('bianco', 'a2', 45);

INSERT INTO `ingrosso-piante`.`pianta` (`colore`, `codice_specie`, `numero_giacenze`) VALUES ('rossa', 'a1', 5);

INSERT INTO `ingrosso-piante`.`pianta` (`colore`, `codice_specie`, `numero_giacenze`) VALUES ('bianca', 'a1', 55);

INSERT INTO `ingrosso-piante`.`pianta` (`colore`, `codice_specie`, `numero_giacenze`) VALUES ('giallo', 'a3', 100);
```

```
COMMIT;



-- -----------------------------------------------------
-- Data for table `ingrosso-piante`.`contiene`
-- -----------------------------------------------------
START TRANSACTION;
USE `ingrosso-piante`;
INSERT INTO `ingrosso-piante`.`contiene` (`ordine`, `colore_pianta`, `specie`, `quantità`) VALUES (1, 'blu', 'a1', 4);
INSERT INTO `ingrosso-piante`.`contiene` (`ordine`, `colore_pianta`, `specie`, `quantità`) VALUES (2, 'bianco', 'a2', 8);
INSERT INTO `ingrosso-piante`.`contiene` (`ordine`, `colore_pianta`, `specie`, `quantità`) VALUES (2, 'blu', 'a1', 1);
INSERT INTO `ingrosso-piante`.`contiene` (`ordine`, `colore_pianta`, `specie`, `quantità`) VALUES (2, 'rossa', 'a1', 2);
INSERT INTO `ingrosso-piante`.`contiene` (`ordine`, `colore_pianta`, `specie`, `quantità`) VALUES (1, 'giallo', 'a3', 5);

COMMIT;



-- -----------------------------------------------------
-- Data for table `ingrosso-piante`.`listino_prezzi`
-- -----------------------------------------------------
START TRANSACTION;
USE `ingrosso-piante`;
INSERT INTO `ingrosso-piante`.`listino_prezzi` (`data_inizio_validità`, `codice_specie`, `valore`, `data_fine_validità`) VALUES ('2021-02-06 21:13:21', 'a1', 20.99, NULL);
INSERT INTO `ingrosso-piante`.`listino_prezzi` (`data_inizio_validità`, `codice_specie`, `valore`, `data_fine_validità`) VALUES ('2021-02-07 14:04:25', 'a2', 9.99, NULL);
INSERT INTO `ingrosso-piante`.`listino_prezzi` (`data_inizio_validità`, `codice_specie`, `valore`, `data_fine_validità`) VALUES ('2021-02-06 14:04:25', 'a3', 10.99, NULL);

COMMIT;



-- -----------------------------------------------------
-- Data for table `ingrosso-piante`.`referente_rivendita`
-- -----------------------------------------------------
START TRANSACTION;
USE `ingrosso-piante`;
```

```
INSERT INTO `ingrosso-piante`.`referente_rivendita` (`cliente`, `nome`, `recapito_primario`) VALUES ('6258529595', 'lorenzo', '4868465418');


COMMIT;
```

Codice del Front-End

Progetto_BD/client/administrator.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "defines.h"

void add_customer(MYSQL *conn, char username1[46]);

void create_user(MYSQL *conn)
{
	MYSQL_STMT *prepared_stmt;
	MYSQL_BIND param[3];
	char options[5] = {'1','2','3','4','5'};
	char r;

	// Input for the registration routine
	char username[46];
	char password[46];
	char ruolo[46];

	// Get the required information
	printf("\nUsername: ");
	getInput(46, username, false);
	printf("password: ");
	getInput(46, password, true);
	printf("Assign a possible role:\n");
```

```
        printf("\t1) Amministratore\n");
        printf("\t2) Cliente\n");
        printf("\t3) Gestore del magazzino\n");
        printf("\t4) Manager\n");
        printf("\t5) Operatore pacchi\n");
        r = multiChoice("Select role", options, 5);

        // Convert role into enum value
        switch(r) {
                case '1':
                        strcpy(ruolo, "amministratore");
                        break;
                case '2':
                        strcpy(ruolo, "cliente");
                        break;
                case '3':
                        strcpy(ruolo, "gestore_magazzino");
                        break;
    case '4':
                        strcpy(ruolo, "manager");
                        break;
    case '5':
                        strcpy(ruolo, "operatore");
                        break;
                default:
                        fprintf(stderr, "Invalid condition at %s:%d\n", __FILE__, __LINE__);
                        abort();
        }

        // Prepare stored procedure call
        if(!setup_prepared_stmt(&prepared_stmt, "call crea_utente(?, ?, ?)", conn)) {
                finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize user insertion statement\n", false);
                goto out;
        }

        // Prepare parameters
```

```
        memset(param, 0, sizeof(param));

        param[0].buffer_type = MYSQL_TYPE_VAR_STRING;
        param[0].buffer = username;
        param[0].buffer_length = strlen(username);

        param[1].buffer_type = MYSQL_TYPE_VAR_STRING;
        param[1].buffer = password;
        param[1].buffer_length = strlen(password);

        param[2].buffer_type = MYSQL_TYPE_VAR_STRING;
        param[2].buffer = ruolo;
        param[2].buffer_length = strlen(ruolo);

        if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
                finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters for user insertion\n", true);
        }

        // Run procedure
        if (mysql_stmt_execute(prepared_stmt) != 0) {
                print_stmt_error (prepared_stmt, "An error occurred while adding the user.");
                goto out;
        } else {
                printf("User correctly added...\n");
        }

        if (strcmp(ruolo, "cliente") == 0){
    add_customer(conn, username);
  }

out: mysql_stmt_close(prepared_stmt);
}

void info_order(MYSQL *conn){
  MYSQL_STMT *prepared_stmt;
        MYSQL_BIND param[1];
```

```
   int ordine;

   puts("Inserisci codice ordine");
   scanf("%d", &ordine);
   getchar();

   if(!setup_prepared_stmt(&prepared_stmt, "call visualizza_info_ordine(?)", conn)) {
                finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize show info order\n", false);
        }

        memset(param, 0, sizeof(param));

        param[0].buffer_type = MYSQL_TYPE_LONG;
        param[0].buffer = &ordine;
        param[0].buffer_length = sizeof(ordine);

        if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
                finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters for show info order\n", true);
        }

        if (mysql_stmt_execute(prepared_stmt) != 0) {
                finish_with_stmt_error(conn, prepared_stmt, "Could not retrieve show info order\n", true);
        }

        dump_result_set(conn, prepared_stmt, "\nOrder info:");
        if(mysql_stmt_next_result(prepared_stmt)>0){    //chiamata che mi libera consuma il result set perchè in caso di
DATETIME non mi si consuma del tutto
      finish_with_stmt_error(conn, prepared_stmt, "Error", true);
   }
        mysql_stmt_close(prepared_stmt);
}

void add_customer(MYSQL *conn, char username1[46]){
   MYSQL_STMT *prepared_stmt;
   MYSQL_BIND param[14];
```

```
char username[46];
char recapito_preferito[46];
char nome[46];
char tipo[10];
char via[46], citta[46], via_fatturazione[46], citta_fatturazione[46];
int civico, civico_fatturazione;
char codice_fiscale[17] = {'-','\0'};
char partita_IVA[12] = {'-','\0'};
char referente[46] = {'-','\0'};
char recapito_referente[46] = {'-','\0'};
char options1[2] = {'1','2'};
char r1;
char r2;

puts("Inserisci dati cliente");
if (strcmp(username1, " ")==0){
   printf("\nUsername: ");
   getInput(46, username, false);
}
else {
   strcpy(username, username1);
}
puts("Iserisci il nome");
getInput(46, nome, false);
puts("Inserisci la via di residenza");
getInput(46, via, false);
puts("Inserisci il numero civico");
scanf("%d", &civico);
getchar();
puts("Inserisci città di residenza");
getInput(46, citta, false);
puts("Inserisci il recapito principale");
getInput(46, recapito_preferito, false);
puts("Inserisci la via di fatturazione");
getInput(46, via_fatturazione, false);
```

```
puts("Iserisci il civico di fatturazione");
scanf("%d", &civico_fatturazione);
getchar();
puts("Inserisci la città di fatturazione");
getInput(46, citta_fatturazione, false);

printf("Assign a possible type:\n");
printf("\t1) Privato\n");
printf("\t2) Rivendita\n");
r1 = multiChoice("Select type", options1, 2);

switch(r1) {
case '1':
   strcpy(tipo, "privato");
   puts("Inserisci il tuo codice fiscale");
   getInput(17, codice_fiscale, false);
   break;
case '2':
   strcpy(tipo, "rivendita");
   puts("Inserisci partita IVA");
   getInput(12, partita_IVA, false);
   puts("Inserisci il nome del referente di rivendita");
   getInput(46, referente, false);
   puts("Vuoi inserire un recapito del referente della rivendita?");
   printf("\t1) Si\n");
   printf("\t2) No\n");
   r2 = multiChoice("Select 1 o 2", options1, 2);
   if (r2 == 1){
      puts("Inserisci il recapito primario del referente");
      getInput(46, recapito_referente, false);
   }
   break;
default:
   fprintf(stderr, "Invalid condition at %s:%d\n", __FILE__, __LINE__);
   abort();
}
```

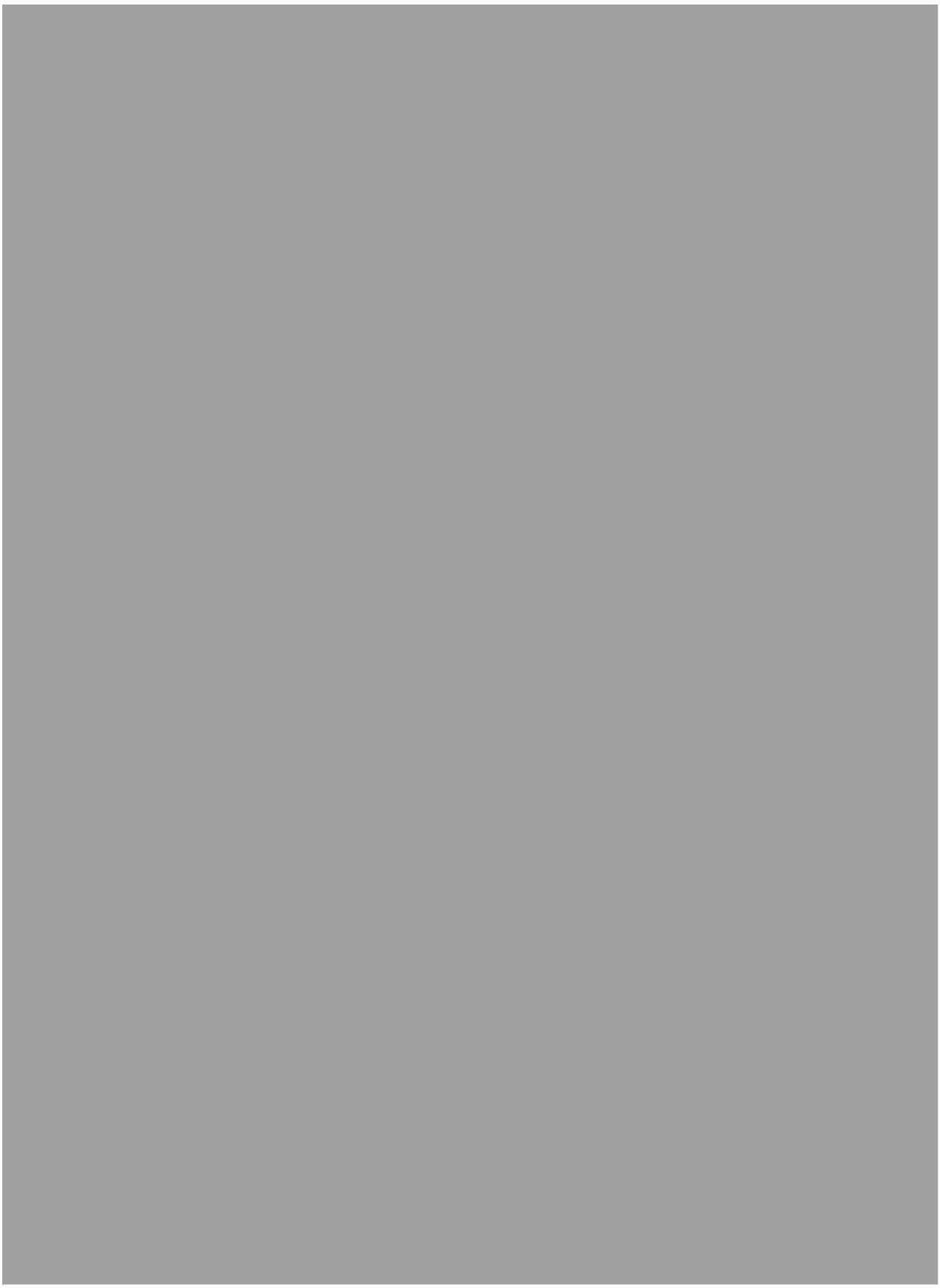
```
if(!setup_prepared_stmt(&prepared_stmt, "call aggiungi_cliente(?,?,?,?,?,?,?,?,?,?,?,?,?,?)", conn)) {
    finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize add customer statement\n", false);
}

// Prepare parameters
memset(param, 0, sizeof(param));

param[0].buffer_type = MYSQL_TYPE_VAR_STRING;
param[0].buffer = username;
param[0].buffer_length = strlen(username);

param[1].buffer_type = MYSQL_TYPE_VAR_STRING;
param[1].buffer = tipo;
param[1].buffer_length = strlen(tipo);

param[2].buffer_type = MYSQL_TYPE_VAR_STRING;
param[2].buffer = recapito_preferito;
param[2].buffer_length = strlen(recapito_preferito);

param[3].buffer_type = MYSQL_TYPE_VAR_STRING;
param[3].buffer = nome;
param[3].buffer_length = strlen(nome);

param[4].buffer_type = MYSQL_TYPE_VAR_STRING;
param[4].buffer = codice_fiscale;
param[4].buffer_length = strlen(codice_fiscale);

param[5].buffer_type = MYSQL_TYPE_VAR_STRING;
param[5].buffer = partita_IVA;
param[5].buffer_length = strlen(partita_IVA);

param[6].buffer_type = MYSQL_TYPE_VAR_STRING;
param[6].buffer = via;
param[6].buffer_length = strlen(via);
```

```
param[7].buffer_type = MYSQL_TYPE_LONG;
param[7].buffer = &civico;
param[7].buffer_length = sizeof(civico);

param[8].buffer_type = MYSQL_TYPE_VAR_STRING;
param[8].buffer = citta;
param[8].buffer_length = strlen(citta);

param[9].buffer_type = MYSQL_TYPE_VAR_STRING;
param[9].buffer = via_fatturazione;
param[9].buffer_length = strlen(via_fatturazione);

param[10].buffer_type = MYSQL_TYPE_LONG;
param[10].buffer = &civico_fatturazione;
param[10].buffer_length = sizeof(civico_fatturazione);

param[11].buffer_type = MYSQL_TYPE_VAR_STRING;
param[11].buffer = citta_fatturazione;
param[11].buffer_length = strlen(citta_fatturazione);

param[12].buffer_type = MYSQL_TYPE_VAR_STRING;
param[12].buffer = referente;
param[12].buffer_length = strlen(referente);

param[13].buffer_type = MYSQL_TYPE_VAR_STRING;
param[13].buffer = recapito_referente;
param[13].buffer_length = strlen(recapito_referente);

if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
    finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters for add customer\n", true);
}

// Run procedure
if (mysql_stmt_execute(prepared_stmt) != 0) {
    print_stmt_error (prepared_stmt, "An error occurred while adding the customer.");
} else {
```

```
        printf("Customer correctly added...\n");
    }
    mysql_stmt_close(prepared_stmt);
}



void customer_orders(MYSQL *conn){
    MYSQL_STMT *prepared_stmt;
            MYSQL_BIND param[1];

    char username[46];
    puts("Inserisci lo username dell'utente del quale si vogliono controllare gli ordini");
    getInput(46, username, false);

    if(!setup_prepared_stmt(&prepared_stmt, "call visualizza_ordini(?)", conn)) {
                    finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize show orders\n", false);
            }

            memset(param, 0, sizeof(param));

            param[0].buffer_type = MYSQL_TYPE_VAR_STRING;
            param[0].buffer = username;
            param[0].buffer_length = strlen(username);

            if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
                    finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters for show orders\n", true);
            }

            if (mysql_stmt_execute(prepared_stmt) != 0) {
                    finish_with_stmt_error(conn, prepared_stmt, "Could not retrieve show orders\n", true);
            }

            dump_result_set(conn, prepared_stmt, "\nList of orders");
            if(mysql_stmt_next_result(prepared_stmt)>0){    //chiamata che mi libera consuma il result set perchè in caso di
DATETIME non mi si consuma del tutto
        finish_with_stmt_error(conn, prepared_stmt, "Error", true);
```

```
    }
        mysql_stmt_close(prepared_stmt);
}

void administrator(MYSQL *conn)
{
        char options[5] = {'1','2', '3', '4', '5'};
        char op;

        printf("Switching to administrative role...\n");

        if(!parse_config("users/amministratore.json", &conf)) {
                fprintf(stderr, "Unable to load administrator configuration\n");
                exit(EXIT_FAILURE);
        }

        if(mysql_change_user(conn, conf.db_username, conf.db_password, conf.database)) {
                fprintf(stderr, "mysql_change_user() failed\n");
                exit(EXIT_FAILURE);
        }

        while(true) {
                printf("\033[2J\033[H");
                printf("*** What should I do for you? ***\n\n");
                printf("1) Create new user\n");
                printf("2) Show ordes of a user\n");
                printf("3) Show info order\n");
                printf("4) Record customer's data\n");
                printf("5) Quit\n");

                op = multiChoice("Select an option", options, 5);

                switch(op) {
                        case '1':
                                create_user(conn);
                                break;
```

```
                        case '2':
                                customer_orders(conn);
                                break;
                        case '3':
                                info_order(conn);
                                break;
                        case '4':
                                add_customer(conn, " ");
                                break;
                        case '5':
                                return;

                        default:
                                fprintf(stderr, "Invalid condition at %s:%d\n", __FILE__, __LINE__);
                                abort();
                }

                getchar();
        }
}
```



Progetto_BD/client/customer.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "defines.h"

void add_plants(MYSQL *conn);
void show_my_orders(MYSQL *conn);
int show_open_orders(MYSQL *conn);

char specie[46];

void add_customer_contact(MYSQL *conn)
{
	MYSQL_STMT *prepared_stmt;
	MYSQL_BIND param[2];

	char recapito[46];

	puts("Inserisci il recapito da aggiungere");
	getInput(46, recapito, false);

	if(!setup_prepared_stmt(&prepared_stmt, "call aggiungi_contatto_cliente(?, ?)", conn)) {
		finish_with_stmt_error(conn, prepared_stmt, "Error setup statement add_customer_contact\n", false);
	}

	// Prepare parameters
	memset(param, 0, sizeof(param));

	param[0].buffer_type = MYSQL_TYPE_VAR_STRING;
	param[0].buffer = conf.username;
	param[0].buffer_length = strlen(conf.username);

	param[1].buffer_type = MYSQL_TYPE_VAR_STRING;
	param[1].buffer = recapito;
	param[1].buffer_length = strlen(recapito);
```

```
	if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
		finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters for adding contact\n", true);
	}

	// Run procedure
	if (mysql_stmt_execute(prepared_stmt) != 0) {
		print_stmt_error (prepared_stmt, "An error occurred while adding contact.");
	} else {
		printf("Contact correctly added...\n");
	}

	mysql_stmt_close(prepared_stmt);
}

void plant_list(MYSQL *conn){
  MYSQL_STMT *prepared_stmt;
	MYSQL_BIND param[2];

  int status;
  float costo = 0;

  puts("A quale specie sei interessato?");
  getInput(46, specie, false);

  if(!setup_prepared_stmt(&prepared_stmt, "call piante_specie(?,?)", conn)) {
		finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize plants statement\n", false);
	}

	// Prepare parameters
	memset(param, 0, sizeof(param));

	param[0].buffer_type = MYSQL_TYPE_VAR_STRING;
	param[0].buffer = specie;
	param[0].buffer_length = strlen(specie);
```

```
param[1].buffer_type = MYSQL_TYPE_FLOAT;
        param[1].buffer = &costo;
        param[1].buffer_length = sizeof(costo);



        if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
                finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters for plant list\n", true);
        }

        if (mysql_stmt_execute(prepared_stmt) != 0) {
                print_stmt_error(prepared_stmt, "Could not retrieve plant list\n");
                goto out;
        }


dump_result_set(conn, prepared_stmt, "\nList of plants of the selected species");

status = mysql_stmt_next_result(prepared_stmt);
if (status >0){
   finish_with_stmt_error(conn, prepared_stmt, "Error: passaggio secondo result non riuscito", true);
}
memset(param, 0, sizeof(param));

param[0].buffer_type = MYSQL_TYPE_FLOAT;
param[0].buffer = &costo;
param[0].buffer_length = sizeof(costo);

if(mysql_stmt_bind_result(prepared_stmt, param)) {
   finish_with_stmt_error(conn, prepared_stmt, "Could not retrieve output parameter", true);
}



if(mysql_stmt_fetch(prepared_stmt)) {
   finish_with_stmt_error(conn, prepared_stmt, "Could not buffer results", true);
}
```

```
    printf("Il costo è di: %f €\n", costo);

    out:
        mysql_stmt_close(prepared_stmt);
}

void create_order(MYSQL *conn)
{
        MYSQL_STMT *prepared_stmt;
        MYSQL_BIND param[5];

        char colore[46];
        int quantita;
        int codice = 0;

        plant_list(conn);

        printf("Quale colore vuoi:\n");
        getInput(46, colore, false);
        printf("Inserire la quantità\n");
        scanf("%d", &quantita);
    getchar();

        if(!setup_prepared_stmt(&prepared_stmt, "call crea_ordine(?, ?, ?, ?, ?)", conn)) {
                finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize create order statement\n", false);
        }

        // Prepare parameters
        memset(param, 0, sizeof(param));

        param[0].buffer_type = MYSQL_TYPE_VAR_STRING;
        param[0].buffer = conf.username;
        param[0].buffer_length = strlen(conf.username);

        param[1].buffer_type = MYSQL_TYPE_VAR_STRING;
        param[1].buffer = colore;
```

```
param[1].buffer_length = strlen(colore);

param[2].buffer_type = MYSQL_TYPE_VAR_STRING;
param[2].buffer = specie;
param[2].buffer_length = strlen(specie);

param[3].buffer_type = MYSQL_TYPE_LONG;
param[3].buffer = &quantita;
param[3].buffer_length = sizeof(quantita);

param[4].buffer_type = MYSQL_TYPE_LONG;
param[4].buffer = &codice;
param[4].buffer_length = sizeof(codice);

if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
        finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters for order creation\n", true);
}

if (mysql_stmt_execute(prepared_stmt) != 0) {
        print_stmt_error(prepared_stmt, "An error occurred while creating order.");
        goto out;
}

memset(param, 0, sizeof(param));
param[0].buffer_type = MYSQL_TYPE_LONG;
param[0].buffer = &codice;
param[0].buffer_length = sizeof(codice);

if(mysql_stmt_bind_result(prepared_stmt, param)) {
        finish_with_stmt_error(conn, prepared_stmt, "Could not retrieve output parameter", true);
}

if(mysql_stmt_fetch(prepared_stmt)) {
        finish_with_stmt_error(conn, prepared_stmt, "Could not buffer results", true);
}
```

```
        printf("Order correctly added with ID %d...\n", codice);

    out:
        mysql_stmt_close(prepared_stmt);
}

void add_plants(MYSQL *conn)
{
        MYSQL_STMT *prepared_stmt;
        MYSQL_BIND param[4];

        char colore[46];
        int quantita;
        int codice;

    if(show_open_orders(conn) == 1){
            puts("Non hai ordini aperti");
            return;
    }
    puts("Inserici codice ordine a cui inserire piante");
    scanf("%d", &codice);
    getchar();

        plant_list(conn);

        printf("Quale colore vuoi:\n");
        getInput(46, colore, false);
        printf("Inserire la quantità\n");
        scanf("%d", &quantita);
    getchar();


        if(!setup_prepared_stmt(&prepared_stmt, "call aggiungi_pianta_ordine(?, ?, ?, ?)", conn)) {
                finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize add plants statement\n", false);
        }
```

```
	memset(param, 0, sizeof(param));


	param[0].buffer_type = MYSQL_TYPE_LONG;
	param[0].buffer = &codice;
	param[0].buffer_length = sizeof(codice);

	param[1].buffer_type = MYSQL_TYPE_VAR_STRING;
	param[1].buffer = specie;
	param[1].buffer_length = strlen(specie);

	param[2].buffer_type = MYSQL_TYPE_VAR_STRING;
	param[2].buffer = colore;
	param[2].buffer_length = strlen(colore);

	param[3].buffer_type = MYSQL_TYPE_LONG;
	param[3].buffer = &quantita;
	param[3].buffer_length = sizeof(quantita);



	if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
		finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters for add plants\n", true);
	}

	if (mysql_stmt_execute(prepared_stmt) != 0) {
		print_stmt_error(prepared_stmt, "Error: probabilmente hai inserito una quantità di piante superiore alle piante che sono disponibili.\nOrdinane un numero minore, i rifornimenti arriveranno presto.");
	}

	mysql_stmt_close(prepared_stmt);
}

int show_open_orders(MYSQL *conn){
    MYSQL_STMT *prepared_stmt;
	 MYSQL_BIND param[1];

    int risultati;
```

```
    if(!setup_prepared_stmt(&prepared_stmt, "call visualizza_ordini_aperti(?)", conn)) {
                    finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize show open orders\n", false);
            }

            memset(param, 0, sizeof(param));

            param[0].buffer_type = MYSQL_TYPE_VAR_STRING;
            param[0].buffer = conf.username;
            param[0].buffer_length = strlen(conf.username);

            if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
                    finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters for open orders\n", true);
            }

            if (mysql_stmt_execute(prepared_stmt) != 0) {
                    finish_with_stmt_error(conn, prepared_stmt, "Could not retrieve open orders\n", true);
            }
   risultati = dump_result_set(conn, prepared_stmt, "\nList of open orders");

            if (risultati == 0){
      if(mysql_stmt_next_result(prepared_stmt)>0){    //chiamata che mi libera consuma il result set perchè in caso di DATETIME non mi si consuma del tutto
          finish_with_stmt_error(conn, prepared_stmt, "Error", true);
      }
   }
            mysql_stmt_close(prepared_stmt);
            return risultati;
}

void show_my_orders(MYSQL *conn){
   MYSQL_STMT *prepared_stmt;
            MYSQL_BIND param[1];

   if(!setup_prepared_stmt(&prepared_stmt, "call visualizza_ordini(?)", conn)) {
                    finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize show orders\n", false);
```

```
        }

        memset(param, 0, sizeof(param));

        param[0].buffer_type = MYSQL_TYPE_VAR_STRING;
        param[0].buffer = conf.username;
        param[0].buffer_length = strlen(conf.username);

        if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
                finish_with_stmt_error(conf, prepared_stmt, "Could not bind parameters for show orders\n", true);
        }

        if (mysql_stmt_execute(prepared_stmt) != 0) {
                finish_with_stmt_error(conf, prepared_stmt, "Could not retrieve show orders\n", true);
        }

        dump_result_set(conf, prepared_stmt, "\nList of orders");
        if(mysql_stmt_next_result(prepared_stmt)>0){        //chiamata che mi libera consuma il result set perchè in caso di DATETIME non mi si consuma del tutto
      finish_with_stmt_error(conf, prepared_stmt, "Error", true);
   }
        mysql_stmt_close(prepared_stmt);
}

void my_info_order(MYSQL *conn){
   MYSQL_STMT *prepared_stmt;
        MYSQL_BIND param[1];

   int ordine;

   show_my_orders(conn);
   puts("Inserisci codice ordine");
   scanf("%d", &ordine);
   getchar();

   if(!setup_prepared_stmt(&prepared_stmt, "call visualizza_info_ordine(?)", conn)) {
```

```
                finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize show info order\n", false);
        }

        memset(param, 0, sizeof(param));

        param[0].buffer_type = MYSQL_TYPE_LONG;
        param[0].buffer = &ordine;
        param[0].buffer_length = sizeof(ordine);

        if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
                finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters for show info order\n", true);
        }

        if (mysql_stmt_execute(prepared_stmt) != 0) {
                finish_with_stmt_error(conn, prepared_stmt, "Could not retrieve show info order\n", true);
        }

        dump_result_set(conn, prepared_stmt, "\nOrder info:");
        if(mysql_stmt_next_result(prepared_stmt)>0){    //chiamata che mi libera consuma il result set perchè in caso di DATETIME non mi si consuma del tutto
      finish_with_stmt_error(conn, prepared_stmt, "Error", true);
   }
        mysql_stmt_close(prepared_stmt);
}

void delete_open_order(MYSQL *conn)
{
        MYSQL_STMT *prepared_stmt;
        MYSQL_BIND param[2];

        int ordine;

   if(show_open_orders(conn) == 1){
             puts("Non hai ordini aperti");
             return;
   }
```

```
	puts("Quale ordine vuoi eliminare?");
	scanf("%d", &ordine);
	getchar();

	if(!setup_prepared_stmt(&prepared_stmt, "call elimina_ordine_aperto(?, ?)", conn)) {
		finish_with_stmt_error(conn, prepared_stmt, "Error setup statement delete order\n", false);
	}

	memset(param, 0, sizeof(param));

	param[0].buffer_type = MYSQL_TYPE_LONG;
	param[0].buffer = &ordine;
	param[0].buffer_length = sizeof(ordine);

	param[1].buffer_type = MYSQL_TYPE_VAR_STRING;
	param[1].buffer = conf.username;
	param[1].buffer_length = strlen(conf.username);

	if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
		finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters for delete order\n", true);
	}

	// Run procedure
	if (mysql_stmt_execute(prepared_stmt) != 0) {
		print_stmt_error (prepared_stmt, "An error occurred while deleting order.");
	} else {
		printf("Order correctly delete...\n");
	}

	mysql_stmt_close(prepared_stmt);
}

void confirm_order(MYSQL *conn)
{
	MYSQL_STMT *prepared_stmt;
	MYSQL_BIND param[7];
```

```
        char via[46];
        char citta[46];
        char recapito[46];
        char referente[46] = {'-','\0'};
        int civico;
        int ordine;
        char options[2] = {'1','2'};
char r;

if(show_open_orders(conn) == 1){
          puts("Non hai ordini aperti");
          return;
}
puts("A ordine vuoi finalizzare?");
puts("Inserici codice");
scanf("%d", &ordine);
getchar();

        puts("Inserisci la via di spedizione");
        getInput(46, via, false);
        printf("Inserire il civico di spedizione\n");
        scanf("%d", &civico);
getchar();
puts("Inserire la città di spedizione");
getInput(46, citta, false);
puts("Inserire il recapito associato all'ordine");
getInput(46, recapito, false);
puts("Si vuole inserire un referente?");
printf("\t1) Si\n");
printf("\t2) No\n");
r = multiChoice("Select 1 o 2", options, 2);
if (r == 1){
   puts("Inserisci il referente dell'ordine");
   getInput(46, referente, false);
}
```

```
        if(!setup_prepared_stmt(&prepared_stmt, "call finalizza_ordine(?, ?, ?, ?, ?, ?, ?)", conn)) {
                finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize confirm order statement\n", false);
        }

        memset(param, 0, sizeof(param));

        param[0].buffer_type = MYSQL_TYPE_LONG;
        param[0].buffer = &ordine;
        param[0].buffer_length = sizeof(ordine);

        param[1].buffer_type = MYSQL_TYPE_VAR_STRING;
        param[1].buffer = via;
        param[1].buffer_length = strlen(via);

        param[2].buffer_type = MYSQL_TYPE_LONG;
        param[2].buffer = &civico;
        param[2].buffer_length = sizeof(civico);

        param[3].buffer_type = MYSQL_TYPE_VAR_STRING;
        param[3].buffer = citta;
        param[3].buffer_length = strlen(citta);

param[4].buffer_type = MYSQL_TYPE_VAR_STRING;
        param[4].buffer = recapito;
        param[4].buffer_length = strlen(recapito);

        param[5].buffer_type = MYSQL_TYPE_VAR_STRING;
        param[5].buffer = referente;
        param[5].buffer_length = strlen(referente);

        param[6].buffer_type = MYSQL_TYPE_VAR_STRING;
        param[6].buffer = conf.username;
        param[6].buffer_length = strlen(conf.username);

        if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
```

```
                finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters for confirm order\n", true);
        }

        if (mysql_stmt_execute(prepared_stmt) != 0) {
                print_stmt_error(prepared_stmt, "An error occurred while confirming an order.");
        }

        mysql_stmt_close(prepared_stmt);
}

void customer(MYSQL *conn)
{
        char options[9] = {'1','2','3','4','5','6','7','8','9'};
        char op;

        printf("Switching to customer role...\n");

        if(!parse_config("users/cliente.json", &conf)) {
                fprintf(stderr, "Unable to load customer configuration\n");
                exit(EXIT_FAILURE);
        }

        if(mysql_change_user(conn, conf.db_username, conf.db_password, conf.database)) {
                fprintf(stderr, "mysql_change_user() failed\n");
                exit(EXIT_FAILURE);
        }

        while(true) {
                printf("\033[2J\033[H");
                printf("*** What should I do for you? ***\n\n");
                printf("1) Add another contact\n");
                printf("2) Create order\n");
                printf("3) Add plants\n");
                printf("4) Delete an open order\n");
                printf("5) Confirm order\n");
                printf("6) Show my orders\n");
```

```
            printf("7) Show info of a order\n");
            printf("8) Plant species list\n");
            printf("9) Quit\n");

            op = multiChoice("Select an option", options, 9);

            switch(op) {
                    case '1':
                            add_customer_contact(conn);
                            break;
                    case '2':
                            create_order(conn);
                            break;
case '3':
                            add_plants(conn);
                            break;
case '4':
                            delete_open_order(conn);
                            break;
case '5':
                            confirm_order(conn);
                            break;
case '6':
                            show_my_orders(conn);
                            break;
case '7':
                            my_info_order(conn);
                            break;
case '8':
                            plant_list(conn);
                            break;
                    case '9':
                            return;

                    default:
                            fprintf(stderr, "Invalid condition at %s:%d\n", __FILE__, __LINE__);
```

```
                                abort();
                }

                getchar();
        }
}
```

Progetto_BD/client/gestore_magazzino.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "defines.h"



void aggiungi_fornitore(MYSQL *conn)
{
```

```
        MYSQL_STMT *prepared_stmt;
        MYSQL_BIND param[7];

        int codice_fornitore;
        char nome[46];
        char codice_fiscale[17];
        char via[46], citta[46];
int civico;
char specie_fornita[46];

        puts("Inserisci il codice del fornitore da aggiungere");
scanf("%d", &codice_fornitore);
getchar();
puts("Inserisci il nome del fornitore da aggiungere");
getInput(46, nome, false);
puts("Inserisci il codice fiscale del fornitore da aggiungere");
getInput(17, codice_fiscale, false);
puts("Inserisci la via della sede principale del fornitore da aggiungere");
getInput(46, via, false);
puts("Inserisci il numero civico della sede principale del fornitore da aggiungere");
scanf("%d", &civico);
getchar();
puts("Inserisci la città della sede principale del fornitore da aggiungere");
getInput(46, citta, false);
puts("Inserisci una specie fornita dal fornitore da aggiungere");
getInput(46, specie_fornita, false);

        if(!setup_prepared_stmt(&prepared_stmt, "call aggiungi_fornitore(?,?,?,?,?,?,?)", conn)) {
                finish_with_stmt_error(conn, prepared_stmt, "Error setup statement aggiungi_fornitore\n", false);
        }

        memset(param, 0, sizeof(param));

        param[0].buffer_type = MYSQL_TYPE_LONG;
        param[0].buffer = &codice_fornitore;
        param[0].buffer_length = sizeof(codice_fornitore);
```

```
        param[1].buffer_type = MYSQL_TYPE_VAR_STRING;
    param[1].buffer = nome;
    param[1].buffer_length = strlen(nome);

    param[2].buffer_type = MYSQL_TYPE_VAR_STRING;
    param[2].buffer = codice_fiscale;
    param[2].buffer_length = strlen(codice_fiscale);

    param[3].buffer_type = MYSQL_TYPE_VAR_STRING;
    param[3].buffer = via;
    param[3].buffer_length = strlen(via);

    param[4].buffer_type = MYSQL_TYPE_LONG;
    param[4].buffer = &civico;
    param[4].buffer_length = sizeof(civico);

    param[5].buffer_type = MYSQL_TYPE_VAR_STRING;
    param[5].buffer = citta;
    param[5].buffer_length = strlen(citta);

        param[6].buffer_type = MYSQL_TYPE_VAR_STRING;
    param[6].buffer = specie_fornita;
    param[6].buffer_length = strlen(specie_fornita);




        if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
                finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters for aggiungi fornitore\n",
true);
        }

        if (mysql_stmt_execute(prepared_stmt) != 0) {
                print_stmt_error (prepared_stmt, "An error occurred while adding fornitore.");
        } else {
                printf("Fornitore correctly added...\n");
```

```
        }

        mysql_stmt_close(prepared_stmt);
}


void aggiungi_indirizzo_fornitore(MYSQL *conn)
{
        MYSQL_STMT *prepared_stmt;
        MYSQL_BIND param[4];

  int codice_fornitore;
        char via[46], citta[46];
  int civico;

  puts("Inserisci il codice del fornitore");
        scanf("%d", &codice_fornitore);
        getchar();
        puts("Inserisci la via da aggiungere");
        getInput(46, via, false);
        puts("Inserisci il numero civico da aggiungere");
        scanf("%d", &civico);
        getchar();
        puts("Inserisci la città da aggiungere");
        getInput(46, citta, false);


        if(!setup_prepared_stmt(&prepared_stmt, "call aggiungi_indirizzo_fornitore(?, ?, ?, ?)", conn)) {
                finish_with_stmt_error(conn, prepared_stmt, "Error setup statement aggiungi indirizzo\n", false);
        }

        memset(param, 0, sizeof(param));

  param[0].buffer_type = MYSQL_TYPE_VAR_STRING;
  param[0].buffer = via;
  param[0].buffer_length = strlen(via);
```

```
    param[1].buffer_type = MYSQL_TYPE_LONG;
    param[1].buffer = &civico;
    param[1].buffer_length = sizeof(civico);

    param[2].buffer_type = MYSQL_TYPE_VAR_STRING;
    param[2].buffer = citta;
    param[2].buffer_length = strlen(citta);

    param[3].buffer_type = MYSQL_TYPE_LONG;
    param[3].buffer = &codice_fornitore;
    param[3].buffer_length = sizeof(codice_fornitore);



        if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
                finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters for adding address\n", true);
        }

        if (mysql_stmt_execute(prepared_stmt) != 0) {
                print_stmt_error (prepared_stmt, "An error occurred while adding address.");
        } else {
                printf("Address correctly added...\n");
        }

        mysql_stmt_close(prepared_stmt);
}

void aggiungi_specie(MYSQL *conn)
{
        MYSQL_STMT *prepared_stmt;
        MYSQL_BIND param[2];

        char codice_fornitore[46];
    char codice_specie[46];

        puts("Inserisci il codice del fornitore");
```

```
        getInput(46, codice_fornitore, false);
    puts("Inserisci il codice della specie fornita");
        getInput(46, codice_specie, false);

        if(!setup_prepared_stmt(&prepared_stmt, "call aggiungi_specie_fornita(?, ?)", conn)) {
            finish_with_stmt_error(conn, prepared_stmt, "Error setup statement aggiungi specie fornita\n", false);
        }

        memset(param, 0, sizeof(param));

        param[0].buffer_type = MYSQL_TYPE_VAR_STRING;
        param[0].buffer = codice_fornitore;
        param[0].buffer_length = strlen(codice_fornitore);

        param[1].buffer_type = MYSQL_TYPE_VAR_STRING;
        param[1].buffer = codice_specie;
        param[1].buffer_length = strlen(codice_specie);

        if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
            finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters for aggiungi specie fornita\n",
true);
        }

        if (mysql_stmt_execute(prepared_stmt) != 0) {
            print_stmt_error (prepared_stmt, "An error occurred while aggiungi specie fornita.");
        } else {
            printf("Specie correctly added...\n");
        }

        mysql_stmt_close(prepared_stmt);
}

void specie_da_rifornire(MYSQL *conn){
    MYSQL_STMT *prepared_stmt;

    if(!setup_prepared_stmt(&prepared_stmt, "call specie_da_rifornire()", conn)) {
```

```
                finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize specie_da_rifornire\n", false);
        }

        if (mysql_stmt_execute(prepared_stmt) != 0) {
                finish_with_stmt_error(conn, prepared_stmt, "Could not retrieve show species\n", true);
        }

   dump_result_set(conn, prepared_stmt, "\nList of species");
      if(mysql_stmt_next_result(prepared_stmt)>0){   //chiamata che mi consuma il result set
         finish_with_stmt_error(conn, prepared_stmt, "Error", true);
      }

        mysql_stmt_close(prepared_stmt);
}

void richiesta_rifornimento(MYSQL *conn)
{
        MYSQL_STMT *prepared_stmt;
        MYSQL_BIND param[3];

        char speci[256];
        char quantita[256];
        char stringa[46];
        char stringa_quantita[46];
        int codice_fornitore;
        int num, l;

        puts("Inserisci il codice del fornitore a cui si commissiona il rifornimento");
        fflush(stdin);
        scanf("%d", &codice_fornitore);
        getchar();


        puts("Quante speci vuoi richiedere?");
        scanf("%d", &num);
        getchar();
```

```
fflush(stdin);

memset(speci, 0, sizeof(speci));
memset(quantita, 0, sizeof(quantita));
strcpy(speci, "");
strcpy(quantita, "");
        for (int i=0; i<num; i++){
    puts("Inserisci codice specie da richiedere");
    getInput(45, stringa, false);
    l = strlen(stringa);
    stringa[l] = '.';
    stringa[l+1] = '\0';
    sprintf(speci+strlen(speci), "%s", stringa);

    puts("Inserisci quantità da richiedere");
    getInput(45, stringa_quantita, false);
    l = strlen(stringa_quantita);
    stringa_quantita[l] = '.';
    stringa_quantita[l+1] = '\0';
    sprintf(quantita+strlen(quantita), "%s", stringa_quantita);
}
printf("%s\n", speci);
printf("%s\n", quantita);

        if(!setup_prepared_stmt(&prepared_stmt, "call richiesta_rifornimento(?, ?, ?)", conn)) {
                finish_with_stmt_error(conn, prepared_stmt, "Error setup statement richiesta rifornimento\n", false);
        }

        memset(param, 0, sizeof(param));

        param[0].buffer_type = MYSQL_TYPE_LONG;
        param[0].buffer = &codice_fornitore;
        param[0].buffer_length = sizeof(codice_fornitore);

        param[1].buffer_type = MYSQL_TYPE_VAR_STRING;
        param[1].buffer = speci;
```

```
	param[1].buffer_length = strlen(speci);

	param[2].buffer_type = MYSQL_TYPE_VAR_STRING;
	param[2].buffer = quantita;
	param[2].buffer_length = strlen(quantita);



	if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
		finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters for richiesta\n", true);
	}

	if (mysql_stmt_execute(prepared_stmt) != 0) {
		print_stmt_error (prepared_stmt, "An error occurred while richiesta rifornimento.");
	} else {
		printf("Richiesta correctly added...\n");
	}

	mysql_stmt_close(prepared_stmt);
}

void warehouse(MYSQL *conn)
{
	char options[6] = {'1','2','3','4','5','6'};
	char op;

	printf("Switching to warehouse role...\n");

	if(!parse_config("users/gestore_magazzino.json", &conf)) {
		fprintf(stderr, "Unable to load warehouse configuration\n");
		exit(EXIT_FAILURE);
	}

	if(mysql_change_user(conn, conf.db_username, conf.db_password, conf.database)) {
		fprintf(stderr, "mysql_change_user() failed\n");
		exit(EXIT_FAILURE);
	}
```

```
while(true) {
        printf("\033[2J\033[H");
        printf("*** What should I do for you? ***\n\n");
        printf("1) Aggiungi un nuovo fornitore\n");
        printf("2) Aggiungi un ulteriore indirizzo al fornitore\n");
        printf("3) Aggiungi una specie alla lista delle speci fornite da un fornitore\n");
        printf("4) Visualizza speci esaurite\n");
        printf("5) Esegui una richiesta di rifornimento\n");
        printf("6) Quit\n");

        op = multiChoice("Select an option", options, 6);

        switch(op) {
                case '1':
                        aggiungi_fornitore(conn);
                        break;
                case '2':
                        aggiungi_indirizzo_fornitore(conn);
                        break;
case '3':
                        aggiungi_specie(conn);
                        break;
case '4':
                        specie_da_rifornire(conn);
                        break;
case '5':
                        richiesta_rifornimento(conn);
                        break;
                case '6':
                        return;

                default:
                        fprintf(stderr, "Invalid condition at %s:%d\n", __FILE__, __LINE__);
                        abort();
        }
```

```
                getchar();
        }
}
```

Progetto_BD/client/manager.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "defines.h"

extern void plant_list(MYSQL *conn);
extern char* pulisci_stringa (char *stringhe);
```

```
void add_plant_species(MYSQL *conn)
{
        MYSQL_STMT *prepared_stmt;
        MYSQL_BIND param[7];

        char codice_specie[46];
        char nome_comune[46];
        char nome_latino[46];
        int da_interno;
        int esotica;
        float prezzo;
        char array_colori[256];
        char* stringa_pulita;
        char stringa[46];
        int num_colorazioni, l;
        char options[2] = {'1','2'};
  char r;

        puts("Inserisci il codice della specie da aggiungere");
        getInput(46, codice_specie, false);
        puts("Inserisci il nome comune della specie da aggiungere");
        getInput(46, nome_comune, false);
        puts("Inserisci nome latino della specie da aggiungere");
        getInput(46, nome_latino, false);
        puts("E' una specie da interno?");
  printf("\t1) Si\n");
  printf("\t2) No\n");
  r = multiChoice("Select 1 o 2", options, 2);
  if (r == 1){
    da_interno = 1;
  }
  else {
    da_interno = 0;
  }
        puts("E' una specie esotica?");
  printf("\t1) Si\n");
```

```
printf("\t2) No\n");
r = multiChoice("Select 1 o 2", options, 2);
if (r == 1){
   esotica = 1;
}
else {
   esotica = 0;
}
        puts("Inserisci il prezzo della specie da aggiungere");
        scanf("%f", &prezzo);
        getchar();
        puts("In quante colorazioni è disponibile?");
        scanf("%d", &num_colorazioni);
        getchar();
        fflush(stdin);

if (num_colorazioni == 0){
   strcpy(array_colori, "-");
}
memset(array_colori, 0, sizeof(array_colori));
strcpy(array_colori, "");
        for (int i=0; i<num_colorazioni; i++){
   puts("Inserisci colorazione");
   fflush(stdin);
   getInput(45, stringa, false);
   l = strlen(stringa);
   stringa[l] = '.';
   stringa[l+1] = '\0';
   sprintf(array_colori+strlen(array_colori), "%s", stringa);
}
stringa_pulita = pulisci_stringa(array_colori);
        if(!setup_prepared_stmt(&prepared_stmt, "call aggiungi_specie(?, ?, ?, ?, ?, ?, ?)", conn)) {
                finish_with_stmt_error(conn, prepared_stmt, "Error setup statement add_plant_species\n", false);
                free(stringa_pulita);
        }
```

```
    memset(param, 0, sizeof(param));

    param[0].buffer_type = MYSQL_TYPE_VAR_STRING;
    param[0].buffer = codice_specie;
    param[0].buffer_length = strlen(codice_specie);

    param[1].buffer_type = MYSQL_TYPE_VAR_STRING;
    param[1].buffer = nome_comune;
    param[1].buffer_length = strlen(nome_comune);

    param[2].buffer_type = MYSQL_TYPE_VAR_STRING;
    param[2].buffer = nome_latino;
    param[2].buffer_length = strlen(nome_latino);

    param[3].buffer_type = MYSQL_TYPE_TINY;
    param[3].buffer = &da_interno;
    param[3].buffer_length = sizeof(da_interno);

    param[4].buffer_type = MYSQL_TYPE_TINY;
    param[4].buffer = &esotica;
    param[4].buffer_length = sizeof(esotica);

    param[5].buffer_type = MYSQL_TYPE_FLOAT;
    param[5].buffer = &prezzo;
    param[5].buffer_length = sizeof(prezzo);

    param[6].buffer_type = MYSQL_TYPE_VAR_STRING;
    param[6].buffer = stringa_pulita;
    param[6].buffer_length = strlen(stringa_pulita);

    if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
            finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters for adding species\n", true);
free(stringa_pulita);
    }

    if (mysql_stmt_execute(prepared_stmt) != 0) {
```

```
                print_stmt_error (prepared_stmt, "An error occurred while adding species.");
        } else {
                printf("Species correctly added...\n");
        }

        mysql_stmt_close(prepared_stmt);
        free(stringa_pulita);
}

void show_species(MYSQL *conn){
   MYSQL_STMT *prepared_stmt;

   if(!setup_prepared_stmt(&prepared_stmt, "call lista_speci_ordinate()", conn)) {
                finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize show species\n", false);
        }

        if (mysql_stmt_execute(prepared_stmt) != 0) {
                finish_with_stmt_error(conn, prepared_stmt, "Could not retrieve show species\n", true);
        }

        dump_result_set(conn, prepared_stmt, "\nList of species");
   if(mysql_stmt_next_result(prepared_stmt)>0){   //chiamata che mi libera il result set
      finish_with_stmt_error(conn, prepared_stmt, "Error", true);
   }
        mysql_stmt_close(prepared_stmt);
}

void show_past_price_lists(MYSQL *conn){
   MYSQL_STMT *prepared_stmt;
   MYSQL_BIND param[1];

   char specie[46];

   puts("A quale specie sei interessato?");
   getInput(46, specie, false);
```

```
   if(!setup_prepared_stmt(&prepared_stmt, "call listini_passati(?)", conn)) {
      finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize listini passati\n", false);
   }

   memset(param, 0, sizeof(param));

   param[0].buffer_type = MYSQL_TYPE_VAR_STRING;
         param[0].buffer = specie;
         param[0].buffer_length = strlen(specie);

   if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
      finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters for listini passati\n", true);
   }

   if (mysql_stmt_execute(prepared_stmt) != 0) {
      finish_with_stmt_error(conn, prepared_stmt, "Could not retrieve listini passati\n", true);
   }

   dump_result_set(conn, prepared_stmt, "\nList of past price");
   if(mysql_stmt_next_result(prepared_stmt)>0){     //chiamata che mi libera consuma il result set perchè in caso di DATETIME non mi si consuma del tutto
      finish_with_stmt_error(conn, prepared_stmt, "Error", true);
   }
   mysql_stmt_close(prepared_stmt);
}

void edit_price_list(MYSQL *conn)
{
         MYSQL_STMT *prepared_stmt;
         MYSQL_BIND param[2];

         char specie[46];
         float costo;

         puts("Inserisci il codice della specie");
```

```
        getInput(46, specie, false);

   puts("Inserisci il nuovo prezzo");
        scanf("%f", &costo);
        getchar();

        if(!setup_prepared_stmt(&prepared_stmt, "call nuovo_listino(?, ?)", conn)) {
                finish_with_stmt_error(conn, prepared_stmt, "Error setup statement nuovo listino\n", false);
        }

        memset(param, 0, sizeof(param));

        param[0].buffer_type = MYSQL_TYPE_VAR_STRING;
        param[0].buffer = specie;
        param[0].buffer_length = strlen(specie);

        param[1].buffer_type = MYSQL_TYPE_FLOAT;
        param[1].buffer = &costo;
        param[1].buffer_length = sizeof(costo);

        if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
                finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters for adding a new price\n",
true);
        }

        // Run procedure
        if (mysql_stmt_execute(prepared_stmt) != 0) {
                print_stmt_error (prepared_stmt, "An error occurred while editing price.");
        } else {
                printf("Price correctly added...\n");
        }

        mysql_stmt_close(prepared_stmt);
}
```

```
void manager(MYSQL *conn)
{
	char options[6] = {'1','2','3','4','5','6'};
	char op;

	printf("Switching to manager role...\n");

	if(!parse_config("users/manager.json", &conf)) {
		fprintf(stderr, "Unable to load manager configuration\n");
		exit(EXIT_FAILURE);
	}

	if(mysql_change_user(conn, conf.db_username, conf.db_password, conf.database)) {
		fprintf(stderr, "mysql_change_user() failed\n");
		exit(EXIT_FAILURE);
	}

	while(true) {
		printf("\033[2J\033[H");
		printf("*** What should I do for you? ***\n\n");
		printf("1) Add plant species\n");
		printf("2) Show species\n");
		printf("3) Show plants\n");
		printf("4) Show past price lists\n");
		printf("5) Edit a price list\n");
		printf("6) Quit\n");

		op = multiChoice("Select an option", options, 6);

		switch(op) {
			case '1':
				add_plant_species(conn);
				break;
			case '2':
				show_species(conn);
```

```
                                        break;
        case '3':
                                        plant_list(conn);
                                        break;
        case '4':
                                        show_past_price_lists(conn);
                                        break;
        case '5':
                                        edit_price_list(conn);
                                        break;
                                case '6':
                                        return;

                                default:
                                        fprintf(stderr, "Invalid condition at %s:%d\n", __FILE__, __LINE__);
                                        abort();
                        }

                        getchar();
          }
}
```

Progetto_BD/client/main.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <mysql.h>

#include "defines.h"



typedef enum {
	ADMINISTRATOR = 1,
	CUSTOMER,
	WAREHOUSE,
```

```
	MANAGER,
	OPERATOR,
	FAILED_LOGIN
} role_t;


struct configuration conf;

static MYSQL *conn;


static role_t attempt_login(MYSQL *conn, char *username, char *password) {
	MYSQL_STMT *login_procedure;

	MYSQL_BIND param[3]; // Used both for input and output
	int role = 0;

	if(!setup_prepared_stmt(&login_procedure, "call login(?, ?, ?)", conn)) {
		print_stmt_error(login_procedure, "Unable to initialize login statement\n");
		goto err2;
	}

	// Prepare parameters
	memset(param, 0, sizeof(param));

	param[0].buffer_type = MYSQL_TYPE_VAR_STRING; // IN
	param[0].buffer = username;
	param[0].buffer_length = strlen(username);

	param[1].buffer_type = MYSQL_TYPE_VAR_STRING; // IN
	param[1].buffer = password;
	param[1].buffer_length = strlen(password);

	param[2].buffer_type = MYSQL_TYPE_LONG; // OUT
	param[2].buffer = &role;
	param[2].buffer_length = sizeof(role);
```

```
        if (mysql_stmt_bind_param(login_procedure, param) != 0) { // Note _param
                print_stmt_error(login_procedure, "Could not bind parameters for login");
                goto err;
        }

        // Run procedure
        if (mysql_stmt_execute(login_procedure) != 0) {
                print_stmt_error(login_procedure, "Could not execute login procedure");
                goto err;
        }

        // Prepare output parameters
        memset(param, 0, sizeof(param));
        param[0].buffer_type = MYSQL_TYPE_LONG; // OUT
        param[0].buffer = &role;
        param[0].buffer_length = sizeof(role);

        if(mysql_stmt_bind_result(login_procedure, param)) {
                print_stmt_error(login_procedure, "Could not retrieve output parameter");
                goto err;
        }

        // Retrieve output parameter
        if(mysql_stmt_fetch(login_procedure)) {
                print_stmt_error(login_procedure, "Could not buffer results");
                goto err;
        }

        mysql_stmt_close(login_procedure);
        return role;

err:
        mysql_stmt_close(login_procedure);
err2:
        return FAILED_LOGIN;
```

```
}

int main(void) {
	role_t role;

	if(!parse_config("users/login.json", &conf)) {
		fprintf(stderr, "Unable to load login configuration\n");
		exit(EXIT_FAILURE);
	}

	conn = mysql_init (NULL);
	if (conn == NULL) {
		fprintf (stderr, "mysql_init() failed (probably out of memory)\n");
		exit(EXIT_FAILURE);
	}

	if (mysql_real_connect(conn, conf.host, conf.db_username, conf.db_password, conf.database, conf.port, NULL,
CLIENT_MULTI_STATEMENTS | CLIENT_MULTI_RESULTS) == NULL) {
		fprintf (stderr, "mysql_real_connect() failed\n");
		mysql_close (conn);
		exit(EXIT_FAILURE);
	}

	printf("Username: ");
	getInput(46, conf.username, false);
	printf("Password: ");
	getInput(46, conf.password, true);

	role = attempt_login(conn, conf.username, conf.password);

	switch(role) {
		case CUSTOMER:
			customer(conn);
			break;

		case WAREHOUSE:
```

```
                warehouse(conn);
                break;

        case ADMINISTRATOR:
                administrator(conn);
                break;

        case MANAGER:
                manager(conn);
                break;

        /*case OPERATOR:
                operator(conn);
                break;*/

        case FAILED_LOGIN:
                fprintf(stderr, "Invalid credentials\n");
                exit(EXIT_FAILURE);
                break;

        default:
                fprintf(stderr, "Invalid condition at %s:%d\n", __FILE__, __LINE__);
                abort();
    }

    printf("Bye!\n");

    mysql_close (conn);
    return 0;
}
```

Progetto_BD/client/parse.c

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "defines.h"

#define BUFF_SIZE 4096

// The final config struct will point into this
static char config[BUFF_SIZE];
```

```
/**
 * JSON type identifier. Basic types are:
 *      o Object
 *      o Array
 *      o String
 *      o Other primitive: number, boolean (true/false) or null
 */
typedef enum {
        JSMN_UNDEFINED = 0,
        JSMN_OBJECT = 1,
        JSMN_ARRAY = 2,
        JSMN_STRING = 3,
        JSMN_PRIMITIVE = 4
} jsmntype_t;

enum jsmnerr {
        /* Not enough tokens were provided */
        JSMN_ERROR_NOMEM = -1,
        /* Invalid character inside JSON string */
        JSMN_ERROR_INVAL = -2,
        /* The string is not a full JSON packet, more bytes expected */
        JSMN_ERROR_PART = -3
};

/**
 * JSON token description.
 * type         type (object, array, string etc.)
 * start    start position in JSON data string
 * end          end position in JSON data string
 */
typedef struct {
        jsmntype_t type;
        int start;
        int end;
        int size;
#ifdef JSMN_PARENT_LINKS
```

```
	int parent;
#endif
} jsmntok_t;

/**
 * JSON parser. Contains an array of token blocks available. Also stores
 * the string being parsed now and current position in that string
 */
typedef struct {
	unsigned int pos; /* offset in the JSON string */
	unsigned int toknext; /* next token to allocate */
	int toksuper; /* superior token node, e.g parent object or array */
} jsmn_parser;

/**
 * Allocates a fresh unused token from the token pool.
 */
static jsmntok_t *jsmn_alloc_token(jsmn_parser *parser, jsmntok_t *tokens, size_t num_tokens) {
	jsmntok_t *tok;
	if (parser->toknext >= num_tokens) {
		return NULL;
	}
	tok = &tokens[parser->toknext++];
	tok->start = tok->end = -1;
	tok->size = 0;
#ifdef JSMN_PARENT_LINKS
	tok->parent = -1;
#endif
	return tok;
}

/**
 * Fills token type and boundaries.
 */
static void jsmn_fill_token(jsmntok_t *token, jsmntype_t type,
		int start, int end) {
```

```
	token->type = type;
	token->start = start;
	token->end = end;
	token->size = 0;
}

/**
 * Fills next available token with JSON primitive.
 */
static int jsmn_parse_primitive(jsmn_parser *parser, const char *js,
		size_t len, jsmntok_t *tokens, size_t num_tokens) {
	jsmntok_t *token;
	int start;

	start = parser->pos;

	for (; parser->pos < len && js[parser->pos] != '\0'; parser->pos++) {
		switch (js[parser->pos]) {
#ifndef JSMN_STRICT
			/* In strict mode primitive must be followed by "," or "}" or "]" */
			case ':':
#endif
			case '\t' : case '\r' : case '\n' : case ' ' :
			case ','  : case ']'  : case '}' :
				goto found;
		}
		if (js[parser->pos] < 32 || js[parser->pos] >= 127) {
			parser->pos = start;
			return JSMN_ERROR_INVAL;
		}
	}
#ifdef JSMN_STRICT
	/* In strict mode primitive must be followed by a comma/object/array */
	parser->pos = start;
	return JSMN_ERROR_PART;
#endif
```

```
found:
	if (tokens == NULL) {
		parser->pos--;
		return 0;
	}
	token = jsmn_alloc_token(parser, tokens, num_tokens);
	if (token == NULL) {
		parser->pos = start;
		return JSMN_ERROR_NOMEM;
	}
	jsmn_fill_token(token, JSMN_PRIMITIVE, start, parser->pos);
#ifdef JSMN_PARENT_LINKS
	token->parent = parser->toksuper;
#endif
	parser->pos--;
	return 0;
}

/**
 * Fills next token with JSON string.
 */
static int jsmn_parse_string(jsmn_parser *parser, const char *js,
		size_t len, jsmntok_t *tokens, size_t num_tokens) {
	jsmntok_t *token;

	int start = parser->pos;

	parser->pos++;

	/* Skip starting quote */
	for (; parser->pos < len && js[parser->pos] != '\0'; parser->pos++) {
		char c = js[parser->pos];

		/* Quote: end of string */
		if (c == '\"') {
```

```
                        if (tokens == NULL) {
                                return 0;
                        }
                        token = jsmn_alloc_token(parser, tokens, num_tokens);
                        if (token == NULL) {
                                parser->pos = start;
                                return JSMN_ERROR_NOMEM;
                        }
                        jsmn_fill_token(token, JSMN_STRING, start+1, parser->pos);
#ifdef JSMN_PARENT_LINKS
                        token->parent = parser->toksuper;
#endif
                        return 0;
                }

                /* Backslash: Quoted symbol expected */
                if (c == '\\' && parser->pos + 1 < len) {
                        int i;
                        parser->pos++;
                        switch (js[parser->pos]) {
                                /* Allowed escaped symbols */
                                case '\"': case '/' : case '\\' : case 'b' :
                                case 'f' : case 'r' : case 'n'  : case 't' :
                                        break;
                                /* Allows escaped symbol \uXXXX */
                                case 'u':
                                        parser->pos++;
                                        for(i = 0; i < 4 && parser->pos < len && js[parser->pos] != '\0'; i++) {
                                                /* If it isn't a hex character we have an error */
                                                if(!((js[parser->pos] >= 48 && js[parser->pos] <= 57) || /* 0-9 */
                                                                (js[parser->pos] >= 65 && js[parser->pos] <= 70) || /* A-F */
                                                                (js[parser->pos] >= 97 && js[parser->pos] <= 102))) { /* a-f */
                                                        parser->pos = start;
                                                        return JSMN_ERROR_INVAL;
                                                }
```

```
                                        parser->pos++;
                                }
                                parser->pos--;
                                break;
                        /* Unexpected symbol */
                        default:
                                parser->pos = start;
                                return JSMN_ERROR_INVAL;
                        }
                }
        }
        parser->pos = start;
        return JSMN_ERROR_PART;
}

/**
 * Parse JSON string and fill tokens.
 */
static int jsmn_parse(jsmn_parser *parser, const char *js, size_t len, jsmntok_t *tokens, unsigned int num_tokens) {
        int r;
        int i;
        jsmntok_t *token;
        int count = parser->toknext;

        for (; parser->pos < len && js[parser->pos] != '\0'; parser->pos++) {
                char c;
                jsmntype_t type;

                c = js[parser->pos];
                switch (c) {
                        case '{': case '[':
                                count++;
                                if (tokens == NULL) {
                                        break;
                                }
                                token = jsmn_alloc_token(parser, tokens, num_tokens);
```

```
                        if (token == NULL)
                                return JSMN_ERROR_NOMEM;
                        if (parser->toksuper != -1) {
                                tokens[parser->toksuper].size++;
#ifdef JSMN_PARENT_LINKS
                                token->parent = parser->toksuper;
#endif
                        }
                        token->type = (c == '{' ? JSMN_OBJECT : JSMN_ARRAY);
                        token->start = parser->pos;
                        parser->toksuper = parser->toknext - 1;
                        break;
                case '}': case ']':
                        if (tokens == NULL)
                                break;
                        type = (c == '}' ? JSMN_OBJECT : JSMN_ARRAY);
#ifdef JSMN_PARENT_LINKS
                        if (parser->toknext < 1) {
                                return JSMN_ERROR_INVAL;
                        }
                        token = &tokens[parser->toknext - 1];
                        for (;;) {
                                if (token->start != -1 && token->end == -1) {
                                        if (token->type != type) {
                                                return JSMN_ERROR_INVAL;
                                        }
                                        token->end = parser->pos + 1;
                                        parser->toksuper = token->parent;
                                        break;
                                }
                                if (token->parent == -1) {
                                        if(token->type != type || parser->toksuper == -1) {
                                                return JSMN_ERROR_INVAL;
                                        }
                                        break;
                                }
```

```
                                token = &tokens[token->parent];
                        }
#else
                        for (i = parser->toknext - 1; i >= 0; i--) {
                                token = &tokens[i];
                                if (token->start != -1 && token->end == -1) {
                                        if (token->type != type) {
                                                return JSMN_ERROR_INVAL;
                                        }
                                        parser->toksuper = -1;
                                        token->end = parser->pos + 1;
                                        break;
                                }
                        }
                        /* Error if unmatched closing bracket */
                        if (i == -1) return JSMN_ERROR_INVAL;
                        for (; i >= 0; i--) {
                                token = &tokens[i];
                                if (token->start != -1 && token->end == -1) {
                                        parser->toksuper = i;
                                        break;
                                }
                        }
#endif
                        break;
                case '\"':
                        r = jsmn_parse_string(parser, js, len, tokens, num_tokens);
                        if (r < 0) return r;
                        count++;
                        if (parser->toksuper != -1 && tokens != NULL)
                                tokens[parser->toksuper].size++;
                        break;
                case '\t' : case '\r' : case '\n' : case ' ':
                        break;
                case ':':
                        parser->toksuper = parser->toknext - 1;
```

```
                        break;
                case ',':
                        if (tokens != NULL && parser->toksuper != -1 &&
                                        tokens[parser->toksuper].type != JSMN_ARRAY &&
                                        tokens[parser->toksuper].type != JSMN_OBJECT) {
#ifdef JSMN_PARENT_LINKS
                                parser->toksuper = tokens[parser->toksuper].parent;
#else
                                for (i = parser->toknext - 1; i >= 0; i--) {
                                        if (tokens[i].type == JSMN_ARRAY || tokens[i].type ==
JSMN_OBJECT) {
                                                if (tokens[i].start != -1 && tokens[i].end == -1) {
                                                        parser->toksuper = i;
                                                        break;
                                                }
                                        }
                                }
#endif
                        }
                        break;
#ifdef JSMN_STRICT
                /* In strict mode primitives are: numbers and booleans */
                case '-': case '0': case '1' : case '2': case '3' : case '4':
                case '5': case '6': case '7' : case '8': case '9':
                case 't': case 'f': case 'n' :
                        /* And they must not be keys of the object */
                        if (tokens != NULL && parser->toksuper != -1) {
                                jsmntok_t *t = &tokens[parser->toksuper];
                                if (t->type == JSMN_OBJECT ||
                                                (t->type == JSMN_STRING && t->size != 0)) {
                                        return JSMN_ERROR_INVAL;
                                }
                        }
#else
                /* In non-strict mode every unquoted value is a primitive */
                default:
```

```
#endif
                                        r = jsmn_parse_primitive(parser, js, len, tokens, num_tokens);
                                        if (r < 0) return r;
                                        count++;
                                        if (parser->toksuper != -1 && tokens != NULL)
                                                tokens[parser->toksuper].size++;
                                        break;

#ifdef JSMN_STRICT
                                /* Unexpected char in strict mode */
                                default:
                                        return JSMN_ERROR_INVAL;
#endif
                        }
                }

                if (tokens != NULL) {
                        for (i = parser->toknext - 1; i >= 0; i--) {
                                /* Unmatched opened object or array */
                                if (tokens[i].start != -1 && tokens[i].end == -1) {
                                        return JSMN_ERROR_PART;
                                }
                        }
                }

                return count;
}

/**
 * Creates a new parser based over a given  buffer with an array of tokens
 * available.
 */
static void jsmn_init(jsmn_parser *parser) {
                parser->pos = 0;
                parser->toknext = 0;
                parser->toksuper = -1;
```

```
}

static int jsoneq(const char *json, jsmntok_t *tok, const char *s)
{
	if (tok->type == JSMN_STRING
	    && (int) strlen(s) == tok->end - tok->start
	    && strncmp(json + tok->start, s, tok->end - tok->start) == 0) {
		return 0;
	}
	return -1;
}

static size_t load_file(char *filename)
{
	FILE *f = fopen(filename, "rb");
	if(f == NULL) {
		fprintf(stderr, "Unable to open file %s\n", filename);
		exit(1);
	}

	fseek(f, 0, SEEK_END);
	size_t fsize = ftell(f);
	fseek(f, 0, SEEK_SET);  //same as rewind(f);

	if(fsize >= BUFF_SIZE) {
		fprintf(stderr, "Configuration file too large\n");
		abort();
	}

	fread(config, fsize, 1, f);
	fclose(f);

	config[fsize] = 0;
	return fsize;
}
```

```
int parse_config(char *path, struct configuration *conf)
{
	int i;
	int r;
	jsmn_parser p;
	jsmntok_t t[128]; /* We expect no more than 128 tokens */

	load_file(path);

	jsmn_init(&p);
	r = jsmn_parse(&p, config, strlen(config), t, sizeof(t)/sizeof(t[0]));
	if (r < 0) {
		printf("Failed to parse JSON: %d\n", r);
		return 0;
	}

	/* Assume the top-level element is an object */
	if (r < 1 || t[0].type != JSMN_OBJECT) {
		printf("Object expected\n");
		return 0;
	}

	/* Loop over all keys of the root object */
	for (i = 1; i < r; i++) {
		if (jsoneq(config, &t[i], "host") == 0) {
			/* We may use strndup() to fetch string value */
			conf->host = strndup(config + t[i+1].start, t[i+1].end-t[i+1].start);
			i++;
		} else if (jsoneq(config, &t[i], "username") == 0) {
			conf->db_username = strndup(config + t[i+1].start, t[i+1].end-t[i+1].start);
			i++;
		} else if (jsoneq(config, &t[i], "password") == 0) {
			conf->db_password = strndup(config + t[i+1].start, t[i+1].end-t[i+1].start);
			i++;
		} else if (jsoneq(config, &t[i], "port") == 0) {
			conf->port = strtol(config + t[i+1].start, NULL, 10);
```

```
                        i++;
                } else if (jsoneq(config, &t[i], "database") == 0) {
                        conf->database = strndup(config + t[i+1].start, t[i+1].end-t[i+1].start);
                        i++;
                } else {
                        printf("Unexpected key: %.*s\n", t[i].end-t[i].start, config + t[i].start);
                }
        }
        return 1;
}
```

Progetto_BD/client/inout.c

```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
```

```
#include <string.h>
#include <ctype.h>
#include <termios.h>
#include <sys/ioctl.h>
#include <pthread.h>
#include <signal.h>
#include <stdbool.h>

#include "defines.h"

// Per la gestione dei segnali
static volatile sig_atomic_t signo;
typedef struct sigaction sigaction_t;
static void handler(int s);

char *getInput(unsigned int lung, char *stringa, bool hide)
{
	char c;
	unsigned int i;

	// Dichiara le variabili necessarie ad un possibile mascheramento dell'input
	sigaction_t sa, savealrm, saveint, savehup, savequit, saveterm;
	sigaction_t savetstp, savettin, savettou;
	struct termios term, oterm;

	if(hide) {
		// Svuota il buffer
		(void) fflush(stdout);

		// Cattura i segnali che altrimenti potrebbero far terminare il programma, lasciando l'utente senza output sulla shell
		sigemptyset(&sa.sa_mask);
		sa.sa_flags = SA_INTERRUPT; // Per non resettare le system call
		sa.sa_handler = handler;
		(void) sigaction(SIGALRM, &sa, &savealrm);
		(void) sigaction(SIGINT, &sa, &saveint);
```

```
		(void) sigaction(SIGHUP, &sa, &savehup);
		(void) sigaction(SIGQUIT, &sa, &savequit);
		(void) sigaction(SIGTERM, &sa, &saveterm);
		(void) sigaction(SIGTSTP, &sa, &savetstp);
		(void) sigaction(SIGTTIN, &sa, &savettin);
		(void) sigaction(SIGTTOU, &sa, &savettou);

		// Disattiva l'output su schermo
		if (tcgetattr(fileno(stdin), &oterm) == 0) {
			(void) memcpy(&term, &oterm, sizeof(struct termios));
			term.c_lflag &= ~(ECHO|ECHONL);
			(void) tcsetattr(fileno(stdin), TCSAFLUSH, &term);
		} else {
			(void) memset(&term, 0, sizeof(struct termios));
			(void) memset(&oterm, 0, sizeof(struct termios));
		}
	}

	// Acquisisce da tastiera al più lung - 1 caratteri
	for(i = 0; i < lung; i++) {
		(void) fread(&c, sizeof(char), 1, stdin);
		if(c == '\n') {
			stringa[i] = '\0';
			break;
		} else
			stringa[i] = c;

		// Gestisce gli asterischi
		if(hide) {
			if(c == '\b') // Backspace
				(void) write(fileno(stdout), &c, sizeof(char));
			else
				(void) write(fileno(stdout), "*", sizeof(char));
		}
	}
```

```
	// Controlla che il terminatore di stringa sia stato inserito
	if(i == lung - 1)
		stringa[i] = '\0';

	// Se sono stati digitati più caratteri, svuota il buffer della tastiera
	if(strlen(stringa) >= lung) {
		// Svuota il buffer della tastiera
		do {
			c = getchar();
		} while (c != '\n');
	}

	if(hide) {
		//L'a capo dopo l'input
		(void) write(fileno(stdout), "\n", 1);

		// Ripristina le impostazioni precedenti dello schermo
		(void) tcsetattr(fileno(stdin), TCSAFLUSH, &oterm);

		// Ripristina la gestione dei segnali
		(void) sigaction(SIGALRM, &savealrm, NULL);
		(void) sigaction(SIGINT, &saveint, NULL);
		(void) sigaction(SIGHUP, &savehup, NULL);
		(void) sigaction(SIGQUIT, &savequit, NULL);
		(void) sigaction(SIGTERM, &saveterm, NULL);
		(void) sigaction(SIGTSTP, &savetstp, NULL);
		(void) sigaction(SIGTTIN, &savettin, NULL);
		(void) sigaction(SIGTTOU, &savettou, NULL);

		// Se era stato ricevuto un segnale viene rilanciato al processo stesso
		if(signo)
			(void) raise(signo);
	}

	return stringa;
}
```

```
// Per la gestione dei segnali
static void handler(int s) {
	signo = s;
}


bool yesOrNo(char *domanda, char yes, char no, bool predef, bool insensitive)
{

	// I caratteri 'yes' e 'no' devono essere minuscoli
	yes = tolower(yes);
	no = tolower(no);

	// Decide quale delle due lettere mostrare come predefinite
	char s, n;
	if(predef) {
		s = toupper(yes);
		n = no;
	} else {
		s = yes;
		n = toupper(no);
	}

	// Richiesta della risposta
	while(true) {
		// Mostra la domanda
		printf("%s [%c/%c]: ", domanda, s, n);

		char c;
		getInput(1, &c, false);

		// Controlla quale risposta è stata data
		if(c == '\0') { // getInput() non può restituire '\n'!
			return predef;
		} else if(c == yes) {
```

```
                return true;
        } else if(c == no) {
                return false;
        } else if(c == toupper(yes)) {
                if(predef || insensitive) return true;
        } else if(c == toupper(yes)) {
                if(!predef || insensitive) return false;
        }
    }
}

char multiChoice(char *domanda, char choices[], int num)
{

    // Genera la stringa delle possibilità
    char *possib = malloc(2 * num * sizeof(char));
    int i, j = 0;
    for(i = 0; i < num; i++) {
        possib[j++] = choices[i];
        possib[j++] = '/';
    }
    possib[j-1] = '\0'; // Per eliminare l'ultima '/'

    // Chiede la risposta
    while(true) {
        // Mostra la domanda
        printf("%s [%s]: ", domanda, possib);

        char c;
        getInput(1, &c, false);

        // Controlla se è un carattere valido
        for(i = 0; i < num; i++) {
            if(c == choices[i])
                return c;
        }
```

```
        }
}
```

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "defines.h"

char* pulisci_stringa (char *stringhe){
    char* nuova_stringa;
    nuova_stringa = (char *) malloc(256);
    int i=0;
    while (*stringhe != '\0') {
        if (*stringhe != ' ') {
```

```
            nuova_stringa[i] = *stringhe;
            i++;
        }
        stringhe++;
    }
    nuova_stringa[i+1]='\0';
    return nuova_stringa;
}



void print_stmt_error (MYSQL_STMT *stmt, char *message)
{
        fprintf (stderr, "%s\n", message);
        if (stmt != NULL) {
                fprintf (stderr, "Error %u (%s): %s\n",
                        mysql_stmt_errno (stmt),
                        mysql_stmt_sqlstate(stmt),
                        mysql_stmt_error (stmt));
        }
}



void print_error(MYSQL *conn, char *message)
{
        fprintf (stderr, "%s\n", message);
        if (conn != NULL) {
                #if MYSQL_VERSION_ID >= 40101
                fprintf (stderr, "Error %u (%s): %s\n",
                mysql_errno (conn), mysql_sqlstate(conn), mysql_error (conn));
                #else
                fprintf (stderr, "Error %u: %s\n",
                mysql_errno (conn), mysql_error (conn));
                #endif
        }
}
```

```
bool setup_prepared_stmt(MYSQL_STMT **stmt, char *statement, MYSQL *conn)
{
	bool update_length = true;

	*stmt = mysql_stmt_init(conn);
	if (*stmt == NULL)
	{
		print_error(conn, "Could not initialize statement handler");
		return false;
	}

	if (mysql_stmt_prepare (*stmt, statement, strlen(statement)) != 0) {
		print_stmt_error(*stmt, "Could not prepare statement");
		return false;
	}

	mysql_stmt_attr_set(*stmt, STMT_ATTR_UPDATE_MAX_LENGTH, &update_length);

	return true;
}

void finish_with_error(MYSQL *conn, char *message)
{
	print_error(conn, message);
	mysql_close(conn);
	exit(EXIT_FAILURE);
}

void finish_with_stmt_error(MYSQL *conn, MYSQL_STMT *stmt, char *message, bool close_stmt)
{
	print_stmt_error(stmt, message);
	if(close_stmt)	mysql_stmt_close(stmt);
	mysql_close(conn);
	exit(EXIT_FAILURE);
}
```

```
static void print_dashes(MYSQL_RES *res_set)
{
	MYSQL_FIELD *field;
	unsigned int i, j;

	mysql_field_seek(res_set, 0);
	putchar('+');
	for (i = 0; i < mysql_num_fields(res_set); i++) {
		field = mysql_fetch_field(res_set);
		for (j = 0; j < field->max_length + 2; j++)
			putchar('-');
		putchar('+');
	}
	putchar('\n');
}

static void dump_result_set_header(MYSQL_RES *res_set)
{
	MYSQL_FIELD *field;
	unsigned long col_len;
	unsigned int i;

	/* determine column display widths -- requires result set to be */
	/* generated with mysql_store_result(), not mysql_use_result() */

	mysql_field_seek (res_set, 0);

	for (i = 0; i < mysql_num_fields (res_set); i++) {
		field = mysql_fetch_field (res_set);
		col_len = strlen(field->name);

		if (col_len < field->max_length)
			col_len = field->max_length;
		if (col_len < 4 && !IS_NOT_NULL(field->flags))
			col_len = 4; /* 4 = length of the word "NULL" */
		field->max_length = col_len; /* reset column info */
```

```
	}

	print_dashes(res_set);
	putchar('|');
	mysql_field_seek (res_set, 0);
	for (i = 0; i < mysql_num_fields(res_set); i++) {
		field = mysql_fetch_field(res_set);
		printf(" %-*s |", (int)field->max_length, field->name);
	}
	putchar('\n');

	print_dashes(res_set);
}

int dump_result_set(MYSQL *conn, MYSQL_STMT *stmt, char *title)
{
	int i;
	int status;
	int num_fields;       /* number of columns in result */
	MYSQL_FIELD *fields;  /* for result set metadata */
	MYSQL_BIND *rs_bind;  /* for output buffers */
	MYSQL_RES *rs_metadata;
	MYSQL_TIME *date;
	size_t attr_size;
	int num_rows;

	/* Prefetch the whole result set. This in conjunction with
	 * STMT_ATTR_UPDATE_MAX_LENGTH set in `setup_prepared_stmt`
	 * updates the result set metadata which are fetched in this
	 * function, to allow to compute the actual max length of
	 * the columns.
	 */
	if (mysql_stmt_store_result(stmt)) {
		fprintf(stderr, " mysql_stmt_execute(), 1 failed\n");
		fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
		exit(0);
```

```
        }

        /* the column count is > 0 if there is a result set */
        /* 0 if the result is only the final status packet */
        num_fields = mysql_stmt_field_count(stmt);
        num_rows = mysql_stmt_num_rows(stmt);

        if (num_rows < 1){
    puts("Non ci sono risultati da mostrare");
    if(mysql_stmt_next_result(stmt)>0){   //chiamata che mi consuma il result set
       finish_with_stmt_error(conn, stmt, "Error", false);
    }
    return 1;
}

        if (num_fields > 0) {
                /* there is a result set to fetch */
                printf("%s\n", title);

                if((rs_metadata = mysql_stmt_result_metadata(stmt)) == NULL) {
                        finish_with_stmt_error(conn, stmt, "Unable to retrieve result metadata\n", true);
                }

                dump_result_set_header(rs_metadata);

                fields = mysql_fetch_fields(rs_metadata);

                rs_bind = (MYSQL_BIND *)malloc(sizeof (MYSQL_BIND) * num_fields);
                if (!rs_bind) {
                        finish_with_stmt_error(conn, stmt, "Cannot allocate output buffers\n", true);
                }
                memset(rs_bind, 0, sizeof (MYSQL_BIND) * num_fields);

                /* set up and bind result set output buffers */
                for (i = 0; i < num_fields; ++i) {
```

```
// Properly size the parameter buffer
switch(fields[i].type) {
        case MYSQL_TYPE_DATE:
        case MYSQL_TYPE_TIMESTAMP:
        case MYSQL_TYPE_DATETIME:
        case MYSQL_TYPE_TIME:
                attr_size = sizeof(MYSQL_TIME);
                break;
        case MYSQL_TYPE_FLOAT:
                attr_size = sizeof(float);
                break;
        case MYSQL_TYPE_DOUBLE:
                attr_size = sizeof(double);
                break;
        case MYSQL_TYPE_TINY:
                attr_size = sizeof(signed char);
                break;
        case MYSQL_TYPE_SHORT:
        case MYSQL_TYPE_YEAR:
                attr_size = sizeof(short int);
                break;
        case MYSQL_TYPE_LONG:
        case MYSQL_TYPE_INT24:
                attr_size = sizeof(int);
                break;
        case MYSQL_TYPE_LONGLONG:
                attr_size = sizeof(int);
                break;
        default:
                attr_size = fields[i].max_length;
                break;
}

// Setup the binding for the current parameter
rs_bind[i].buffer_type = fields[i].type;
rs_bind[i].buffer = malloc(attr_size + 1);
```

```
                rs_bind[i].buffer_length = attr_size + 1;

                if(rs_bind[i].buffer == NULL) {
                        finish_with_stmt_error(conn, stmt, "Cannot allocate output buffers\n", true);
                }
        }

        if(mysql_stmt_bind_result(stmt, rs_bind)) {
                finish_with_stmt_error(conn, stmt, "Unable to bind output parameters\n", true);
        }

        /* fetch and display result set rows */
        while (true) {
                status = mysql_stmt_fetch(stmt);

                if (status == 1 || status == MYSQL_NO_DATA)
                        break;

                putchar('|');

                for (i = 0; i < num_fields; i++) {

                        if (rs_bind[i].is_null_value) {
                                printf (" %-*s |", (int)fields[i].max_length, "NULL");
                                continue;
                        }

                        switch (rs_bind[i].buffer_type) {

                                case MYSQL_TYPE_VAR_STRING:

                                        printf(" %-*s |", (int)fields[i].max_length, (char*)rs_bind[i].buffer);
                                        break;

case MYSQL_TYPE_DATETIME:
   date = (MYSQL_TIME *)rs_bind[i].buffer;
```

```
                                        printf(" %d-%02d-%02d Ora:%02d Min:%02d Sec:%02d |",
date->year, date->month, date->day, date->hour, date->minute, date->second);
                                        break;

                                    case MYSQL_TYPE_DATE:
                                    case MYSQL_TYPE_TIMESTAMP:
                                        date = (MYSQL_TIME *)rs_bind[i].buffer;
                                        printf(" %d-%02d-%02d |", date->year, date->month, date->day);
                                        break;

                                    case MYSQL_TYPE_STRING:
                                        printf(" %-*s |", (int)fields[i].max_length, (char
*)rs_bind[i].buffer);
                                        break;

                                    case MYSQL_TYPE_FLOAT:
                                    case MYSQL_TYPE_DOUBLE:
                                        printf(" %.02f |", *(float *)rs_bind[i].buffer);
                                        break;

                                    case MYSQL_TYPE_LONG:
                                    case MYSQL_TYPE_SHORT:
                                    case MYSQL_TYPE_TINY:
                                        printf(" %-*d |", (int)fields[i].max_length, *(int
*)rs_bind[i].buffer);
                                        break;

                                    case MYSQL_TYPE_NEWDECIMAL:
                                        printf(" %-*.02lf |", (int)fields[i].max_length, *(float*)
rs_bind[i].buffer);
                                        break;

                                    default:
                                       printf("ERROR: Unhandled type (%d)\n", rs_bind[i].buffer_type);
                                       abort();
                                }
                            }
```

```
                putchar('\n');
                print_dashes(rs_metadata);
        }

        mysql_free_result(rs_metadata); /* free metadata */

        /* free output buffers */
        for (i = 0; i < num_fields; i++) {
                free(rs_bind[i].buffer);
        }
        free(rs_bind);
    }
    return 0;
}
```

Progetto_BD/client/defines.h

```
#pragma once

#include <stdbool.h>
#include <mysql.h>

struct configuration {
	char *host;
	char *db_username;
	char *db_password;
	unsigned int port;
	char *database;

	char username[128];
	char password[128];
};

extern struct configuration conf;

extern int parse_config(char *path, struct configuration *conf);
extern char *getInput(unsigned int lung, char *stringa, bool hide);
extern bool yesOrNo(char *domanda, char yes, char no, bool predef, bool insensitive);
extern char multiChoice(char *domanda, char choices[], int num);
extern void print_error (MYSQL *conn, char *message);
extern void print_stmt_error (MYSQL_STMT *stmt, char *message);
extern void finish_with_error(MYSQL *conn, char *message);
extern void finish_with_stmt_error(MYSQL *conn, MYSQL_STMT *stmt, char *message, bool close_stmt);
extern bool setup_prepared_stmt(MYSQL_STMT **stmt, char *statement, MYSQL *conn);
extern int dump_result_set(MYSQL *conn, MYSQL_STMT *stmt, char *title);
extern void customer(MYSQL *conn);
extern void administrator(MYSQL *conn);
extern void manager(MYSQL *conn);
extern void warehouse(MYSQL *conn);
```

```
//extern void run_as_operator(MYSQL *conn);
```