Basi di Dati e Conoscenza

Progetto A.A. 2019/2020

SISTEMA GESTIONE PIANTE

0253467

Massimo Mazzetti

Indice

1. Descrizione del Minimondo

Si vuole realizzare un Sistema 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	Esterni	Sinonimo
7	Appartamento	Interni	Sinonimo
7, 31, 43, 46	Piante	Specie di Piante	Sinonimo
12	Indirizzo di persona	Indirizzo di residenza	Sinonimo
13	Indirizzo di rivendita	Indirizzo di residenza	Sinonimo
30	acquisto	ordine	Sinonimo

Specifica disambiguata

Si vuole realizzare un Sistema per la gestione della vendita all'ingrosso 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 INTERNI o da ESTERNI, se sia una specie esotica o meno e se sono 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 di residenza.
Per ogni rivendita sono noti la partita iva, il nome e l'indirizzo di residenza.
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 ordine, 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 inserire una determinata specie in 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.

Glossario dei Termini

Termine	Descrizione	Sinonimi	Collegamenti
Specie di piante	Per specie di piante indichiamo tutti i diversi tipi di piante trattate dall'azienda.	Piante	Listino prezzi, Magazzino, Ordini, Fornitori, Pacchi
Clienti	Possono essere privati o rivendite, sono coloro che acquistano le specie di piante.		Ordine
Fornitori	Aziende da cui si rifornisce il magazzino dell'azienda.		Specie di Piante
Magazzino	Posto dove si ha la giacenza delle piante.		Specie di piante
Ordini	Specie di Piante acquistate da cliente.	Acquisti	Clienti, Specie di Piante, Pacchi
Listino Prezzi	Costo di vendita delle piante.		Specie di piante
Pacchi	Mezzo attraverso cui vengono evasi gli ordini.		Ordini, 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 INTERNI o da ESTERNI, se sia una specie esotica o meno e se sono verdi oppure fiorite. Nel caso di specie di piante fiorite, sono note tutte le colorazioni in cui una specie è disponibile.

Frasi relative a Clienti
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 di residenza. Per ogni rivendita sono noti la partita iva, il nome e l'indirizzo di residenza. 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.

Frasi relative a 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.

Frasi relative a Magazzino
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.

Frasi relative a Ordine
Si vuole tener traccia di tutti gli ordini eseguiti 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. 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 inserire una determinata specie in un ordine se non vi è disponibilità in magazzino.

Frasi relative a Listino Prezzi
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.

Frasi relative a 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.

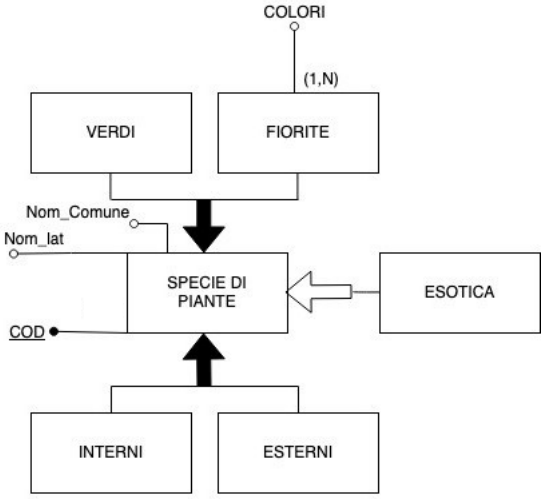
3. Progettazione concettuale

Costruzione dello schema E-R

Ho deciso di approcciare una tecnica mista per la realizzazione del modello E-R.

----------PIANTE----------

L'entità SPECIE DI PIANTA tiene traccia di tutte le specie gestite.
COD è un codice di 6 cifre alfanumerico che rappresenta la chiave primaria per differenziare e ricercare ogni specie.
Ci sono due generalizzazioni totali perché le Specie possono essere verdi o fiorite(in questo caso potrà presentare diversi tipi di colorazione), e allo stesso tempo, da interno o da esterno. C'è anche una generalizzazione parziale in quanto le piante possono essere esotiche.

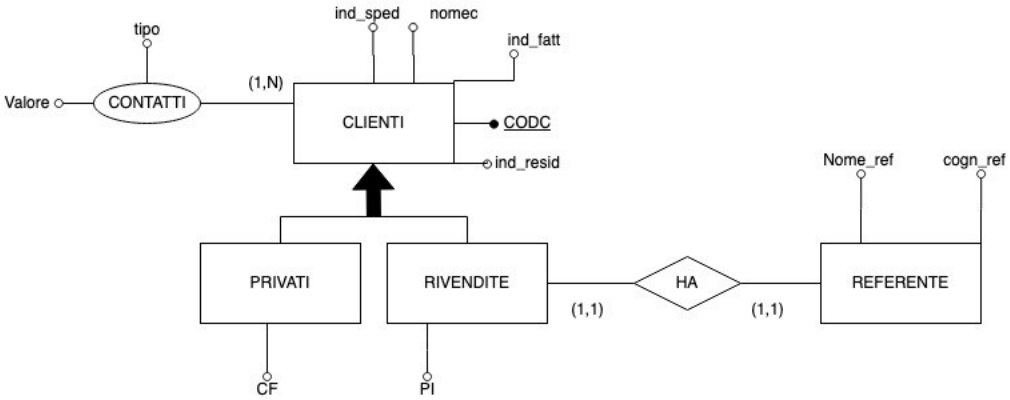


----------CLIENTI----------

Nell'entità CLIENTI si tiene traccia di tutti i clienti che ha l'azienda.

C'è una generalizzazione totale in quanto i clienti possono essere privati oppure rivendite, nel caso delle rivendite c'è una relazione che lo assocerà ad un referente.

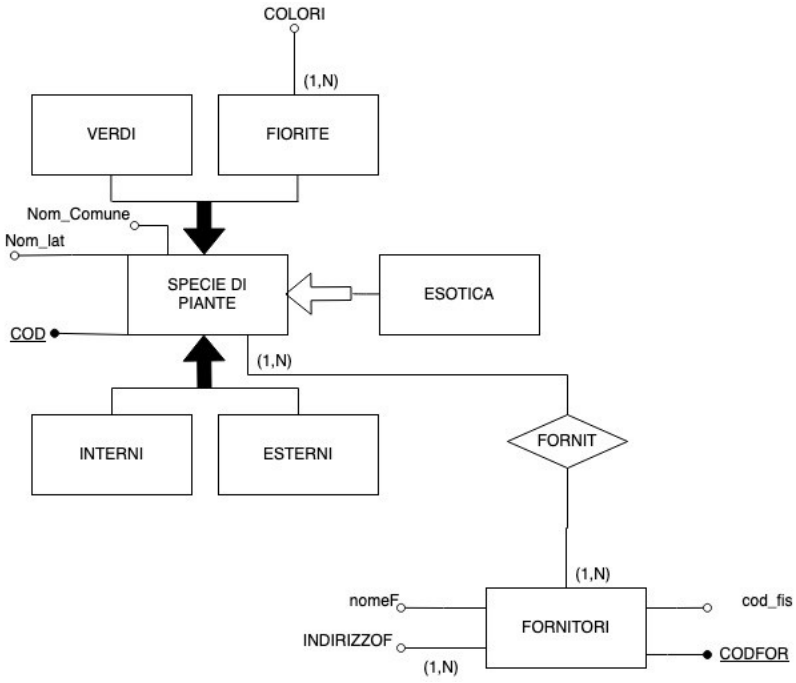
I clienti possono avere più contatti, perciò in un attributo multi valore si terranno i dati tipo(email, cell, telefono) e valore(aaa@gmail.com, ecc…)



----------FORNITORI----------

Nell'entità FORNITORI sono salvate le informazioni di cui si vuole tenere traccia riguardo tutti i fornitori dell'azienda.

I fornitori sono identificati da un codice e possono avere un diverso numero di indirizzi, inoltre un fornitore può fornire diverse specie di piante e una specie può essere fornita da più fornitori.

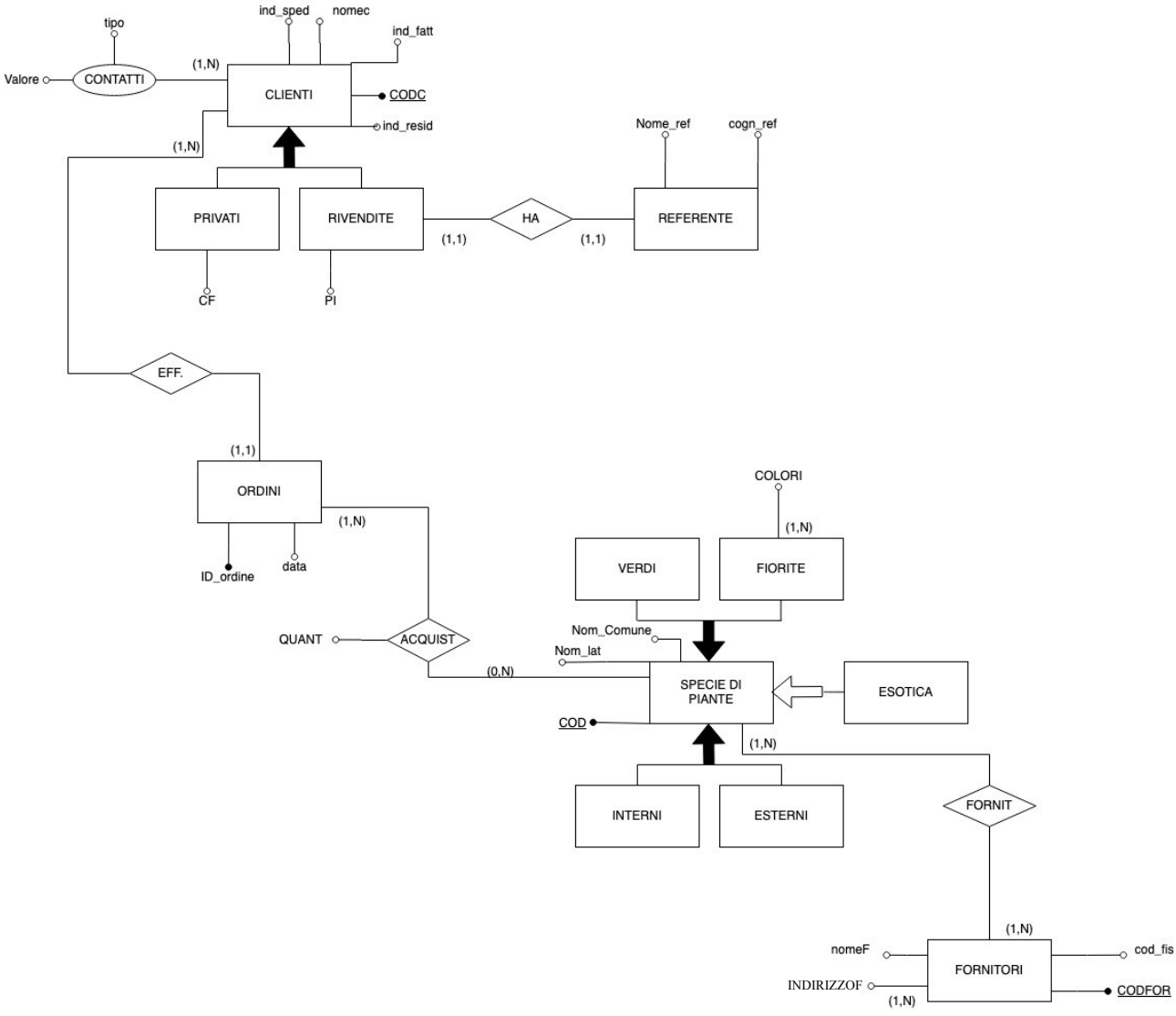


----------ORDINI----------

Per tenere traccia di tutti gli ordini effettuatati da un cliente si inserisce l'entità ORDINI che identificherà ogni ordine con un codice(ID_ordine).

Un cliente potrà eseguire più ordini ma un ordine sarà solo di un cliente.

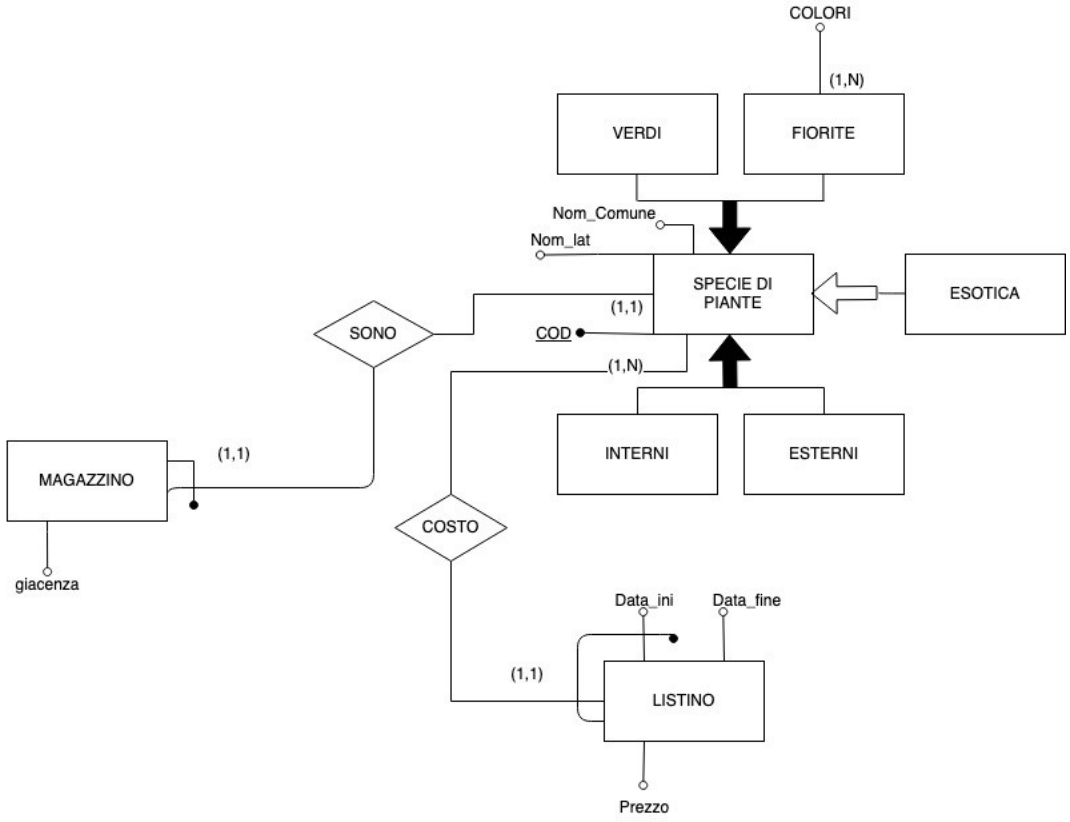
In un ordine ci saranno diverse specie in diverse quantità (Relazione ACQUIST tiene traccia delle quantità di una specie relative a un determinato ordine) ma non per forza una specie disponibile dovrà essere in un ordine.



----------MAGAZZINO E LISTINO----------

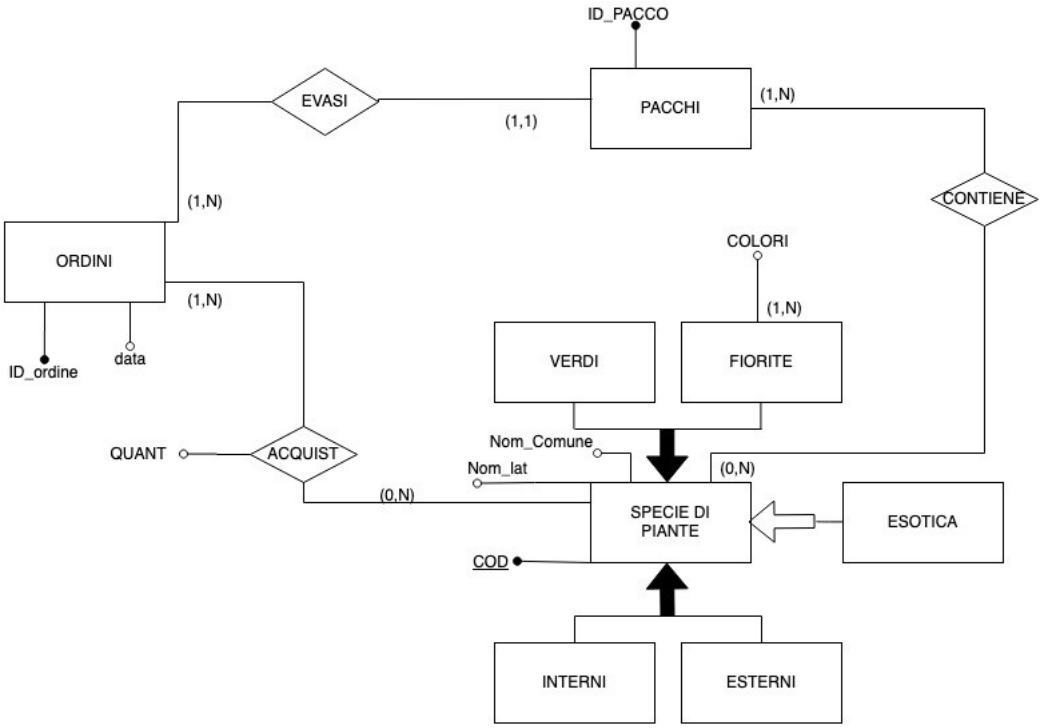
Nell'entità MAGAZZINO si terrà traccia della giacenza di una determinata specie.

Dato che le Specie possono cambiare prezzo e si vuole tenere traccia di tutti i prezzi anche passati nell'entità LISTINO saranno salvati tutti i prezzi di un Specie nel periodo di riferimento.

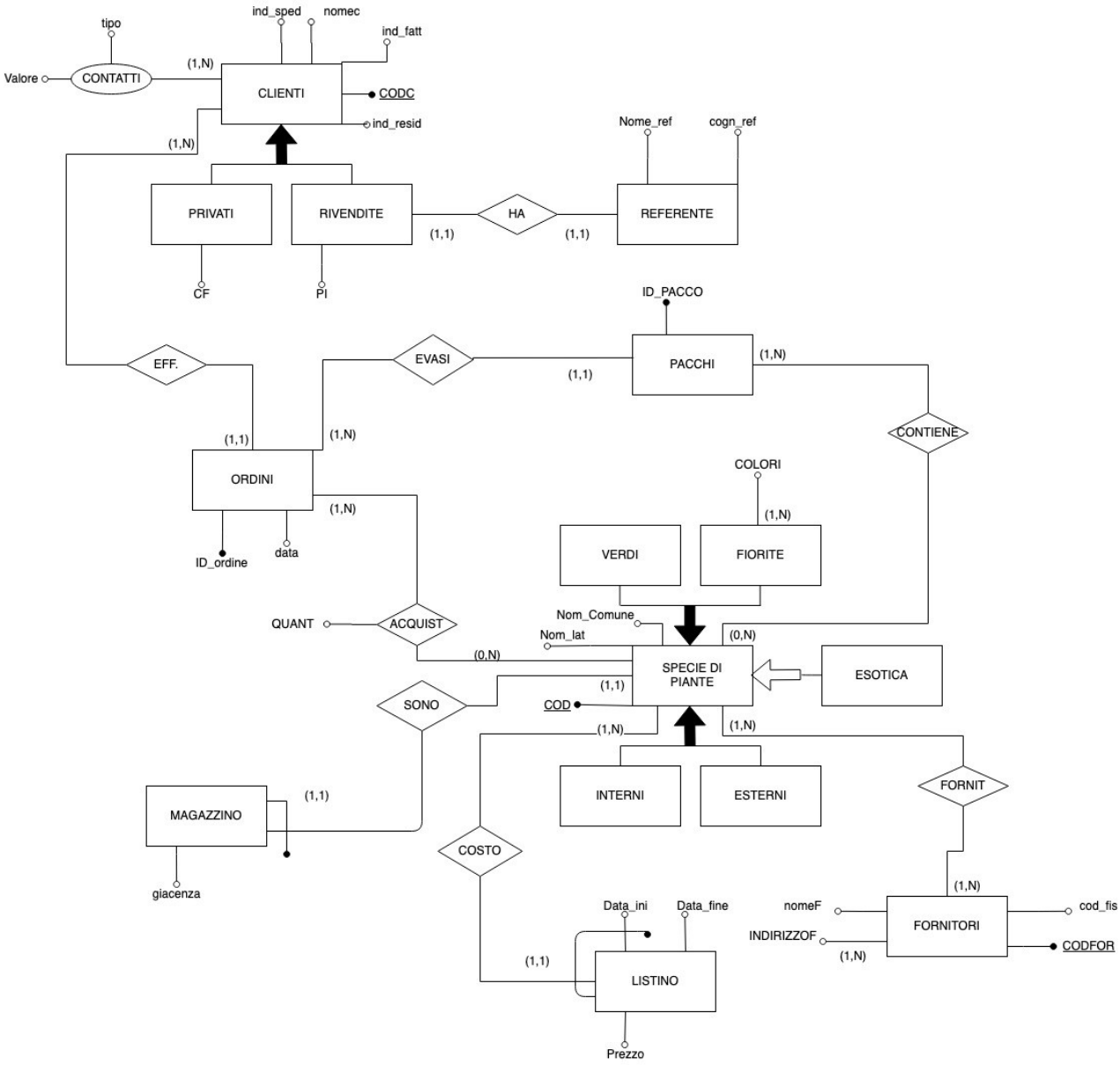


----------PACCHI----------

Per evadere l'ordine bisognerà creare un numero arbitrario di pacchi e inserirci le Specie di piante. Ogni pacco sarà identificato da un proprio codice univoco e conterrà solo piante di un determinato ordine.



Integrazione finale



Regole aziendali

1. Non è possibile aprire un ordine 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.
3. La giacenza nel magazzino si ottiene sottraendo la quantità di piante vendute in un ordine dal totale.
4. Un pacco può contenere solo piante di un determinato ordine.

Dizionario dei dati

Entità	Descrizione	Attributi	Identificatori
Specie di Piante	Tabella che tiene memorizzate tutte le specie gestite dall'azienda	Nom_comune, Nom_latino, COD	COD
Listino	Tiene traccia dei prezzi assunti nel tempo da ciascuna specie di piante	Prezzo, Dataini, Datafin, CODP	CODP, Dataini
Magazzino	Indica il numero di pezzi disponibili della specifica pianta.	Pianta, giacenz	Pianta
Fornitori	Memorizza tutti dati relativi ai fornitori.	CODFOR, codfis, nomef, indirizzof, piantaforn	CODFOR
Ordini	Tiene traccia di tutti gli ordini eseguiti da ciascun cliente.	ID_ordine, data, cliente	ID_ordine
Clienti	Memorizza tutti i dati relativi ai clienti	CODC, nomec, ind_fatt, ind_resid, contatti(tipo, valore) , ind_sped	CODC
Pacchi	Tiene traccia dei pacchi dove vengono evasi gli ordini.	Id_pacco, NUM_ORD	Id_pacco
Referente	Referente di una rivendita	NOME_REF, COGN_REF, COD_CLI	COD_CLI
Verdi	Piante verdi	COD	COD
Fiorite	Piante fiorite	COD, colorazioni	COD
Esotica	Piante esotiche	COD	COD
Interni	Piante da interni	COD	COD
Esterni	Piante da esterni	COD	COD
Privati	Clienti Privati	CODC, cf	CODC
Rivendite	Clienti rivendite	CODC, pi	CODC
Acquist	Relazione di piante acquistate in un ordine	Piante, quant, Ordine	Ordine, Piante

4. Progettazione logica

Volume dei dati

Concetto nello schema	Tipo[1]	Volume atteso
SPECIE DI PIANTE	E	1.000
LISTINO	E	2.000
MAGAZZINO	E	1.000
FORNITORI	E	50
ORDINI	E	9.000
PACCHI	E	18.000
CLIENTI	E	3.000
PRIVATI	E	1.200
RIVENDITE	E	1.800
REFERENTE	E	1.800
ESOTICA	E	50
VERDI	E	350
FIORITE	E	650
INTERNI	E	200
ESTERNI	E	800
Fornit	R	1.500
Acquist	R	54.000
Evasi	R	18.000
Eff.	R	9.000
Ha	R	1.800
Contiene	R	54.000

Assumo media 6 specie di piante a ordine. Acquist=6*ordini=54.000

Assumo in media 3 ordine per cliente. ORDINI = CLIENTI *3= 9.000

Assumo che ogni fornitore in media fornisce 30 specie.

In ogni ordine mediamente viene evaso in 2 pacchi. ORDINI*2=18.000

Ogni Pacco mediamente ha 3 Specie. 3*18.000=54.000

Tavola delle operazioni

1 Indicare con E le entità, con R le relazioni

Cod.	Descrizione	Frequenza attesa
01	Aggiungere specie di pianta	1/mese
02	Visualizzare specie di pianta	300/mese
03	Visualizza Colorazioni Specie	60/mese
04	Modificare Prezzo pianta	15/mese
05	Visualizzare Storico prezzi pianta	10/mese
06	Aggiungere ordine	200/mese
07	Aggiungere cliente privato	8/mese
07-bis	Aggiungere cliente rivendita	12/mese
08	Aggiungere piante a pacco	1.200/mese
09	Visualizzare ordini di un cliente	10/mese
10	Visualizzare Pacco	1.200/mese
11	Visualizzare ordine	200/mese
12	Aggiornare Giacenza Specie	100/mese
13	Aggiungi colorazione	2/mese
14	Aggiungi Contatti	15/mese
15	Aggiungi Pacco	400/mese
16	Aggiungi pianta nell'ordine	1.200/mese
17	Scegli contatto preferito	10/mese
18	Visualizza fornitori di una specie	100/mese
19	Visualizza Piante da inserire	400/mese
20	Visualizza costo totale dell'ordine	200/mese
21	Aggiungi Fornitore a Specie di Pianta	1/mese

Costo delle operazioni

Operazione 01␣ costo 15 * 1/MESE= 15/MESE			
Concetto	**Costrutto**	**Accessi**	**Tipo**
SPECIE DI PIANTA	E	1	S
VERDI/FIORITE	E	1	S
INTERNI/ESTERNI	E	1	S
ESOTICA	E	1	S
LISITNO	E	1	S
MAGAZZINO	E	1	S
FORNIT	R	1	S
FORNITORI	E	1	L
Operazione 04␣ costo 3 * 15/MESE = 45/MESE			
Concetto	**Costrutto**	**Accessi**	**Tipo**
LISTINO	E	1	S

SPECIE DI PIANTA	E	1	L
Operazione 06⊔ costo 3 * 200/MESE = 600/MESE			
Concetto	**Costrutto**	**Accessi**	**Tipo**
ORDINI	E	1	S
CLIENTI	E	1	L
Operazione 07⊔ costo 6 * 8/MESE = 48/MESE			
Concetto	**Costrutto**	**Accessi**	**Tipo**
CLIENTE	E	1	S
PRIVATI	E	1	S
CONTATTI	E	1	S
Operazione 07bis⊔ costo 8 * 12/MESE = 96/MESE			
Concetto	**Costrutto**	**Accessi**	**Tipo**
REFERENTE	E	1	S
RIVENDITE	E	1	S
CLIENTE	E	1	S
CONTATTI	E	1	S
Operazione 08⊔ costo 12 * 1.200/MESE = 14.400/MESE			
Concetto	**Costrutto**	**Accessi**	**Tipo**
PACCHI	E	1	L
SPECIE DI PIANTA	E	1	L
CONTIENE	R	3	S
ORDINI	E	1	L
ACQUIST	R	3	L
Operazione 12⊔ costo 3 * 100/MESE = 300/MESE			
Concetto	**Costrutto**	**Accessi**	**Tipo**
SPECIE DI PIANTE	E	1	L
MAGAZZINO	E	1	S
Operazione 13⊔ costo 3 * 2/MESE = 6/MESE			
Concetto	**Costrutto**	**Accessi**	**Tipo**
SPECIE DI PIANTE	E	1	L
FIORITE	R	1	S
Operazione 14⊔ costo 3* 15/MESE = 45/MESE			
Concetto	**Costrutto**	**Accessi**	**Tipo**
CLIENTI	E	1	L
CONTATTI	E	1	S
Operazione 15⊔ costo 3 * 400/MESE = 1.200/MESE			
Concetto	**Costrutto**	**Accessi**	**Tipo**
ORDINI	E	1	L
PACCHI	E	1	S

Operazione 16⊔ costo 19 * 1.200 = 22.800/MESE			
Concetto	**Costrutto**	**Accessi**	**Tipo**
SPECIE DI PIANTE	E	6	L
ORDINI	E	1	L
ACQUIST	R	6	S
Operazione 17⊔ costo 3 * 10/MESE = 30/MESE			
Concetto	**Costrutto**	**Accessi**	**Tipo**
CLIENTI	E	1	L
CONTATTI	E	1	S
Operazione 21⊔ costo 4 * 1/MESE = 4/MESE			
Concetto	**Costrutto**	**Accessi**	**Tipo**
FORNIT	R	1	S
FORNITORI	E	1	L
SPECIE DI PIANTE	E	1	L

Ristrutturazione dello schema E-R

La generalizzazione totale interni e esterni viene trasformata in un attributo(tipo) dell'entità specie di piante.

La generalizzazione parziale esotica viene trasformata in un attributo di Specie di Piante.

La generalizzazione VERDI viene eliminata, in quanto se una pianta non presenta colore è verde, invece per FIORITE viene creata un'entità e viene messa in relazione con Specie di Piante con cardinalità (0,N) dove si terrà traccia di tutte le eventuali colorazioni.

L'operazione 1 costerà 9/mese anziché 15/mese diminuendo del 40%.

La generalizzazione totale di PRIVATI e RIVENDITE viene raggruppata in un'unica entità (CLIENTI) e come chiave primaria anziché utilizzare un codice interno verrà utilizzato il codice fiscale per i privati e la Partita IVA per le rivendite. Inoltre verrà messa la relazione di REFERENTE con CLIENTI con cardinalità (0,1).

Per attributi multi-valore INDIRIZZO dei Fornitori e CONTATTI dei Clienti vengono create entità a parte.

L'entità COLORE viene creata per evitare numerosi valori null nella tabella (35% delle piante sono verdi).

RIVENDITE e PRIVATI vengono uniti per guadagnare in termini di memoria, inoltre riduce le operazioni 7 e 7/bis.

L'operazione 7 adesso costerà 32/mese anziché 48/mese diminuendo del 33,3%.

L'operazione 7/bis costerà 72/mese anizché 96/mese diminuendo del 25%.

REFERENTE viene creato per una rimozione di valori null (dato che si assume che il 40% degli utenti siano privati quindi non abbino un referente).

Trasformazione di attributi e identificatori

La chiave primaria di CLIENTI sarà in codice fiscale o la PARTITA IVA del cliente.

MAGAZZINO avrà come chiave esterna Pianta, che sarà il codice della pianta.

LISTINO avrà come chiave la tupla Pianta_listino e Data_inizio.

PACCHI e ORDINI avranno entrambi degli identificativi assegnati dall'utente.

Traduzione di entità e associazioni

Specie di Piante (COD_Pianta, Nom_latino, Nom_comune, Tipo, esotica)

Colore (Colore, IdFiorite)

Listino (Prezzo, Data_inizio, Data_fine, Pianta_listino)

Magazzino (Giacenza, Pianta)

Fornit (Pianta_Fornita, Fornitore)

Fornitori (idFornitori, cod_fis, Nome_fornitore)

Indirizzi (indirizzo, forn)

Acquist (Specie, ordine, quantita)

Ordini (idOrdine, dataAcqu, cliente)

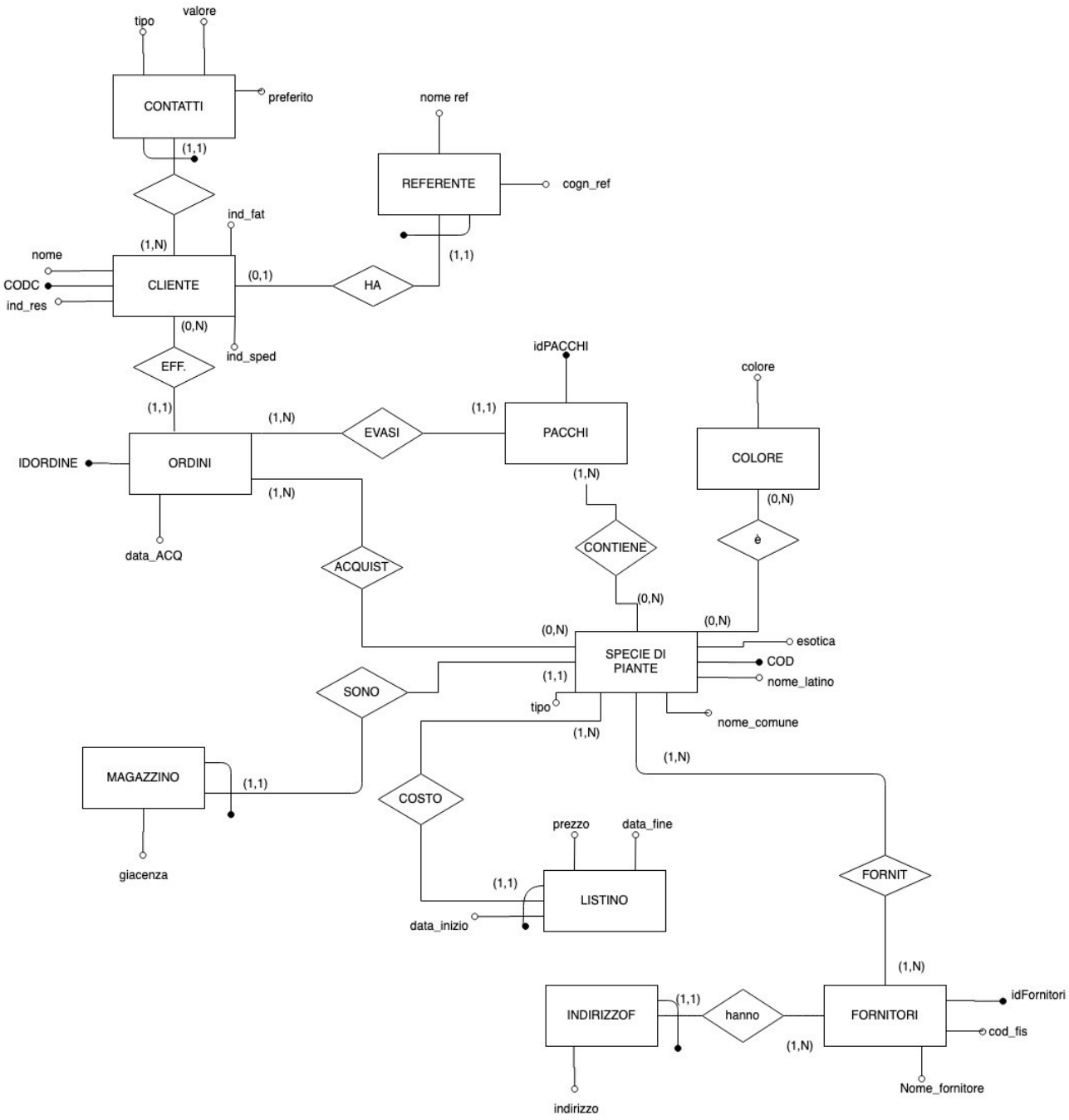
Pacchi (idPacco, ordine)

Contiene (pacco, pianta_pacco)

Clienti (COD_C, Nome, ind_fatt, ind_resid, ind_sped)

Referente (Nome, Cognome, rivendita)

Contatti (preferito ,tipo, valore, cliente)



NB: CONTATTI ha come id la tupla (valore, cliente).

Normalizzazione del modello relazionale

Le tabelle sono tutte 3NF.

5. Progettazione fisica

Utenti e privilegi

----------UTENTI----------

1. **Manager :** Colui che gestisce l'azienda, le relative specie di piante e i loro prezzi.
2. **Operatore :** Colui che registra nuovi clienti, prende nuovi ordini e prepara i pacchi.

----------PRIVILEGI----------

1. **Manager:**
   - Aggiungere nuove specie di piante e aggiungere le varie colorazioni;
   - Modificare i prezzi delle piante e visualizzarne lo storico;
   - Visualizzare tutti i Fornitori di una specie;
   - Creare nuovi utenti per l'utilizzo del DB;
   - Visualizzare le Specie di Piante ed eventualmente le loro colorazioni tramite il loro codice;
   - Aggiornare le giacenze delle Specie;
   - Assegna un fornitore alla Specie di Pianta;
   - Visualizzare il costo totale dell'ordine.
2. **Operatore:**
   - Creare un nuovo ordine;
   - Aggiungere specie ad ordine;
   - Registrare nuovi clienti;
   - Aggiungere contatti e selezionare un preferito;
   - Creare nuovi pacchi, visualizzare le piante inserite e quelle ancora da inserire;
   - Visualizzare un ordine e tutti gli ordini di un cliente;
   - Visualizzare Specie, le loro colorazioni, il loro storico dei prezzi e il costo totale di un ordine.

Strutture di memorizzazione

Tabella Cliente		
Attributo	**Tipo di dato**	**Attributi[2]**
COD_C	VARCHAR(16)	PK, NN
Nome	VARCHAR(45)	NN

Ind_fatt	VARCHAR(45)	NN
Ind_sped	VARCHAR(45)	NN
Ind_resid	VARCHAR(45)	NN

Tabella Colore		
Attributo	**Tipo di dato**	**Attributi**[3]
ID_colore	INT	PK, NN, AI
Colore	VARCHAR(20)	NN

Tabella Colore_has_Specie_di_Piante		
Attributo	**Tipo di dato**	**Attributi**[4]
Colore_ID_colore	INT	PK, NN
ID_Fiorita	VARCHAR(6)	PK, NN

Tabella Contatti		
Attributo	**Tipo di dato**	**Attributi**[5]
Cliente_COD_C	VARCHAR(16)	PK, NN
Valore	VARCHAR(45)	PK,NN
tipo	ENUM(“email”, “cel”, “tel”)	NN
preferito	TINYINT	NN

Tabella Fornitori		
Attributo	**Tipo di dato**	**Attributi**[6]
idFornitori	INT	PK, NN
cod_fis	VARCHAR(16)	NN
Nome_fornitore	VARCHAR(45)	NN

Tabella INDIRIZZI		
Attributo	**Tipo di dato**	**Attributi**[7]

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.

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

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

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

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

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

indirizzo	VARCHAR(64)	PK, NN
Fornitori_idFornitori	INT	PK,NN

Tabella LISTINO		
Attributo	**Tipo di dato**	**Attributi**[8]
Pianta_listino	VARCHAR(6)	PK, NN
Data_inizio	DATETIME	PK,NN
Data_fine	DATETIME	
Prezzo	DOUBLE	NN, UN

Tabella MAGAZZINO		
Attributo	**Tipo di dato**	**Attributi**[9]
Pianta	VARCHAR(6)	PK, NN
Giacenza	INT	NN, UN

Tabella ORDINI		
Attributo	**Tipo di dato**	**Attributi**[10]
idORDINI	INT	PK, NN
dataAcq	DATETIME	NN
Cliente	VARCHAR(16)	NN

Tabella ORDINI_has_Specie_di_Piante		
Attributo	**Tipo di dato**	**Attributi**[11]
ORDINI_idORDINI	INT	PK, NN
Specie	VARCHAR(6)	PK,NN
Quantita	INT	NN

Tabella PACCHI		
Attributo	**Tipo di dato**	**Attributi**[12]

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

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

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

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

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

idPACCHI	INT	PK, NN
Ordine	INT	NN

Tabella PACCHI_has_Specie_di_Piante		
Attributo	**Tipo di dato**	**Attributi**[13]
PACCHI_idPACCHI	INT	PK, NN
Pianta_Pacco	VARCHAR(6)	PK,NN

Tabella Referente		
Attributo	**Tipo di dato**	**Attributi**[14]
Rivendita	VARCHAR(11)	PK, NN
Cognome_referente	VARCHAR(45)	PK,NN
Nome_referente	VARCHAR(45)	NN

Tabella Specie_di_Piante		
Attributo	**Tipo di dato**	**Attributi**[15]
COD_Pianta	VARCHAR(6)	PK, NN
Nom_latino	VARCHAR(45)	NN
Nome_comune	VARCHAR(45)	NN
Tipo	ENUM(“interno”,”esterno”)	NN
esotica	TINYINT	NN

Tabella Specie_di_Piante_has_Fornitori		
Attributo	**Tipo di dato**	**Attributi**[16]
Pianta_Fornita	VARCHAR(6)	PK, NN
Fornitore	INT	PK, NN

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

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

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

16 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 utenti		
Attributo	**Tipo di dato**	**Attributi[17]**
username	VARCHAR(45)	PK, NN
password	VARCHAR(32)	NN
ruolo	ENUM("Manager", "operatore")	NN

Indici

Tabella Cliente	
Indice PRIMARY	**Tipo[18]:**
COD_C	PR
Tabella Colore	
Indice PRIMARY	**Tipo[19]:**
ID_colore	PR
Tabella Colore_has_Specie_di_Piante	
Indice PRIMARY	**Tipo[20]:**
Colore_ID_colore ID_Fiorita	PR
Indice fk_Colore_has_Specie_di_Piante_Specie_di_Piante1_idx	**Tipo:**
ID_Fiorita	IDX
Indice fk_Colore_has_Specie_di_Piante_Colore1_idx	**Tipo:**
Colore_ID_colore	IDX
Tabella Contatti	
Indice PRIMARY	**Tipo[21]:**
Valore Cliente_COD_C	PR
Indice fk_Contatti_Cliente1_idx	**Tipo** :
Client_COD_C	IDX
Tabella Fornitori	
Indice PRIMARY	**Tipo[22]:**
idFornitori	PR

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

18 IDX = index, UQ = unique, FT = full text, PR = primary.

19 IDX = index, UQ = unique, FT = full text, PR = primary.

20 IDX = index, UQ = unique, FT = full text, PR = primary.

21 IDX = index, UQ = unique, FT = full text, PR = primary.

Tabella INDIRIZZI	
Indice PRIMARY	**Tipo[23]:**
indirizzo Fornitori_idFornitori	PR
Indice fk_INDIRIZZI_Fornitori1_idx	**Tipo[24]:**
Fornitori_idfornitori	IDX
Tabella LISTINO	
Indice PRIMARY	**Tipo[25]:**
Data_inizio Pianta_listino	PR
Indice fk_LISTINO_Specie_di_Piante_idx	**Tipo[26]:**
Pianta_listino	IDX
Tabella MAGAZZINO	
Indice PRIMARY	**Tipo[27]:**
Pianta	PR
Tabella ORDINI	
Indice PRIMARY	**Tipo[28]:**
idORDINI	PR
Indice fk_ORDINI_Cliente1_idx	**Tipo[29]:**
Cliente	IDX
Tabella ORDINI_has_Specie_di_Piante	
Indice PRIMARY	**Tipo[30]:**
ORDINI_idORDINI Specie	PR
Indice fk_ORDINI_has_Specie_di_Piante_Specie_di_Piante1_idx	**Tipo[31]:**
Specie	IDX
Indice fk_ORDINI_has_Specie_di_Piante_ORDINI1_idx	**Tipo[32]:**
ORDINI_idORDINI	IDX

23 IDX = index, UQ = unique, FT = full text, PR = primary.

24 IDX = index, UQ = unique, FT = full text, PR = primary.

25 IDX = index, UQ = unique, FT = full text, PR = primary.

26 IDX = index, UQ = unique, FT = full text, PR = primary.

27 IDX = index, UQ = unique, FT = full text, PR = primary.

28 IDX = index, UQ = unique, FT = full text, PR = primary.

29 IDX = index, UQ = unique, FT = full text, PR = primary.

30 IDX = index, UQ = unique, FT = full text, PR = primary.

31 IDX = index, UQ = unique, FT = full text, PR = primary.

Tabella PACCHI	
Indice PRIMARY	**Tipo[33]:**
idPACCHI	PR
Indice fk_PACCHI_ORDINI1_idx	**Tipo[34]:**
Ordine	IDX
Tabella PACCHI_has_Specie_di_Piante	
Indice PRIMARY	**Tipo[35]:**
PACCHI_idPACCHI Pianta_Pacco	PR
Indice fk_PACCHI_has_Specie_di_Piante_Specie_di_Piante1_idx	**Tipo[36]:**
Pianta_Pacco	IDX
Indice fk_PACCHI_has_Specie_di_Piante_PACCHI1_idx	**Tipo[37]:**
PACCHI_idPACCHI	IDX
Tabella Referente	
Indice PRIMARY	**Tipo[38]:**
Rivendita Cognome_referente	PR
Indice fk_Referente_Cliente1_idx	**Tipo[39]:**
Rivendita	IDX
Tabella Specie_di_Piante	
Indice PRIMARY	**Tipo[40]:**
COD_Pianta	PR
Tabella Specie_di_Piante_has_Fornitori	
Indice PRIMARY	**Tipo:**
Pianta_Fortnita Fornitore	PR
Indice fk_Specie_di_Piante_has_Fornitori_Fornitori1_idx	**Tipo:**
Fornitore	IDX
Indice fk_Specie_di_Piante_has_Fornitori_Specie_di_Piante1_idx	**Tipo:**
Pianta_Fornita	IDX
Tabella utenti	
Indice PRIMARY	**Tipo:**
username	PR

Trigger

Nel DB è stato implementato un solo TRIGGER BEFORE INSERT nella tabella ORDINI_has_Specie_di_Piante, che controlla se può essere effettuato un ordine di una certa quantità in base alla Giacenza in Magazzino e in caso affermativo scala dalla Giacenza la quantità ordinata.

```
CREATE DEFINER = CURRENT_USER TRIGGER
`mydb`.`ORDINI_has_Specie_di_Piante_BEFORE_INSERT` BEFORE INSERT ON
`ORDINI_has_Specie_di_Piante` FOR EACH ROW
BEGIN
        if((NEW.Quantita)>(select Giacenza
                                from mydb.MAGAZZINO, Specie_di_Piante
                                where mydb.MAGAZZINO.Pianta = Specie_di_Piante.COD_Pianta and
NEW.Specie = Specie_di_Piante.COD_Pianta))
                then
                signal sqlstate '45000' set message_text=' Non ci sono abbastanza Piante in
magazzino';
        else update MAGAZZINO
             set Giacenza = Giacenza - NEW.Quantita
              where MAGAZZINO.Pianta = NEW.Specie;
        end if;
END
```

Eventi

Non ci sono eventi specifici.

Stored Procedures e transazioni

1. **Aggiornare Giacenze:**

```
CREATE PROCEDURE `Aggiornare_Giacenze`(in Specie_pianta varchar(6),in giac int)
BEGIN
declare exit handler for sqlexception
begin
   rollback;
   resignal;
```

```
end;
set transaction isolation level repeatable read ;
   start transaction;
   if(Specie_pianta in (select COD_Pianta
                        from Specie_di_Piante))
               then
               update MAGAZZINO
               SET Giacenza = Giacenza + giac
               where Pianta = Specie_pianta ;
         else
   signal sqlstate '45002' set message_text=' Pianta non trovata';
   end if;
   commit;
END
```

2. Aggiungere Specie di Piante

```
CREATE PROCEDURE  `aggiungere_specie_pianta`(in codice varchar(6), in n_com varchar(45), in n_lat varchar(45), in posizione enum('interno', 'esterno'), in eso TINYINT, in quantita int, in Costo double, in cod_for int)
BEGIN
set transaction isolation level repeatable read;
start transaction;
        insert into Specie_di_Piante(COD_Pianta, Nom_latino, Nom_comune, Tipo, esotica)values (codice, n_lat, n_com, posizione, eso) ;
        insert into LISTINO(Prezzo, Data_inizio, Pianta_listino) values (Costo, NOW() ,codice);
        insert into MAGAZZINO(Giacenza, Pianta) values (quantita, codice);
        insert into Specie_di_Piante_has_Fornitori(Pianta_Fornita, Fornitore) values (codice, cod_for);
commit;
```

END

3. **Aggiungere Cliente Privato**

CREATE PROCEDURE `Aggiungi_cliente_privato`(in cod varchar(16), in nom varchar(45), in fat varchar(45),in sped varchar(45), in res varchar(45),in tipo_contatto ENUM('email', 'cel', 'tel'), in valore_contatto varchar(45))
BEGIN
    insert into Cliente(COD_C, Nome, ind_fatt, ind_sped, ind_resid) values(cod, nom, fat, sped, res);
    insert into Contatti(tipo, Valore, preferito, Cliente_COD_C) values(tipo_contatto, valore_contatto, 1, cod);
END

4. **Aggiungere cliente Rivendita**

CREATE PROCEDURE `Aggiungi_cliente_rivendita`(in cod varchar(11), in nom varchar(45), in fat varchar(45),in sped varchar(45), in res varchar(45),in tipo_contatto ENUM('email', 'cel', 'tel'), in valore_contatto varchar(45), in n_ref varchar(45), in c_ref varchar(45))
BEGIN

---

22 IDX = index, UQ = unique, FT = full text, PR = primary.

32 IDX = index, UQ = unique, FT = full text, PR = primary.

33 IDX = index, UQ = unique, FT = full text, PR = primary.

34 IDX = index, UQ = unique, FT = full text, PR = primary.

35 IDX = index, UQ = unique, FT = full text, PR = primary.

36 IDX = index, UQ = unique, FT = full text, PR = primary.

37 IDX = index, UQ = unique, FT = full text, PR = primary.

38 IDX = index, UQ = unique, FT = full text, PR = primary.

39 IDX = index, UQ = unique, FT = full text, PR = primary.

40 IDX = index, UQ = unique, FT = full text, PR = primary.

```
        insert into Cliente(COD_C, Nome, ind_fatt, ind_sped, ind_resid) values(cod, nom, fat, sped,
        res);
        insert into Contatti(tipo, Valore, preferito, Cliente_COD_C) values(tipo_contatto,
        valore_contatto,1, cod);
        insert into Referente(Nome_referente, Cognome_referente, Rivendita) values (n_ref, c_ref,
        cod);
END
```

5. **Aggiungere colore a Pianta**

```
CREATE PROCEDURE `Aggiungi_colore_pianta`(in colorazione varchar(20), in pianta varchar(6) )
BEGIN
        if((select count(*)
            from Colore
            where Colore =colorazione) = 0)
            then insert into Colore (Colore) values (colorazione) ;
        end if;
       insert into Colore_has_Specie_di_Piante(id_Fiorita, Colore_ID_colore) values
 (pianta, (select ID_colore
            from Colore
            where Colore =colorazione));
END
```

6. **Aggiungere Contatto**

```
CREATE PROCEDURE `aggiungi_contatto`(in cod_Cliente varchar(16), in tipo_contatto
ENUM('email', 'cel', 'tel'), in valore_contatto varchar(45))
BEGIN
        insert into Contatti(tipo, Valore, preferito, Cliente_COD_C) values(tipo_contatto,
        valore_contatto, 0, cod_Cliente);
END
```

7. **Aggiungere ordine**

```
CREATE PROCEDURE `Aggiungi_ordine`(in id int, in cod_cliente varchar(16))
BEGIN
        insert into ORDINI(idORDINI, dataAcq, Cliente) values (id, NOW(), cod_cliente);
```

```
END
```

8. Aggiungere Pacco

```
CREATE PROCEDURE `Aggiungi_pacco`(in ordine int, in pacco int)
BEGIN
        insert into PACCHI(idPACCHI, Ordine) values (pacco, ordine);
END
```

9. Aggiungere Pianta a Pacco

```
CREATE PROCEDURE `Aggiungi_piante_a_pacco`(in idpacco int,in pianta varchar(6))
BEGIN
declare exit handler for sqlexception
begin
  rollback;
   resignal;
end;
set transaction isolation level repeatable read;
start transaction;
if (pianta in (select Specie
                  from ORDINI_has_Specie_di_Piante, ORDINI, PACCHI
                  where ORDINI_has_Specie_di_Piante.ORDINI_idORDINI = ORDINI.idORDINI
and ORDINI.idORDINI = PACCHI.ordine and PACCHI.idPACCHI = idpacco
            and ORDINI_has_Specie_di_Piante.Specie NOT IN (select Pianta_Pacco as Specie
                                                           from PACCHI_has_Specie_di_Piante
                                                           where idpacco = PACCHI_idPACCHI)))
        THEN
        insert into PACCHI_has_Specie_di_Piante(PACCHI_idPACCHI, Pianta_Pacco) values
(idpacco, pianta);
   else
  signal sqlstate '45001' set message_text=' Pianta non valida';
   END IF;
commit;
END
```

10. Aggiungere Pianta a ordine

```
CREATE PROCEDURE `agg_piante_ordine`(in ordine int, in pianta VARCHAR(6), in quant int)
BEGIN
	insert into ORDINI_has_Specie_di_Piante(ORDINI_idORDINI, Specie, Quantita) values (ordine, pianta, quant);
END
```

11. Creare nuovo utente

```
CREATE PROCEDURE `crea_utente`(IN username VARCHAR(45), IN pass VARCHAR(32), IN ruolo enum('Manager','operatore'))
BEGIN
	insert into utenti(username,password, ruolo) VALUES(username, MD5(pass), ruolo);
END
```

12. Login

```
CREATE PROCEDURE `login`(in var_username varchar(45), in var_pass varchar(45), out var_role INT)
BEGIN
	declare var_user_role ENUM('operatore', 'Manager');

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

	-- See the corresponding enum in the client
		if var_user_role = 'operatore' then
			set var_role = 1;
		elseif var_user_role = 'Manager' then
			set var_role = 2;
		else
			set var_role = 3;
		end if;
END
```

13. Modificare Prezzo Pianta

```
CREATE PROCEDURE `Modificare_prezzo_pianta`(in Specie_pianta varchar(6), in Nuovo_prezzo double)
BEGIN
declare exit handler for sqlexception
begin
        rollback;
   resignal;
end;
set transaction isolation level serializable;
start transaction;
if((select count(*)
        from Specie_di_Piante
        where Specie_pianta = COD_Pianta)>0)
   then
   update LISTINO
   set Data_fine = NOW()
   where Pianta_listino = Specie_pianta and Data_fine is null;
      insert into LISTINO(Prezzo, Data_inizio, Pianta_listino) values (Nuovo_prezzo,NOW() ,Specie_pianta);
else
signal sqlstate '45002'set message_text="Pianta non trovata";
end if;
commit;
END
```

14. Scegliere Contatto Preferito

```
CREATE PROCEDURE `scegli_contatto_preferito`(in cliente varchar(16), in valore_contatto varchar(45))
BEGIN
declare exit handler for sqlexception
begin
        rollback;
   resignal;
end;
```

```
if(valore_contatto in (select Valore from Contatti where Cliente_COD_C = cliente))
then
        if ((select count(*)
                from Contatti
      where cliente = Cliente_COD_C and preferito = 1)>0)
      then update Contatti set preferito = 0 where cliente = Cliente_COD_C;
        end if;
        update Contatti set preferito = 1 where valore_contatto = Valore and cliente =
Cliente_COD_C;
else
signal sqlstate '45004' set message_text='Contatto non trovato';
end if;
END
```

15. Visualizzare Fornitori di una Specie

```
CREATE PROCEDURE `visualizza_fornitori_pianta`(in pianta varchar(6) )
BEGIN
declare exit handler for sqlexception
begin
   rollback;
   resignal;
end;
if((select count(*)
        from Specie_di_Piante
   where pianta = COD_Pianta)>0)
   then
select idFornitori, cod_fis as 'codice fiscale', Nome_fornitore as 'Nome', Pianta_Fornita as 'Pianta',
indirizzo
   from Fornitori, Specie_di_Piante_has_Fornitori, INDIRIZZI
  where pianta = Specie_di_Piante_has_Fornitori.Pianta_Fornita and
Specie_di_Piante_has_Fornitori.Fornitore = Fornitori.idFornitori and Fornitori.idFornitori =
INDIRIZZI.Fornitori_idFornitori;
else
 signal sqlstate '45002'set message_text="Pianta non trovata";
```

```
 end if;
END
```

16. Visualizzare ordine

```
CREATE PROCEDURE `Visualizza_ordine`(IN ordine int)
BEGIN
declare exit handler for sqlexception
begin
        rollback;
   resignal;
end;
set transaction isolation level repeatable read;
start transaction;
if((select count(*) from ORDINI where idORDINI = ordine)>0)
   then
        if((select  count(*)
        from ORDINI, ORDINI_has_Specie_di_Piante, Cliente, Contatti, Referente
        where ordine = ORDINI.idORDINI and Cliente.COD_C = ORDINI.Cliente and
Cliente.COD_C = Contatti.Cliente_COD_C and
ORDINI_has_Specie_di_Piante.ORDINI_idORDINI = ordine  and Cliente.COD_C =
Referente.Rivendita) > 0)
        then
        select  dataAcq, ORDINI_has_Specie_di_Piante.Specie, Quantita,Cliente.COD_C,
Cliente.Nome, Cliente.ind_sped, Contatti.tipo, Contatti.Valore, Referente.Nome_referente,
Referente.Cognome_referente
        from ORDINI, ORDINI_has_Specie_di_Piante, Cliente, Contatti, Referente
        where ordine = ORDINI.idORDINI and Cliente.COD_C = ORDINI.Cliente and
Cliente.COD_C = Contatti.Cliente_COD_C and
ORDINI_has_Specie_di_Piante.ORDINI_idORDINI = ordine and Cliente.COD_C =
Referente.Rivendita and Contatti.preferito =1;
   else
        select  distinct dataAcq, ORDINI_has_Specie_di_Piante.Specie, Quantita,Cliente.COD_C,
Cliente.Nome, Cliente.ind_sped, Contatti.tipo, Contatti.Valore
        from ORDINI, ORDINI_has_Specie_di_Piante, Cliente, Contatti
```

```
        where (ordine = ORDINI.idORDINI and Cliente.COD_C = ORDINI.Cliente and
Cliente.COD_C = Contatti.Cliente_COD_C and
ORDINI_has_Specie_di_Piante.ORDINI_idORDINI = ordine and Contatti.preferito=1);
   end if;
else
signal sqlstate '45003' set message_text='Ordine non trovato';
end if;
commit;
END
```

17. Visualizzare ordini effettuati da un cliente

```
CREATE PROCEDURE `Visualizza_ordini_cliente`(IN COD_CLI VARCHAR(16))
BEGIN
declare exit handler for sqlexception
begin
   rollback;
   resignal;
end;
set transaction isolation level repeatable read;
start transaction;
if((select count(*) from Cliente where COD_CLI = COD_C)>0)
   then
        select distinct idORDINI, dataAcq, ORDINI_has_Specie_di_Piante.Specie, Quantita
        from ORDINI, ORDINI_has_Specie_di_Piante
        where COD_CLI = Cliente and idORDINI = ORDINI_idORDINI;
else
        signal sqlstate '45005' set message_text="Cliente non trovato";
end if;
 commit;
END
```

18. Visualizzare Specie di Pianta

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `Visualizza_Pianta`(IN var_Pianta
VARCHAR(6))
BEGIN
declare exit handler for sqlexception
```

```
begin
  rollback;
   resignal;
end;
if((select count(*) from Specie_di_Piante  where COD_Pianta = var_Pianta)>0)
   then
  select distinct Nom_latino, Nom_comune, Tipo, esotica, Giacenza
   from Specie_di_Piante, MAGAZZINO
   where var_Pianta = COD_Pianta and Pianta=var_Pianta;
else
        signal sqlstate '45002' set message_text ="Pianta non trovata";
end if;
END
```

19. Visualizzare Colorazioni Specie

```
CREATE PROCEDURE `Visualizzare_Colorazioni`(IN var_Pianta VARCHAR(6))
BEGIN
declare exit handler for sqlexception
begin
   rollback;
   resignal;
end;
set transaction isolation level repeatable read;
start transaction;
if((select count(*) from Specie_di_Piante  where var_Pianta = COD_Pianta)>0)
   then

   select distinct Nom_latino, Nom_comune, Tipo, esotica, Colore
    from Specie_di_Piante, Colore_has_Specie_di_Piante, Colore
   where var_Pianta = COD_Pianta and var_Pianta = ID_Fiorita and Colore_ID_colore = ID_colore;
else
        signal sqlstate '45002' set message_text='Pianta non trovata';
end if;
commit;
END
```

20. Visualizzare Piante da inserire nei pacchi

```
CREATE PROCEDURE `visualizzare_piante_inserire_pacco`(in ordine int)
BEGIN
declare exit handler for sqlexception
begin
   rollback;
   resignal;
end;
set transaction isolation level repeatable read;
start transaction;
        if((select count(*) from ORDINI where ordine = idORDINI)>0)
      then
        select Specie
        from ORDINI_has_Specie_di_Piante
        where ORDINI_idORDINI = ordine and Specie NOT IN (select Pianta_Pacco as Specie
                                                          from PACCHI_has_Specie_di_Piante, PACCHI
                          where PACCHI.ordine = ordine and idPACCHI = PACCHI_idPACCHI);
        else
                signal sqlstate '45003' set message_text='Ordine non trovato';
end if;
commit;
END
```

21. Visualizzare Piante contenute in un pacco

```
CREATE PROCEDURE `Visualizzare_piante_Pacco`(in id_pacco int)
BEGIN
declare exit handler for sqlexception
begin
  rollback;
   resignal;
end;
```

```
set transaction isolation level repeatable read;
start transaction;
if((select count(*) from PACCHI where id_pacco= idPACCHI)>0)
    then
   select Pianta_Pacco, Nom_comune, Nom_latino, Ordine, Quantita
    from PACCHI_has_Specie_di_Piante, Specie_di_Piante, PACCHI, ORDINI,
ORDINI_has_Specie_di_Piante
    where PACCHI_idPACCHI = id_pacco and Pianta_Pacco = COD_Pianta and
id_pacco=idPACCHI and Ordine = idORDINI and idORDINI=ORDINI_idORDINI;
 else
    signal sqlstate '45006' set message_text="Pacco non trovato";
    end if;
    commit;
END
```

22. Visualizzare Storico de Prezzi

```
CREATE PROCEDURE `Visualizzare_Storico_Prezzo`(IN var_Pianta VARCHAR(6))
BEGIN
declare exit handler for sqlexception
begin
   rollback;
    resignal;
end;
set transaction isolation level repeatable read;
start transaction;
if((select count(*) from Specie_di_Piante where var_Pianta = COD_Pianta)>0)
    then
    select *
    from LISTINO
    where Pianta_listino = var_Pianta;
else signal sqlstate '45002' set message_text="Pianta non trovata";
end if;
commit;
END
```

23. Visualizzare il costo totale di un ordine

```
CREATE PROCEDURE `visualizza_totale_ordine`(in ordine int)
BEGIN
declare exit handler for sqlexception
begin
  rollback;
   resignal;
end;
   set transaction isolation level read committed;
   start transaction;
   if((select count(*) from ORDINI  where idORDINI = ordine)>0)
      then
                  select sum(Prezzo * quantita) as 'Totale'
                  from LISTINO, Specie_di_Piante, ORDINI_has_Specie_di_Piante, ORDINI
                  where ordine = ORDINI_idORDINI and ordine=idORDINI and
dataAcq>=Data_inizio  and (dataAcq<=Data_fine or Data_fine is null) and Pianta_listino =
COD_Pianta and COD_Pianta = Specie;
         else
                  signal sqlstate '45003' set message_text= 'Ordine non trovato';
   end if;
   commit;
END
```

24. Aggiungere Fornitore ad una specie

```
CREATE PROCEDURE `aggiungi_fornitore_specie`(in cod int, in pianta varchar(6))
BEGIN
        insert into Specie_di_Piante_has_Fornitori(Pianta_Fornita, Fornitore) values(pianta, cod);
END
```

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 mydb
-- -----------------------------------------------------

-- -----------------------------------------------------
-- Schema mydb
-- -----------------------------------------------------
CREATE SCHEMA IF NOT EXISTS `mydb` DEFAULT CHARACTER SET utf8 ;
USE `mydb` ;

-- -----------------------------------------------------
-- Table `mydb`.`Specie_di_Piante`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `mydb`.`Specie_di_Piante` (
  `COD_Pianta` VARCHAR(6) NOT NULL,
  `Nom_latino` VARCHAR(45) NOT NULL,
  `Nom_comune` VARCHAR(45) NOT NULL,
  `Tipo` ENUM("interno", "esterno") NOT NULL,
  `esotica` TINYINT NOT NULL,
  PRIMARY KEY (`COD_Pianta`))
ENGINE = InnoDB;
```

```
-- -----------------------------------------------------
-- Table `mydb`.`LISTINO`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `mydb`.`LISTINO` (
  `Prezzo` DOUBLE UNSIGNED NOT NULL,
  `Data_inizio` DATETIME NOT NULL,
  `Data_fine` DATETIME NULL,
  `Pianta_listino` VARCHAR(6) NOT NULL,
  INDEX `fk_LISTINO_Specie_di_Piante_idx` (`Pianta_listino` ASC) VISIBLE,
  PRIMARY KEY (`Pianta_listino`, `Data_inizio`),
  CONSTRAINT `fk_LISTINO_Specie_di_Piante`
    FOREIGN KEY (`Pianta_listino`)
    REFERENCES `mydb`.`Specie_di_Piante` (`COD_Pianta`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `mydb`.`Colore`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `mydb`.`Colore` (
  `Colore` VARCHAR(20) NOT NULL,
  `ID_colore` INT NOT NULL AUTO_INCREMENT,
  PRIMARY KEY (`ID_colore`))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `mydb`.`Colore_has_Specie_di_Piante`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `mydb`.`Colore_has_Specie_di_Piante` (
```

```
  `Colore_ID_colore` INT NOT NULL,
  `ID_Fiorita` VARCHAR(6) NOT NULL,
  PRIMARY KEY (`Colore_ID_colore`, `ID_Fiorita`),
  INDEX `fk_Colore_has_Specie_di_Piante_Specie_di_Piante1_idx` (`ID_Fiorita` ASC) VISIBLE,
  INDEX `fk_Colore_has_Specie_di_Piante_Colore1_idx` (`Colore_ID_colore` ASC) VISIBLE,
  CONSTRAINT `fk_Colore_has_Specie_di_Piante_Colore1`
    FOREIGN KEY (`Colore_ID_colore`)
    REFERENCES `mydb`.`Colore` (`ID_colore`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Colore_has_Specie_di_Piante_Specie_di_Piante1`
    FOREIGN KEY (`ID_Fiorita`)
    REFERENCES `mydb`.`Specie_di_Piante` (`COD_Pianta`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `mydb`.`Fornitori`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `mydb`.`Fornitori` (
  `idFornitori` INT NOT NULL,
  `cod_fis` VARCHAR(16) NOT NULL,
  `Nome_fornitore` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`idFornitori`))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `mydb`.`Specie_di_Piante_has_Fornitori`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `mydb`.`Specie_di_Piante_has_Fornitori` (
  `Pianta_Fornita` VARCHAR(6) NOT NULL,
```

```
  `Fornitore` INT NOT NULL,
  PRIMARY KEY (`Pianta_Fornita`, `Fornitore`),
  INDEX `fk_Specie_di_Piante_has_Fornitori_Fornitori1_idx` (`Fornitore` ASC) VISIBLE,
    INDEX `fk_Specie_di_Piante_has_Fornitori_Specie_di_Piante1_idx` (`Pianta_Fornita` ASC)
VISIBLE,
  CONSTRAINT `fk_Specie_di_Piante_has_Fornitori_Specie_di_Piante1`
    FOREIGN KEY (`Pianta_Fornita`)
    REFERENCES `mydb`.`Specie_di_Piante` (`COD_Pianta`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Specie_di_Piante_has_Fornitori_Fornitori1`
    FOREIGN KEY (`Fornitore`)
    REFERENCES `mydb`.`Fornitori` (`idFornitori`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `mydb`.`INDIRIZZI`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `mydb`.`INDIRIZZI` (
  `indirizzo` VARCHAR(64) NOT NULL,
  `Fornitori_idFornitori` INT NOT NULL,
  INDEX `fk_INDIRIZZI_Fornitori1_idx` (`Fornitori_idFornitori` ASC) VISIBLE,
  PRIMARY KEY (`indirizzo`, `Fornitori_idFornitori`),
  CONSTRAINT `fk_INDIRIZZI_Fornitori1`
    FOREIGN KEY (`Fornitori_idFornitori`)
    REFERENCES `mydb`.`Fornitori` (`idFornitori`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

```
-- -----------------------------------------------------
-- Table `mydb`.`MAGAZZINO`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `mydb`.`MAGAZZINO` (
  `Giacenza` INT UNSIGNED NOT NULL,
  `Pianta` VARCHAR(6) NOT NULL,
  PRIMARY KEY (`Pianta`),
  CONSTRAINT `fk_MAGAZZINO_Specie_di_Piante1`
    FOREIGN KEY (`Pianta`)
    REFERENCES `mydb`.`Specie_di_Piante` (`COD_Pianta`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `mydb`.`Cliente`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `mydb`.`Cliente` (
  `COD_C` VARCHAR(16) NOT NULL,
  `Nome` VARCHAR(45) NOT NULL,
  `ind_fatt` VARCHAR(45) NOT NULL,
  `ind_sped` VARCHAR(45) NOT NULL,
  `ind_resid` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`COD_C`))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `mydb`.`ORDINI`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `mydb`.`ORDINI` (
  `idORDINI` INT NOT NULL,
  `dataAcq` DATETIME NOT NULL,
```

```
  `Cliente` VARCHAR(16) NOT NULL,
  PRIMARY KEY (`idORDINI`),
  INDEX `fk_ORDINI_Cliente1_idx` (`Cliente` ASC) VISIBLE,
  CONSTRAINT `fk_ORDINI_Cliente1`
    FOREIGN KEY (`Cliente`)
    REFERENCES `mydb`.`Cliente` (`COD_C`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;



-- -----------------------------------------------------
-- Table `mydb`.`ORDINI_has_Specie_di_Piante`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `mydb`.`ORDINI_has_Specie_di_Piante` (
  `ORDINI_idORDINI` INT NOT NULL,
  `Specie` VARCHAR(6) NOT NULL,
  `Quantita` INT NOT NULL,
  PRIMARY KEY (`ORDINI_idORDINI`, `Specie`),
  INDEX `fk_ORDINI_has_Specie_di_Piante_Specie_di_Piante1_idx` (`Specie` ASC) VISIBLE,
      INDEX `fk_ORDINI_has_Specie_di_Piante_ORDINI1_idx` (`ORDINI_idORDINI` ASC)
VISIBLE,
  CONSTRAINT `fk_ORDINI_has_Specie_di_Piante_ORDINI1`
    FOREIGN KEY (`ORDINI_idORDINI`)
    REFERENCES `mydb`.`ORDINI` (`idORDINI`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_ORDINI_has_Specie_di_Piante_Specie_di_Piante1`
    FOREIGN KEY (`Specie`)
    REFERENCES `mydb`.`Specie_di_Piante` (`COD_Pianta`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

```
-- -----------------------------------------------------
-- Table `mydb`.`PACCHI`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `mydb`.`PACCHI` (
  `idPACCHI` INT NOT NULL,
  `Ordine` INT NOT NULL,
  PRIMARY KEY (`idPACCHI`),
  INDEX `fk_PACCHI_ORDINI1_idx` (`Ordine` ASC) VISIBLE,
  CONSTRAINT `fk_PACCHI_ORDINI1`
    FOREIGN KEY (`Ordine`)
    REFERENCES `mydb`.`ORDINI` (`idORDINI`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `mydb`.`PACCHI_has_Specie_di_Piante`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `mydb`.`PACCHI_has_Specie_di_Piante` (
  `PACCHI_idPACCHI` INT NOT NULL,
  `Pianta_Pacco` VARCHAR(6) NOT NULL,
  PRIMARY KEY (`PACCHI_idPACCHI`, `Pianta_Pacco`),
      INDEX  `fk_PACCHI_has_Specie_di_Piante_Specie_di_Piante1_idx`  (`Pianta_Pacco`  ASC)
VISIBLE,
      INDEX  `fk_PACCHI_has_Specie_di_Piante_PACCHI1_idx`  (`PACCHI_idPACCHI`  ASC)
VISIBLE,
  CONSTRAINT `fk_PACCHI_has_Specie_di_Piante_PACCHI1`
    FOREIGN KEY (`PACCHI_idPACCHI`)
    REFERENCES `mydb`.`PACCHI` (`idPACCHI`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_PACCHI_has_Specie_di_Piante_Specie_di_Piante1`
```

```
    FOREIGN KEY (`Pianta_Pacco`)
    REFERENCES `mydb`.`Specie_di_Piante` (`COD_Pianta`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `mydb`.`Referente`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `mydb`.`Referente` (
  `Nome_referente` VARCHAR(45) NOT NULL,
  `Cognome_referente` VARCHAR(45) NOT NULL,
  `Rivendita` VARCHAR(11) NOT NULL,
  INDEX `fk_Referente_Cliente1_idx` (`Rivendita` ASC) VISIBLE,
  PRIMARY KEY (`Rivendita`, `Cognome_referente`),
  CONSTRAINT `fk_Referente_Cliente1`
    FOREIGN KEY (`Rivendita`)
    REFERENCES `mydb`.`Cliente` (`COD_C`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `mydb`.`Contatti`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `mydb`.`Contatti` (
  `tipo` ENUM("email", "cel", "tel") NOT NULL,
  `Valore` VARCHAR(45) NOT NULL,
  `preferito` TINYINT NOT NULL,
  `Cliente_COD_C` VARCHAR(16) NOT NULL,
  INDEX `fk_Contatti_Cliente1_idx` (`Cliente_COD_C` ASC) VISIBLE,
  PRIMARY KEY (`Cliente_COD_C`, `Valore`),
```

```
  CONSTRAINT `fk_Contatti_Cliente1`
    FOREIGN KEY (`Cliente_COD_C`)
    REFERENCES `mydb`.`Cliente` (`COD_C`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;



-- -----------------------------------------------------
-- Table `mydb`.`utenti`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `mydb`.`utenti` (
  `username` VARCHAR(45) NOT NULL,
  `password` VARCHAR(32) NOT NULL,
  `ruolo` ENUM("operatore", "Manager") NOT NULL,
  PRIMARY KEY (`username`));

USE `mydb` ;

-- -----------------------------------------------------
-- procedure Visualizza_Pianta
-- -----------------------------------------------------

DELIMITER $$
USE `mydb`$$
CREATE DEFINER=`root`@`localhost` PROCEDURE `Visualizza_Pianta`(IN var_Pianta
VARCHAR(6))
BEGIN
declare exit handler for sqlexception
begin
        rollback;
   resignal;
end;
if((select count(*)
```

```
        from Specie_di_Piante
    where COD_Pianta = var_Pianta)>0)
    then
        select distinct Nom_latino, Nom_comune, Tipo, esotica, Giacenza
    from Specie_di_Piante, MAGAZZINO
    where var_Pianta = COD_Pianta and Pianta=var_Pianta;
else
signal sqlstate '45002' set message_text ="Pianta non trovata";
end if;

END$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure Visualizzare_Storico_Prezzo
-- -----------------------------------------------------

DELIMITER $$
USE `mydb`$$
CREATE PROCEDURE `Visualizzare_Storico_Prezzo`(IN var_Pianta VARCHAR(6))
BEGIN
declare exit handler for sqlexception
begin
        rollback;
    resignal;
end;
set transaction isolation level repeatable read;
start transaction;
if((select count(*)
        from Specie_di_Piante
    where var_Pianta = COD_Pianta)>0)
    then
        select *
```

```
   from LISTINO
   where Pianta_listino = var_Pianta;
else signal sqlstate '45002' set message_text="Pianta non trovata";
end if;
commit;
END$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure Visualizzare_Colorazioni
-- -----------------------------------------------------

DELIMITER $$
USE `mydb`$$
CREATE PROCEDURE `Visualizzare_Colorazioni`(IN var_Pianta VARCHAR(6))
BEGIN
declare exit handler for sqlexception
begin
        rollback;
   resignal;
end;
set transaction isolation level repeatable read;
start transaction;
if((select count(*)
        from Specie_di_Piante
   where var_Pianta = COD_Pianta)>0)
   then
        select distinct Nom_latino, Nom_comune, Tipo, esotica, Colore
   from Specie_di_Piante, Colore_has_Specie_di_Piante, Colore
   where var_Pianta = COD_Pianta and var_Pianta = ID_Fiorita and Colore_ID_colore = ID_colore;
else
signal sqlstate '45002' set message_text='Pianta non trovata';
end if;
```

```
commit;
END$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure Visualizza_ordine
-- -----------------------------------------------------

DELIMITER $$
USE `mydb`$$
CREATE PROCEDURE `Visualizza_ordine`(IN ordine int)
BEGIN
declare exit handler for sqlexception
begin
        rollback;
   resignal;
end;
set transaction isolation level repeatable read;
start transaction;
if((select count(*)
        from ORDINI
   where idORDINI = ordine)>0)
   then
        if((select  count(*)
                from ORDINI, ORDINI_has_Specie_di_Piante, Cliente, Contatti, Referente
                where ordine = ORDINI.idORDINI and Cliente.COD_C = ORDINI.Cliente and
Cliente.COD_C = Contatti.Cliente_COD_C and
ORDINI_has_Specie_di_Piante.ORDINI_idORDINI = ordine and Cliente.COD_C =
Referente.Rivendita) > 0)
                then
                select dataAcq, ORDINI_has_Specie_di_Piante.Specie, Quantita,Cliente.COD_C,
Cliente.Nome, Cliente.ind_sped, Contatti.tipo, Contatti.Valore, Referente.Nome_referente,
Referente.Cognome_referente
```

```
            from ORDINI, ORDINI_has_Specie_di_Piante, Cliente, Contatti, Referente
            where ordine = ORDINI.idORDINI and Cliente.COD_C = ORDINI.Cliente and
Cliente.COD_C = Contatti.Cliente_COD_C and
ORDINI_has_Specie_di_Piante.ORDINI_idORDINI = ordine and Cliente.COD_C =
Referente.Rivendita and Contatti.preferito =1;
   else
            select distinct dataAcq, ORDINI_has_Specie_di_Piante.Specie,
Quantita,Cliente.COD_C, Cliente.Nome, Cliente.ind_sped, Contatti.tipo, Contatti.Valore
            from ORDINI, ORDINI_has_Specie_di_Piante, Cliente, Contatti
            where (ordine = ORDINI.idORDINI and Cliente.COD_C = ORDINI.Cliente and
Cliente.COD_C = Contatti.Cliente_COD_C and
ORDINI_has_Specie_di_Piante.ORDINI_idORDINI = ordine and Contatti.preferito=1);
   end if;
else
signal sqlstate '45003' set message_text='Ordine non trovato';
end if;
commit;

END$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure Visualizza_ordini_cliente
-- -----------------------------------------------------

DELIMITER $$
USE `mydb`$$
CREATE PROCEDURE `Visualizza_ordini_cliente`(IN COD_CLI VARCHAR(16))
BEGIN
declare exit handler for sqlexception
begin
        rollback;
   resignal;
```

```
end;
set transaction isolation level repeatable read;
start transaction;
if((select count(*)
        from Cliente
   where COD_CLI = COD_C)>0)
   then
        select distinct idORDINI, dataAcq, ORDINI_has_Specie_di_Piante.Specie, Quantita
   from ORDINI, ORDINI_has_Specie_di_Piante
   where COD_CLI = Cliente and idORDINI = ORDINI_idORDINI;
else
signal sqlstate '45005' set message_text="Cliente non trovato";
end if;
   commit;
END$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure Visualizzare_piante_Pacco
-- -----------------------------------------------------

DELIMITER $$
USE `mydb`$$
CREATE PROCEDURE `Visualizzare_piante_Pacco`(in id_pacco int)
BEGIN
declare exit handler for sqlexception
begin
        rollback;
   resignal;
end;
set transaction isolation level repeatable read;
start transaction;
if((select count(*)
```

```
        from PACCHI
   where id_pacco= idPACCHI)>0)
   then
        select Pianta_Pacco, Nom_comune, Nom_latino, Ordine, Quantita
            from PACCHI_has_Specie_di_Piante, Specie_di_Piante, PACCHI, ORDINI,
ORDINI_has_Specie_di_Piante
          where PACCHI_idPACCHI = id_pacco and Pianta_Pacco = COD_Pianta and
id_pacco=idPACCHI and Ordine = idORDINI and idORDINI=ORDINI_idORDINI;
   else
   signal sqlstate '45006' set message_text="Pacco non trovato";
   end if;
   commit;

END$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure visualizzare_piante_inserire_pacco
-- -----------------------------------------------------

DELIMITER $$
USE `mydb`$$
CREATE PROCEDURE `visualizzare_piante_inserire_pacco`(in ordine int)
BEGIN
declare exit handler for sqlexception
begin
        rollback;
   resignal;
end;
set transaction isolation level repeatable read;
start transaction;
        if((select count(*)
                from ORDINI
```

```
	where ordine = idORDINI)>0)
	then
		select Specie
		from ORDINI_has_Specie_di_Piante
		where ORDINI_idORDINI = ordine and Specie NOT IN (select Pianta_Pacco as
Specie

	from PACCHI_has_Specie_di_Piante, PACCHI

	where PACCHI.ordine = ordine and idPACCHI = PACCHI_idPACCHI);
	else
		signal sqlstate '45003' set message_text='Ordine non trovato';
end if;
commit;
END$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure Aggiornare_Giacenze
-- -----------------------------------------------------

DELIMITER $$
USE `mydb`$$
CREATE PROCEDURE `Aggiornare_Giacenze`(in Specie_pianta varchar(6),in giac int)
BEGIN
declare exit handler for sqlexception
begin
	rollback;
  resignal;
end;
	set transaction isolation level repeatable read ;
  start transaction;
  if(Specie_pianta in (select COD_Pianta
```

```
                                        from Specie_di_Piante))
                then
                update MAGAZZINO
                SET Giacenza = Giacenza + giac
                where Pianta = Specie_pianta ;
        else
   signal sqlstate '45002' set message_text=' Pianta non trovata';
   end if;
   commit;
END$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure Modificare_prezzo_pianta
-- -----------------------------------------------------

DELIMITER $$
USE `mydb`$$
CREATE PROCEDURE `Modificare_prezzo_pianta`(in Specie_pianta varchar(6), in Nuovo_prezzo
double)
BEGIN
declare exit handler for sqlexception
begin
        rollback;
   resignal;
end;
set transaction isolation level serializable;
start transaction;
if((select count(*)
        from Specie_di_Piante
        where Specie_pianta = COD_Pianta)>0)
   then
   update LISTINO
```

```
   set Data_fine = NOW()
   where Pianta_listino = Specie_pianta and Data_fine is null;
       insert into LISTINO(Prezzo, Data_inizio, Pianta_listino) values (Nuovo_prezzo,NOW() ,Specie_pianta);
else
signal sqlstate '45002'set message_text="Pianta non trovata";
end if;
commit;
END$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure aggiungere_specie_pianta
-- -----------------------------------------------------

DELIMITER $$
USE `mydb`$$
CREATE  PROCEDURE `aggiungere_specie_pianta`(in codice varchar(6), in n_com varchar(45), in n_lat varchar(45), in posizione enum('interno', 'esterno'), in eso TINYINT, in quantita int, in Costo double, in cod_for int)
BEGIN
set transaction isolation level repeatable read;
start transaction;
        insert into Specie_di_Piante(COD_Pianta, Nom_latino, Nom_comune, Tipo, esotica) values (codice, n_lat, n_com, posizione, eso) ;
        insert into LISTINO(Prezzo, Data_inizio, Pianta_listino) values (Costo, NOW() ,codice);
        insert into MAGAZZINO(Giacenza, Pianta) values (quantita, codice);
   insert into Specie_di_Piante_has_Fornitori(Pianta_Fornita, Fornitore) values (codice, cod_for);
commit;
END$$

DELIMITER ;
```

```
-- -----------------------------------------------------
-- procedure Aggiungi_cliente_privato
-- -----------------------------------------------------

DELIMITER $$
USE `mydb`$$
CREATE PROCEDURE `Aggiungi_cliente_privato`(in cod varchar(16), in nom varchar(45), in fat varchar(45),in sped varchar(45), in res varchar(45),in tipo_contatto ENUM('email', 'cel', 'tel'), in valore_contatto varchar(45))
BEGIN
	insert into Cliente(COD_C, Nome, ind_fatt, ind_sped, ind_resid) values(cod, nom, fat, sped, res);
   insert into Contatti(tipo, Valore, preferito, Cliente_COD_C) values(tipo_contatto, valore_contatto, 1, cod);
END$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure Aggiungi_cliente_rivendita
-- -----------------------------------------------------

DELIMITER $$
USE `mydb`$$
CREATE PROCEDURE `Aggiungi_cliente_rivendita`(in cod varchar(11), in nom varchar(45), in fat varchar(45),in sped varchar(45), in res varchar(45),in tipo_contatto ENUM('email', 'cel', 'tel'), in valore_contatto varchar(45), in n_ref varchar(45), in c_ref varchar(45))
BEGIN
	insert into Cliente(COD_C, Nome, ind_fatt, ind_sped, ind_resid) values(cod, nom, fat, sped, res);
	  insert into Contatti(tipo, Valore, preferito, Cliente_COD_C) values(tipo_contatto, valore_contatto,1, cod);
   insert into Referente(Nome_referente, Cognome_referente, Rivendita) values (n_ref, c_ref, cod);
END$$
```

```
DELIMITER ;

-- -----------------------------------------------------
-- procedure Aggiungi_ordine
-- -----------------------------------------------------

DELIMITER $$
USE `mydb`$$
CREATE PROCEDURE `Aggiungi_ordine`(in id int, in cod_cliente varchar(16))
BEGIN
	insert into ORDINI(idORDINI, dataAcq, Cliente) values (id, NOW(), cod_cliente);
END$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure agg_piante_ordine
-- -----------------------------------------------------

DELIMITER $$
USE `mydb`$$
CREATE PROCEDURE `agg_piante_ordine`(in ordine int, in pianta VARCHAR(6), in quant int)
BEGIN
	insert into ORDINI_has_Specie_di_Piante(ORDINI_idORDINI, Specie, Quantita) values (ordine, pianta, quant);
END$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure aggiungi_contatto
-- -----------------------------------------------------
```

```
DELIMITER $$
USE `mydb`$$
CREATE PROCEDURE `aggiungi_contatto`(in cod_Cliente varchar(16), in tipo_contatto ENUM('email', 'cel', 'tel'), in valore_contatto varchar(45))
BEGIN
        insert into Contatti(tipo, Valore, preferito, Cliente_COD_C) values(tipo_contatto, valore_contatto, 0, cod_Cliente);
END$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure scegli_contatto_preferito
-- -----------------------------------------------------

DELIMITER $$
USE `mydb`$$
CREATE PROCEDURE `scegli_contatto_preferito`(in cliente varchar(16), in valore_contatto varchar(45))
BEGIN
declare exit handler for sqlexception
begin
        rollback;
   resignal;
end;
if(valore_contatto in (select Valore from Contatti where Cliente_COD_C = cliente))
then
        if ((select count(*)
                from Contatti
      where cliente = Cliente_COD_C and preferito = 1)>0)
      then update Contatti set preferito = 0 where cliente = Cliente_COD_C;
        end if;
        update Contatti set preferito = 1 where valore_contatto = Valore and cliente = Cliente_COD_C;
```

```
else
signal sqlstate '45004' set message_text='Contatto non trovato';

end if;

END$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure Aggiungi_piante_a_pacco
-- -----------------------------------------------------

DELIMITER $$
USE `mydb`$$
CREATE PROCEDURE `Aggiungi_piante_a_pacco`(in idpacco int,in pianta varchar(6))
BEGIN
declare exit handler for sqlexception
begin
        rollback;
   resignal;
end;
set transaction isolation level repeatable read;
start transaction;
if (pianta in (select Specie
                                        from ORDINI_has_Specie_di_Piante, ORDINI, PACCHI
                                        where  ORDINI_has_Specie_di_Piante.ORDINI_idORDINI  =
ORDINI.idORDINI and ORDINI.idORDINI = PACCHI.ordine and PACCHI.idPACCHI = idpacco
            and ORDINI_has_Specie_di_Piante.Specie NOT IN (select Pianta_Pacco as Specie

        from PACCHI_has_Specie_di_Piante

        where idpacco = PACCHI_idPACCHI)))
                                    THEN
```

```
        insert into PACCHI_has_Specie_di_Piante(PACCHI_idPACCHI, Pianta_Pacco) values (idpacco, pianta);
    else
   signal sqlstate '45001' set message_text=' Pianta non valida';
    END IF;
commit;
END$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure Aggiungi_colore_pianta
-- -----------------------------------------------------

DELIMITER $$
USE `mydb`$$
CREATE PROCEDURE `Aggiungi_colore_pianta`(in colorazione varchar(20), in pianta varchar(6) )
BEGIN

        if((select count(*)
                from Colore
      where Colore =colorazione) = 0)
                then insert into Colore (Colore) values (colorazione) ;
        end if;
     insert into Colore_has_Specie_di_Piante(id_Fiorita, Colore_ID_colore) values (pianta, (select ID_colore

                                                                from Colore
                                                        where Colore =colorazione));


END$$

DELIMITER ;
```

```
-- -----------------------------------------------------
-- procedure visualizza_fornitori_pianta
-- -----------------------------------------------------

DELIMITER $$
USE `mydb`$$
CREATE PROCEDURE `visualizza_fornitori_pianta`(in pianta varchar(6) )
BEGIN
declare exit handler for sqlexception
begin
        rollback;
   resignal;
end;
if((select count(*)
        from Specie_di_Piante
   where pianta = COD_Pianta)>0)
   then
        select idFornitori, cod_fis as 'codice fiscale', Nome_fornitore as 'Nome', Pianta_Fornita as 'Pianta', indirizzo
   from Fornitori, Specie_di_Piante_has_Fornitori, INDIRIZZI
                where pianta = Specie_di_Piante_has_Fornitori.Pianta_Fornita and Specie_di_Piante_has_Fornitori.Fornitore = Fornitori.idFornitori and Fornitori.idFornitori = INDIRIZZI.Fornitori_idFornitori;
else
 signal sqlstate '45002'set message_text="Pianta non trovata";
 end if;
END$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure login
-- -----------------------------------------------------
```

```
DELIMITER $$
USE `mydb`$$
CREATE PROCEDURE `login`(in var_username varchar(45), in var_pass varchar(45), out var_role
INT)
BEGIN
	declare var_user_role ENUM('operatore', 'Manager');

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

	-- See the corresponding enum in the client
		if var_user_role = 'operatore' then
			set var_role = 1;
		elseif var_user_role = 'Manager' then
			set var_role = 2;
		else
			set var_role = 3;
		end if;
END$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure Aggiungi_pacco
-- -----------------------------------------------------

DELIMITER $$
USE `mydb`$$
CREATE PROCEDURE `Aggiungi_pacco`(in ordine int, in pacco int)
BEGIN
	insert into PACCHI(idPACCHI, Ordine) values (pacco, ordine);
```

```
END$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure visualizza_totale_ordine
-- -----------------------------------------------------

DELIMITER $$
USE `mydb`$$
CREATE PROCEDURE `visualizza_totale_ordine`(in ordine int)
BEGIN
declare exit handler for sqlexception
begin
        rollback;
   resignal;
end;
   set transaction isolation level read committed;
   start transaction;
   if((select count(*)
                from ORDINI
      where idORDINI = ordine)>0)
      then
                select sum(Prezzo * quantita) as 'Totale'
                from LISTINO, Specie_di_Piante, ORDINI_has_Specie_di_Piante, ORDINI
                where ordine = ORDINI_idORDINI and ordine=idORDINI and
dataAcq>=Data_inizio  and (dataAcq<=Data_fine or Data_fine is null) and Pianta_listino =
COD_Pianta and COD_Pianta = Specie;
        else
                signal sqlstate '45003' set message_text= 'Ordine non trovato';
   end if;
   commit;
END$$
```

```
DELIMITER ;

-- -----------------------------------------------------
-- procedure crea_utente
-- -----------------------------------------------------

DELIMITER $$
USE `mydb`$$
CREATE PROCEDURE `crea_utente`(IN username VARCHAR(45), IN pass VARCHAR(32), IN ruolo enum('Manager','operatore'))
BEGIN
	insert into utenti(username,password, ruolo) VALUES(username, MD5(pass), ruolo);
END$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure aggiungi_fornitore_specie
-- -----------------------------------------------------

DELIMITER $$
USE `mydb`$$
CREATE PROCEDURE `aggiungi_fornitore_specie`(in cod int, in pianta varchar(6))
BEGIN

insert into Specie_di_Piante_has_Fornitori(Pianta_Fornita, Fornitore) values(pianta, cod);
END$$

DELIMITER ;
USE `mydb`;

DELIMITER $$
USE `mydb`$$
```

```
CREATE DEFINER = CURRENT_USER TRIGGER `mydb`.`ORDINI_has_Specie_di_Piante_BEFORE_INSERT` BEFORE INSERT ON `ORDINI_has_Specie_di_Piante` FOR EACH ROW
BEGIN

	if((NEW.Quantita)>(select Giacenza
					from mydb.MAGAZZINO, Specie_di_Piante
					where mydb.MAGAZZINO.Pianta = Specie_di_Piante.COD_Pianta and NEW.Specie = Specie_di_Piante.COD_Pianta))
		then
					signal sqlstate '45000' set message_text=' Non ci sono abbastanza Piante in magazzino';
	else update MAGAZZINO
		set Giacenza = Giacenza - NEW.Quantita
	where MAGAZZINO.Pianta = NEW.Specie;
	end if;

END$$


DELIMITER ;
CREATE USER 'operatore' IDENTIFIED BY 'operatore';

GRANT EXECUTE ON procedure `mydb`.`agg_piante_ordine` TO 'operatore';
GRANT EXECUTE ON procedure `mydb`.`Aggiungi_cliente_privato` TO 'operatore';
GRANT EXECUTE ON procedure `mydb`.`Aggiungi_cliente_rivendita` TO 'operatore';
GRANT EXECUTE ON procedure `mydb`.`aggiungi_contatto` TO 'operatore';
GRANT EXECUTE ON procedure `mydb`.`Aggiungi_ordine` TO 'operatore';
GRANT EXECUTE ON procedure `mydb`.`Aggiungi_piante_a_pacco` TO 'operatore';
GRANT EXECUTE ON procedure `mydb`.`scegli_contatto_preferito` TO 'operatore';
GRANT EXECUTE ON procedure `mydb`.`Visualizza_ordine` TO 'operatore';
GRANT EXECUTE ON procedure `mydb`.`Visualizza_ordini_cliente` TO 'operatore';
GRANT EXECUTE ON procedure `mydb`.`visualizzare_piante_inserire_pacco` TO 'operatore';
GRANT EXECUTE ON procedure `mydb`.`Visualizzare_piante_Pacco` TO 'operatore';
```

```
GRANT EXECUTE ON procedure `mydb`.`Visualizzare_Storico_Prezzo` TO 'operatore';
GRANT EXECUTE ON procedure `mydb`.`Aggiungi_pacco` TO 'operatore';
GRANT EXECUTE ON procedure `mydb`.`Visualizza_Pianta` TO 'operatore';
GRANT EXECUTE ON procedure `mydb`.`visualizza_totale_ordine` TO 'operatore';
GRANT EXECUTE ON procedure `mydb`.`Visualizzare_Colorazioni` TO 'operatore';
CREATE USER 'Manager' IDENTIFIED BY 'Manager';

GRANT EXECUTE ON procedure `mydb`.`aggiungere_specie_pianta` TO 'Manager';
GRANT EXECUTE ON procedure `mydb`.`Aggiungi_colore_pianta` TO 'Manager';
GRANT EXECUTE ON procedure `mydb`.`Modificare_prezzo_pianta` TO 'Manager';
GRANT EXECUTE ON procedure `mydb`.`Visualizzare_Storico_Prezzo` TO 'Manager';
GRANT EXECUTE ON procedure `mydb`.`visualizza_fornitori_pianta` TO 'Manager';
GRANT EXECUTE ON procedure `mydb`.`crea_utente` TO 'Manager';
GRANT EXECUTE ON procedure `mydb`.`Visualizza_Pianta` TO 'Manager';
GRANT EXECUTE ON procedure `mydb`.`Visualizzare_Colorazioni` TO 'Manager';
GRANT EXECUTE ON procedure `mydb`.`Aggiornare_Giacenze` TO 'Manager';
GRANT EXECUTE ON procedure `mydb`.`visualizza_totale_ordine` TO 'Manager';
CREATE USER 'login' IDENTIFIED BY 'login';

GRANT EXECUTE ON procedure `mydb`.`login` TO 'login';

SET SQL_MODE=@OLD_SQL_MODE;
SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS;
SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS;


call mydb.crea_utente('admin', 'admin', 'Manager');
INSERT INTO `mydb`.`Fornitori` (`idFornitori`, `cod_fis`, `Nome_fornitore`) VALUES ('1',
'aaabbb12c34d567e', 'Mario Rossi');
INSERT INTO `mydb`.`Fornitori` (`idFornitori`, `cod_fis`, `Nome_fornitore`) VALUES ('2',
'xxxyyy12z34t567e', 'Carlo Flora');
INSERT INTO `mydb`.`Fornitori` (`idFornitori`, `cod_fis`, `Nome_fornitore`) VALUES ('3',
'01786610894', 'Piante co.');
```

```
INSERT INTO `mydb`.`Fornitori` (`idFornitori`, `cod_fis`, `Nome_fornitore`) VALUES ('4', '12345678910', 'EXPOLANT');
INSERT INTO `mydb`.`Fornitori` (`idFornitori`, `cod_fis`, `Nome_fornitore`) VALUES ('5', '10987654321', 'naturarbor');
INSERT INTO `mydb`.`INDIRIZZI` (`indirizzo`, `Fornitori_idFornitori`) VALUES ('Via del corso, 1', '1');
INSERT INTO `mydb`.`INDIRIZZI` (`indirizzo`, `Fornitori_idFornitori`) VALUES ('Via Brombeis, 8', '2');
INSERT INTO `mydb`.`INDIRIZZI` (`indirizzo`, `Fornitori_idFornitori`) VALUES ('Via dei Tribunali, 5', '2');
INSERT INTO `mydb`.`INDIRIZZI` (`indirizzo`, `Fornitori_idFornitori`) VALUES ('Via Po, 101', '3');
INSERT INTO `mydb`.`INDIRIZZI` (`indirizzo`, `Fornitori_idFornitori`) VALUES ('Viale Crispi, 74', '3');
INSERT INTO `mydb`.`INDIRIZZI` (`indirizzo`, `Fornitori_idFornitori`) VALUES ('Via Tuscolana, 404', '4');
INSERT INTO `mydb`.`INDIRIZZI` (`indirizzo`, `Fornitori_idFornitori`) VALUES ('Via Casilina, 102', '4');
INSERT INTO `mydb`.`INDIRIZZI` (`indirizzo`, `Fornitori_idFornitori`) VALUES ('Via Cassia, 104', '4');
INSERT INTO `mydb`.`INDIRIZZI` (`indirizzo`, `Fornitori_idFornitori`) VALUES ('Via Prenestina, 500', '4');
INSERT INTO `mydb`.`INDIRIZZI` (`indirizzo`, `Fornitori_idFornitori`) VALUES ('Viale Bovio, 15', '5');
INSERT INTO `mydb`.`INDIRIZZI` (`indirizzo`, `Fornitori_idFornitori`) VALUES ('Via Cona, 19', '5');
```

Codice del Front-End

----------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 void dump_result_set(MYSQL *conn, MYSQL_STMT *stmt, char *title);
extern void run_as_manager(MYSQL *conn);
extern void run_as_operatore(MYSQL *conn);
```

----------Utils.c----------

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include "defines.h"

void flush_stdin(){
	int c;
	while ((c = getchar()) != '\n' && c != EOF) { }
}

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->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->length)
                  col_len = field->length;
          if (col_len < 4 && !IS_NOT_NULL(field->flags))
                  col_len = 4; /* 4 = length of the word "NULL" */
          field->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->length, field->name);


        }
        putchar('\n');

        print_dashes(res_set);
}

void 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;

        /* Prefetch the whole result set. This in conjunction with
         * STMT_ATTR_UPDATE_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);

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(long long int);
                        break;
                default:
                        attr_size = fields[i].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].buffer_type == MYSQL_TYPE_NULL) {
                        printf (" %-*s |", (int)fields[i].length, "NULL");

                        continue;
                }

                switch (rs_bind[i].buffer_type) {

                        case MYSQL_TYPE_VAR_STRING:

                                printf(" %-*s |",  (int)fields[i].length, (char*)rs_bind[i].buffer);
                                break;

                   case MYSQL_TYPE_DATETIME:

                   case MYSQL_TYPE_TIMESTAMP:
        date = (MYSQL_TIME *)rs_bind[i].buffer;
```

```
                printf("%02d/%02d/%4d %02d:%02d:%02d  |", date->day, date->month, date-
>year,date->hour,date->minute,date->second);
                break;
                        case MYSQL_TYPE_TIME:
                                date = (MYSQL_TIME *)rs_bind[i].buffer;
                                printf(" %02d-%02d-%02d |  ", date->hour,date->minute,date-
>second);
                                break;
                        case MYSQL_TYPE_DATE:
                                date = (MYSQL_TIME *)rs_bind[i].buffer;
                                printf(" %4d-%02d-%02d   |", date->year, date->month, date-
>day);
                                break;

                        case MYSQL_TYPE_STRING:
                                printf(" %-*s |", (int)fields[i].length, (char *)rs_bind[i].buffer);
                                break;

                        case MYSQL_TYPE_FLOAT:
                        case MYSQL_TYPE_DOUBLE:
                                printf(" %-*.02f |", (int)fields[i].length,*(double
*)rs_bind[i].buffer);
                                break;

                        case MYSQL_TYPE_LONG:
                printf(" %-*d |", (int)fields[i].length, *(int *)rs_bind[i].buffer);
                        break;
                        case MYSQL_TYPE_SHORT:
                        case MYSQL_TYPE_TINY:
                if(*(signed char*)rs_bind[i].buffer == 0){
                   printf("no      |");
                }
                else{printf("si      |");}
                        //      printf(" %-*u    |", (int)fields[i].max_length, *(signed char
*)rs_bind[i].buffer);
                                break;
                    case MYSQL_TYPE_LONGLONG:
                        printf(" %-*lld |", (int)fields[i].length, *(long long int
*)rs_bind[i].buffer);
                        break;
                        case MYSQL_TYPE_NEWDECIMAL:
                                printf(" %-*.02lf |", (int)fields[i].length, *(float*)
rs_bind[i].buffer);
                                break;
            case MYSQL_TYPE_NULL:
                printf("null");
                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);
      }
}
```

----------main.c----------

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


typedef enum {
   operatore = 1,
   Manager = 2,
   FAILED_LOGIN = 3
} 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;
    char username[45];
    char pswd[45];


    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, "localhost", "root", "root1234", "mydb", 3306, NULL,
CLIENT_MULTI_STATEMENTS | CLIENT_MULTI_RESULTS) == NULL) {

        fprintf (stderr, "mysql_real_connect() failed\n");
        mysql_close (conn);
        exit(EXIT_FAILURE);
    }

   printf("--------Benvenuto nel database di Verde S.r.l--------\n \t --------Inserisci le tue
credenziali--------\n");
   printf("Username: ");
   getInput(45, username, false);


   printf("Password: ");
   getInput(45, pswd, true);

   role = attempt_login(conn, username, pswd);


   switch(role) {
      case operatore:
         run_as_operatore(conn);
         break;

      case Manager:
         run_as_manager(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("---------------------------- CIAO ------------------------------------\n");


   mysql_close (conn);



}
```

----------Manager.c----------

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

#include "defines.h"

void clear_stdin(){
   int a;
   while((a = getchar()) != '\n' && a != EOF){}
}

static void Aggiungi_Specie_di_Pianta(MYSQL *conn){
        MYSQL_STMT *prepared_stmt;
        MYSQL_BIND param[8];

        char Pianta[6];
        char Nome_comune[45];
        char Nome_latino[45];
        char posizione[8];
        char options[4] ={'1','2'};
        char p;
        signed char esotica;
        int quantita;
        double Costo;
        char Fornitore[16];


        printf("\nCodice Pianta: ");
        getInput(6, Pianta, false);
        printf("Nome Comune: ");
        getInput(45, Nome_comune, false);
        printf("Nome Latino: ");
        getInput(45, Nome_latino, false);
        printf("Posizione Pianta\n");
        printf("\t1) interno\n");
        printf("\t2) esterno\n");
        p = multiChoice("Seleziona Posizione", options, 2);
        switch(p) {
          case '1':
                strcpy(posizione, "interno");
                break;
          case '2':
                strcpy(posizione, "esterno");
                break;

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

```
        }

        printf("La Pianta è esotica?\n");
        p = multiChoice("1)No \t2)Si", options, 2);
        switch(p) {
          case '1':
                    esotica=0;
                    break;
          case '2':
                    esotica=1;
                    break;

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

        printf("Quantità disponibile: ");
        scanf("%d", &quantita);
    clear_stdin();
        printf("Costo Pianta: ");
        scanf("%lf", &Costo);
    clear_stdin();
        printf("Indicare il codice del Fornitore: ");
        getInput(17, Fornitore, false);


        if(!setup_prepared_stmt(&prepared_stmt, "call aggiungere_specie_pianta(?, ?, ?, ?, ?, ?, ?, ?)",
conn)) {
          finish_with_stmt_error(conn, prepared_stmt, "Impossibile inzializzare lo statement\n", false);
        }


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

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

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

        param[2].buffer_type = MYSQL_TYPE_VAR_STRING;  //IN
        param[2].buffer = Nome_latino;
        param[2].buffer_length = strlen(Nome_latino);

        param[3].buffer_type =MYSQL_TYPE_VAR_STRING ;  //IN
        param[3].buffer = posizione;
```

```
        param[3].buffer_length = strlen(posizione);

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

        param[5].buffer_type = MYSQL_TYPE_LONG;  //IN
        param[5].buffer = &quantita;
        param[5].buffer_length = sizeof(quantita);

        param[6].buffer_type = MYSQL_TYPE_DOUBLE;  //IN
        param[6].buffer = &Costo;
        param[6].buffer_length = sizeof(Costo);

        param[7].buffer_type = MYSQL_TYPE_VAR_STRING;  //IN
        param[7].buffer = Fornitore;
        param[7].buffer_length = strlen(Fornitore);


        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, "errore durante l'inseriemento");
        } else {
          printf("Pianta inserita correttamente\n");
        }

        mysql_stmt_close(prepared_stmt);

}

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

        char colorazione[20];
        char pianta[7];

        printf("\nInserisci Pianta: ");
        getInput(6, pianta, false);
        printf("Inserisci colorazione: ");
        getInput(20, colorazione, false);

        if(!setup_prepared_stmt(&prepared_stmt, "call Aggiungi_colore_pianta(?, ?)", conn)) {
          finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize student insertion
statement\n", false);
```

```
        }

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

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

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



        if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
          finish_with_stmt_error(conn, prepared_stmt, "Impossibile associare i parametri.\n", true);
        }

        // Run procedure
        if (mysql_stmt_execute(prepared_stmt) != 0) {
          print_stmt_error (prepared_stmt, "Errore durante l'inserimento colorazione.");
        } else {
          printf("Colorazione aggiunta con successo.\n");
        }

        mysql_stmt_close(prepared_stmt);
}

static void Modifica_Prezzo (MYSQL *conn){

        MYSQL_STMT *prepared_stmt;
        MYSQL_BIND param[2];

        char pianta[6];
        double Prezzo;

        printf("\nInserisci Pianta: ");
        getInput(6, pianta, false);
        printf("Inserisci Prezzo nuovo: ");
        scanf("%lf", &Prezzo);
    clear_stdin();

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

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

        param[0].buffer_type = MYSQL_TYPE_VAR_STRING;  //IN
```

```
        param[0].buffer = pianta;
        param[0].buffer_length = strlen(pianta);

        param[1].buffer_type = MYSQL_TYPE_DOUBLE;  //IN
        param[1].buffer = &Prezzo;
        param[1].buffer_length = sizeof(Prezzo);

        if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
          finish_with_stmt_error(conn, prepared_stmt, "Impossibile associare i parametri.\n", true);
        }

        // Run procedure
        if (mysql_stmt_execute(prepared_stmt) != 0) {
          print_stmt_error (prepared_stmt, "Errore durante l'aggiornamento del prezzo.");
        } else {
          printf("Prezzo modificato con successo.\n");
        }

        mysql_stmt_close(prepared_stmt);


}

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

        char username[45];
        char password[32];
   char tipo[10];
   char options[4] ={'1','2'};
        char p;

        printf("\nInserisci Username: ");
        getInput(45, username, false);
        printf("Inserisci Password: ");
        getInput(32, password, false);
   printf("Seleziona Ruolo: \n");
   printf("\t1) Manager\n");
        printf("\t2) Operatore\n");
        p = multiChoice("Seleziona Ruolo:", options, 2);
        switch(p) {
          case '1':
                    strcpy(tipo, "Manager");
                    break;
          case '2':
                    strcpy(tipo, "operatore");
                    break;
       default:
                    fprintf(stderr, "Invalid condition at %s:%d\n", __FILE__, __LINE__);
```

```
                abort();
        }

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

        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_VAR_STRING;  //IN
        param[2].buffer =tipo;
        param[2].buffer_length = strlen(tipo);

        if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
          finish_with_stmt_error(conn, prepared_stmt, "Impossibile associare i parametri.\n", true);
        }

        // Run procedure
        if (mysql_stmt_execute(prepared_stmt) != 0) {
          print_stmt_error (prepared_stmt, "Errore durante la creazione dell'utente.");
        } else {
          printf("Utente creato con successo.\n");
        }

        mysql_stmt_close(prepared_stmt);

}

static void V_costo_ordine(MYSQL *conn){

    MYSQL_STMT *prepared_stmt;
        MYSQL_BIND param[1];

        int ordine;

        printf("Inserisci num. Ordine: ");
        scanf("%d", &ordine);

        if(!setup_prepared_stmt(&prepared_stmt, "call visualizza_totale_ordine(?)", conn)) {
          finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize student insertion
statement\n", false);
```

```
        }

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

        param[0].buffer_type = MYSQL_TYPE_LONG;  //IN
        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, "Impossibile associare i parametri.\n", true);
        }

        // Run procedure
        if (mysql_stmt_execute(prepared_stmt) != 0) {
          print_stmt_error (prepared_stmt, "Errore durante la visualizzazione delle costo totale
dell'ordine.");
        } else {
          dump_result_set(conn, prepared_stmt, "-------------Costo Totale Ordine-------------");
        }
    mysql_stmt_next_result(prepared_stmt);
        mysql_stmt_close(prepared_stmt);

}

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


        char pianta[6];


        printf("\nInserisci Pianta:");
        getInput(6, pianta, false);


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

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

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

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

```
        finish_with_stmt_error(conn, prepared_stmt, "Impossibile associare i parametri.\n", true);
      }

      // Run procedure
      if (mysql_stmt_execute(prepared_stmt) != 0) {
        print_stmt_error (prepared_stmt, "Errore durante la visualizzazione dello storico dei prezzi
della pianta.");
      } else{
     dump_result_set(conn, prepared_stmt, "-------------Storico Prezzo Pianta-------------");
  }

  mysql_stmt_next_result(prepared_stmt);
      mysql_stmt_close(prepared_stmt);

}

static void V_colorazione(MYSQL *conn){

      MYSQL_STMT *prepared_stmt;
      MYSQL_BIND param[1];

      char pianta[6];


      printf("\nInserisci Pianta: ");
      getInput(6, pianta, false);


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

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

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

      if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
        finish_with_stmt_error(conn, prepared_stmt, "Impossibile associare i parametri.\n", true);
      }

      // Run procedure
      if (mysql_stmt_execute(prepared_stmt) != 0) {
        print_stmt_error (prepared_stmt, "Errore durante la visualizzazione dei colori.");
      } else {

     dump_result_set(conn, prepared_stmt, "-------------Visualizza Colorazioni della
pianta-------------");
```

```
    }
   mysql_stmt_next_result(prepared_stmt);
        mysql_stmt_close(prepared_stmt);

}

static void V_pianta(MYSQL *conn){


        MYSQL_STMT *prepared_stmt;
        MYSQL_BIND param[1];

        char pianta[6];


        printf("\nInserisci Pianta: ");
        getInput(6, pianta, false);


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

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

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

        if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
          finish_with_stmt_error(conn, prepared_stmt, "Impossibile associare i parametri.\n", true);
        }

        // Run procedure
        if (mysql_stmt_execute(prepared_stmt) != 0) {
          print_stmt_error (prepared_stmt, "Errore durante la visualizzazione delle informazioni della
pianta.");
        } else {
       dump_result_set(conn, prepared_stmt, "-------------Visualizza Pianta-------------");
    }
   mysql_stmt_next_result(prepared_stmt);
        mysql_stmt_close(prepared_stmt);


}

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

```
        char pianta[6];


        printf("\nInserisci Pianta: ");
        getInput(6, pianta, false);


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

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

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

        if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
          finish_with_stmt_error(conn, prepared_stmt, "Impossibile associare i parametri.\n", true);
        }

        // Run procedure
        if (mysql_stmt_execute(prepared_stmt) != 0) {
          print_stmt_error (prepared_stmt, "Errore durante la visualizzazione dei Fornitori.");
        } else {
          dump_result_set(conn, prepared_stmt, "-------------Visualizza Fornitori Pianta-------------");
        }
    mysql_stmt_next_result(prepared_stmt);
        mysql_stmt_close(prepared_stmt);

}

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

        char pianta[6];
        int giacenza;

        printf("\nInserisci Pianta: ");
        getInput(6, pianta, false);
        printf("Inserisci la quantità da aggiungere: ");
        scanf("%d", &giacenza);

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

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

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

        param[1].buffer_type = MYSQL_TYPE_LONG;  //IN
        param[1].buffer = &giacenza;
        param[1].buffer_length = sizeof(giacenza);

        if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
          finish_with_stmt_error(conn, prepared_stmt, "Impossibile associare i parametri.\n", true);
        }

        // Run procedure
        if (mysql_stmt_execute(prepared_stmt) != 0) {
          print_stmt_error (prepared_stmt, "Errore durante l'aggiornamento della Giacenza.");
        } else {
          printf("-------------Giacenza aggiornata-------------.\n");
        }

        mysql_stmt_close(prepared_stmt);
}

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

        char pianta[6];
        int Fornitore;

        printf("\nInserisci Pianta: ");
        getInput(6, pianta, false);
        printf("Inserisci codice Fornitore: ");
   scanf("%d", &Fornitore);
   clear_stdin();

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

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

        param[0].buffer_type = MYSQL_TYPE_LONG;  //IN
        param[0].buffer = &Fornitore;
        param[0].buffer_length = sizeof(Fornitore);

        param[1].buffer_type = MYSQL_TYPE_VAR_STRING;  //IN
```

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

        if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
          finish_with_stmt_error(conn, prepared_stmt, "Impossibile associare i parametri.\n", true);
        }

        // Run procedure
        if (mysql_stmt_execute(prepared_stmt) != 0) {
          print_stmt_error (prepared_stmt, "Errore durante l'aggiunta del fornitore.");
        } else {
          printf("-------------Fornitore aggiunto alla specie-------------.\n");
        }

        mysql_stmt_close(prepared_stmt);
}

void run_as_manager(MYSQL *conn){
        char options[12] = {'1','2','3','4','5','6','7','8','9','a','b','e'};
        char op;

        printf("------------>Passo al ruolo di Manager...\n");

        if(!parse_config("users/Manager.json", &conf)) {
          fprintf(stderr, "Impossibile caricare info Manager\n");
          exit(EXIT_FAILURE);
        }

        if(mysql_change_user(conn, "root", "root1234", "mydb")) {
          fprintf(stderr, "mysql_change_user() failed\n");
          exit(EXIT_FAILURE);
        }

        while(true) {

          printf("----- Cosa desidera fare ? -----\n\n");
          printf("1) Aggiungi Specie di Pianta\n");
          printf("2) Aggiungi Colore a Pianta\n");
          printf("3) Modifica Prezzo di una Pianta\n");
        printf("4) Aggiungi un nuovo utente\n");
        printf("5) Aggiorna Giacenze Magazzino\n");
          printf("6) Visualizza storico dei prezzi\n");
          printf("7) Visualizza fornitori di una specie\n");
        printf("8) Visualizza Specie\n");
        printf("9) Visualizza Colorazioni di una Specie\n");
        printf("a) Visualizza Totale dell'ordine\n");
        printf("b) Aggiungi Fornitore a Pianta\n");
          printf("e) Esci\n\n");

          op = multiChoice("Seleziona un opzione", options, 12);
```

```
        switch(op) {

                case '1':
                        Aggiungi_Specie_di_Pianta(conn);
                        break;
                case '2':
                        Aggiungi_Colore(conn);
                        break;
                case '3':
                        Modifica_Prezzo(conn);
                        break;
        case '4':
           crea_utente(conn);
           break;
                case '5':
           Aggiorna_Giacenza(conn);
                        break;
        case '6':
                        V_prezzi(conn);
                        break;
                case '7':
                        V_Fornitori(conn);
                        break;
                case '8':
                        V_pianta(conn);
                        break;
        case '9':
           V_colorazione(conn);
           break;
                case 'a':
                        V_costo_ordine(conn);
           break;
        case 'b':
           Aggiungi_fornitore_specie(conn);
           break;
        case'e':
           return;

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

      }
}
```

----------operatore.c----------

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

#include "defines.h"

void clear(){
   int a;
   while((a = getchar()) != '\n' && a != EOF){}
}

static void Aggiungi_pianta(MYSQL *conn, int ordine);
//1
static void Add_cPrivato(MYSQL *conn){
   MYSQL_STMT *prepared_stmt;
        MYSQL_BIND param[7];

        char codice[16];
   char nome[45];
   char fatturazione[45];
        char spedizione[45];
   char residenza[45];
   char tipo[6];
   char options[4] ={'1','2', '3'};
        char p;
   char valore[45];

        printf("\nInserisci Codice Fiscale Cliente: ");
        getInput(16, codice, false);
        printf("\nInserisci Nome Cliente: ");
        getInput(45, nome, false);
   printf("\nInserisci indirizzo fatturazione: ");
        getInput(45, fatturazione, false);
        printf("\nInserisci indirizzo spedizione: ");
        getInput(45, spedizione, false);
   printf("\nInserisci indirizzo residenza: ");
        getInput(45, residenza, false);
   printf("Seleziona Contatto\n");
        printf("\t1) email\n");
        printf("\t2) telefono\n");
   printf("\t3) cellulare\n");
        p = multiChoice("Seleziona Posizione", options, 3);
        switch(p) {
          case '1':
                    strcpy(tipo, "email");
                    break;
          case '2':
                    strcpy(tipo, "tel");
```

```
                break;
        case '3':
            strcpy(tipo, "cel");
            break;
        default:
                fprintf(stderr, "Invalid condition at %s:%d\n", __FILE__, __LINE__);
                abort();
        }

        printf("\nInserisci Contatto: ");
        getInput(45, valore, false);


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

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

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

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

        param[2].buffer_type = MYSQL_TYPE_VAR_STRING;  //IN
        param[2].buffer = fatturazione;
        param[2].buffer_length = strlen(fatturazione);

        param[3].buffer_type =MYSQL_TYPE_VAR_STRING ;  //IN
        param[3].buffer = spedizione;
        param[3].buffer_length = strlen(spedizione);

        param[4].buffer_type = MYSQL_TYPE_VAR_STRING;  //IN
        param[4].buffer = residenza;
        param[4].buffer_length = strlen(residenza);

        param[5].buffer_type = MYSQL_TYPE_VAR_STRING;  //IN
        param[5].buffer = tipo;
        param[5].buffer_length = strlen(tipo);

        param[6].buffer_type = MYSQL_TYPE_VAR_STRING;  //IN
        param[6].buffer = valore;
        param[6].buffer_length = strlen(valore);
        if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
            finish_with_stmt_error(conn, prepared_stmt, "Impossibile associare i parametri.\n", true);
```

```
        }

        // Run procedure
        if (mysql_stmt_execute(prepared_stmt) != 0) {
          print_stmt_error (prepared_stmt, "Errore durante l'inserimento.");
        } else {
          printf("-------------Cliente aggiunto con successo-------------\n");
        }

        mysql_stmt_close(prepared_stmt);

}
//2
static void Add_cRivendita(MYSQL *conn){

 MYSQL_STMT *prepared_stmt;
        MYSQL_BIND param[9];

        char codice[16];
    char nome[45];
    char fatturazione[45];
        char spedizione[45];
    char residenza[45];
    char tipo[6];
    char options[4] ={'1','2', '3'};
        char p;
    char valore[45];
    char nome_referente[45];
    char cognome_referente[45];

        printf("\nInserisci Partita IVA Cliente: ");
        getInput(16, codice, false);
        printf("\nInserisci Nome Rivendita: ");
        getInput(45, nome, false);
    printf("\nInserisci indirizzo fatturazione: ");
        getInput(45, fatturazione, false);
        printf("\nInserisci indirizzo spedizione: ");
        getInput(45, spedizione, false);
    printf("\nInserisci indirizzo residenza: ");
        getInput(45, residenza, false);
    printf("Seleziona Contatto\n");
        printf("\t1) email\n");
        printf("\t2) telefono\n");
    printf("\t3) cellulare\n");
        p = multiChoice("Seleziona Posizione", options, 3);
        switch(p) {
          case '1':
                    strcpy(tipo, "email");
                    break;
          case '2':
```

```
                strcpy(tipo, "tel");
                break;
        case '3':
          strcpy(tipo, "cel");
          break;
        default:
                fprintf(stderr, "Invalid condition at %s:%d\n", __FILE__, __LINE__);
                abort();
        }

        printf("\nInserisci Contatto: ");
        getInput(45, valore, false);
    printf("\nInserisci Nome Referente della Rivendita: ");
        getInput(45, nome_referente, false);
    printf("\nInserisci Cognome Referente della Rivendita: ");
        getInput(45, cognome_referente, false);


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

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

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

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

        param[2].buffer_type = MYSQL_TYPE_VAR_STRING;  //IN
        param[2].buffer = fatturazione;
        param[2].buffer_length = strlen(fatturazione);

        param[3].buffer_type =MYSQL_TYPE_VAR_STRING ;  //IN
        param[3].buffer = spedizione;
        param[3].buffer_length = strlen(spedizione);

        param[4].buffer_type = MYSQL_TYPE_VAR_STRING;  //IN
        param[4].buffer = residenza;
        param[4].buffer_length = strlen(residenza);

        param[5].buffer_type = MYSQL_TYPE_VAR_STRING;  //IN
        param[5].buffer = tipo;
        param[5].buffer_length = strlen(tipo);
```

```
        param[6].buffer_type = MYSQL_TYPE_VAR_STRING;  //IN
        param[6].buffer = valore;
        param[6].buffer_length = strlen(valore);

    param[7].buffer_type = MYSQL_TYPE_VAR_STRING;  //IN
        param[7].buffer = nome_referente;
        param[7].buffer_length = strlen(nome_referente);

        param[8].buffer_type = MYSQL_TYPE_VAR_STRING;  //IN
        param[8].buffer = cognome_referente;
        param[8].buffer_length = strlen(cognome_referente);

        if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
          finish_with_stmt_error(conn, prepared_stmt, "Impossibile associare i parametri.\n", true);
        }

        // Run procedure
        if (mysql_stmt_execute(prepared_stmt) != 0) {
          print_stmt_error (prepared_stmt, "Errore durante l'inserimento.");
        } else {
          printf("-------------Cliente aggiunto con successo-------------\n");
        }

        mysql_stmt_close(prepared_stmt);

}
//3
static void Add_contatti(MYSQL *conn){
    MYSQL_STMT *prepared_stmt;
        MYSQL_BIND param[3];

    char cliente[45];
    char tipo[6];
    char options[4] ={'1','2', '3'};
        char p;
    char valore[45];

    printf("\nInserisci Codice del Cliente: ");
        getInput(45, cliente, false);
    printf("Seleziona Contatto\n");
        printf("\t1) email\n");
        printf("\t2) telefono\n");
    printf("\t3) cellulare\n");
        p = multiChoice("Seleziona Posizione", options, 3);
        switch(p) {
          case '1':
                    strcpy(tipo, "email");
                    break;
          case '2':
                    strcpy(tipo, "tel");
```

```
                break;
        case '3':
          strcpy(tipo, "cel");
          break;
        default:
                fprintf(stderr, "Invalid condition at %s:%d\n", __FILE__, __LINE__);
                abort();
         }

         printf("\nInserisci Contatto: ");
         getInput(45, valore, false);

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

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

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

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

         param[2].buffer_type = MYSQL_TYPE_VAR_STRING;  //IN
         param[2].buffer = valore;
         param[2].buffer_length = strlen(valore);


         if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
           finish_with_stmt_error(conn, prepared_stmt, "Impossibile associare i parametri.\n", true);
         }

         // Run procedure
         if (mysql_stmt_execute(prepared_stmt) != 0) {
           print_stmt_error (prepared_stmt, "Errore durante l'inserimento.");
         } else {
           printf("-------------Contatto aggiunto con successo-------------\n");
         }

         mysql_stmt_close(prepared_stmt);


}
//4
static void scegli_Contatto(MYSQL *conn){
         MYSQL_STMT *prepared_stmt;
```

```
        MYSQL_BIND param[2];

        char cliente[16];
        char valore[45];

        printf("Inserisci Cliente: ");
        getInput(16, cliente, false);
        printf("Inserisci Contatto Preferito:");
        getInput(45, valore, false);

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

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

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

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

        if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
          finish_with_stmt_error(conn, prepared_stmt, "Impossibile associare i parametri.\n", true);
        }

        // Run procedure
        if (mysql_stmt_execute(prepared_stmt) != 0) {
          print_stmt_error (prepared_stmt, "Errore durante l'aggiornamento del contatto preferito.");
        } else {
          printf("-------------Contatto preferito aggiornato con successo-------------\n");
        }

        mysql_stmt_close(prepared_stmt);

}
//5
static void Add_ordine(MYSQL *conn){
        MYSQL_STMT *prepared_stmt;
        MYSQL_BIND param[2];

        int ordine;
        char cliente[17];

        printf("\nInserisci num. Ordine: ");
        scanf("%d", &ordine);
   clear();
```

```
        printf("Inserisci Cliente: ");
        getInput(17, cliente, false);
    clear();

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

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

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

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

        if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
          finish_with_stmt_error(conn, prepared_stmt, "Impossibile associare i parametri.\n", true);
        }

        // Run procedure
        if (mysql_stmt_execute(prepared_stmt) != 0) {
          print_stmt_error (prepared_stmt, "Errore durante l'inserimento dell'ordine.");
        }else{
       printf("-------------Ordine Registrato-------------\n");
       int i=0;
       int loop;
       printf("Quante piante si vogliono inserire: ");
       scanf("%d", &loop);
       while(loop>i){

       Aggiungi_pianta(conn, ordine);

       i++;
       }
    }

}
//6
static void Aggiungi_pianta(MYSQL *conn, int ordine){
    MYSQL_STMT *prepared_stmt;
       MYSQL_BIND param[3];
       char pianta[6];
       int quantita;

       printf("\nInsersci codice Pianta: " );
       scanf("%s", pianta);
```

```
        clear();
        printf("Inserisci quantità acquistata: ");
        scanf("%d", &quantita);
        clear();
        if(!setup_prepared_stmt(&prepared_stmt, "call agg_piante_ordine(?,?,?)", conn)) {
            finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize insertion statement\n",
false);
        }

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

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

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

        param[2].buffer_type = MYSQL_TYPE_LONG;  //IN
        param[2].buffer = &quantita;
        param[2].buffer_length = sizeof(quantita);


        if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
            finish_with_stmt_error(conn, prepared_stmt, "Impossibile associare i parametri.\n", true);
        }

        // Run procedure
        if (mysql_stmt_execute(prepared_stmt) != 0) {
            print_stmt_error (prepared_stmt, "Errore durante l'inserimento.");
        } else {
            printf("----------Pianta aggiunta con successo----------\n");
        }

        mysql_stmt_close(prepared_stmt);


}
//7
static void Aggiungi_pianta_pacco(MYSQL *conn, int pacco){
    MYSQL_STMT *prepared_stmt;
        MYSQL_BIND param[3];
        char pianta[6];


        printf("\nInsersci codice Pianta: " );
        scanf("%s", pianta);
        clear();
        if(!setup_prepared_stmt(&prepared_stmt, "call Aggiungi_piante_a_pacco(?,?)", conn)) {
```

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

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

    param[0].buffer_type = MYSQL_TYPE_LONG;  //IN
    param[0].buffer = &pacco;
    param[0].buffer_length = sizeof(pacco);

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



    if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
        finish_with_stmt_error(conn, prepared_stmt, "Impossibile associare i parametri.\n", true);
    }

    // Run procedure
    if (mysql_stmt_execute(prepared_stmt) != 0) {
        print_stmt_error (prepared_stmt, "Errore durante l'inserimento.");
    } else {
        printf("-------------Pianta aggiunta con successo-------------\n");
    }

    mysql_stmt_close(prepared_stmt);
}
//8
static void Add_pacco(MYSQL *conn){
      MYSQL_STMT *prepared_stmt;
      MYSQL_BIND param[2];

      int ordine;
      int pacco;

      printf("\nInserisci num. Ordine: ");
      scanf("%d", &ordine);
  clear();
      printf("Inserisci numero Pacco: ");
      scanf("%d", &pacco);
  clear();

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

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

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

        param[1].buffer_type = MYSQL_TYPE_LONG;  //IN
        param[1].buffer = &pacco;
        param[1].buffer_length = sizeof(pacco);

        if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
          finish_with_stmt_error(conn, prepared_stmt, "Impossibile associare i parametri.\n", true);
        }

        // Run procedure
        if (mysql_stmt_execute(prepared_stmt) != 0) {
          print_stmt_error (prepared_stmt, "Errore durante la creazione del pacco.");
        }else{
       printf("-------------Pacco creato con successo-------------\n");
       int i=0;
       int loop;
       printf("Quante specie di piante si vogliono inserire nel pacco %d: ", pacco);
       scanf("%d", &loop);
       while(loop>i){

       Aggiungi_pianta_pacco(conn, pacco);

       i++;
       }
    }

}

//9
static void V_ordine(MYSQL *conn){
        MYSQL_STMT *prepared_stmt;
        MYSQL_BIND param[1];

        int ordine;

        printf("Inserisci num. Ordine: ");
        scanf("%d", &ordine);

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

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

        param[0].buffer_type = MYSQL_TYPE_LONG;  //IN
```

```
        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, "Impossibile associare i parametri.\n", true);
        }

        // Run procedure
        if (mysql_stmt_execute(prepared_stmt) != 0) {
          print_stmt_error (prepared_stmt, "Errore durante la visualizzazione dell'ordine.");
        } else {
          dump_result_set(conn, prepared_stmt, "-------------Visualizza informazioni
ordine-------------");
        }
    mysql_stmt_next_result(prepared_stmt);
        mysql_stmt_close(prepared_stmt);
}
//10
static void V_Ordini_Cliente(MYSQL *conn){
        MYSQL_STMT *prepared_stmt;
        MYSQL_BIND param[1];

        char cliente[16];


        printf("\nInserisci codice Cliente: ");
        getInput(16, cliente, false);

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

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

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



        if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
          finish_with_stmt_error(conn, prepared_stmt, "Impossibile associare i parametri.\n", true);
        }

        // Run procedure
        if (mysql_stmt_execute(prepared_stmt) != 0) {
          print_stmt_error (prepared_stmt, "Errore durante la visualizzazione degli ordini del cliente.");
```

```
        } else {
          dump_result_set(conn, prepared_stmt, "-------------Visualizza Ordini di un
Cliente-------------");
        }
    mysql_stmt_next_result(prepared_stmt);
        mysql_stmt_close(prepared_stmt);
}
//11
static void V_pianta(MYSQL *conn){
        MYSQL_STMT *prepared_stmt;
        MYSQL_BIND param[1];

        char pianta[6];


        printf("\nInserisci Pianta: ");
        getInput(6, pianta, false);


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

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

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

        if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
          finish_with_stmt_error(conn, prepared_stmt, "Impossibile associare i parametri.\n", true);
        }

        // Run procedure
        if (mysql_stmt_execute(prepared_stmt) != 0) {
          print_stmt_error (prepared_stmt, "Errore durante la visualizzazione delle informazioni della
pianta.");
        } else {
       dump_result_set(conn, prepared_stmt, "-------------Visualizza Pianta-------------");
    }
    mysql_stmt_next_result(prepared_stmt);
        mysql_stmt_close(prepared_stmt);


}
//12
static void V_colorazione(MYSQL *conn){

        MYSQL_STMT *prepared_stmt;
```

```
        MYSQL_BIND param[1];

        char pianta[6];


        printf("\nInserisci Pianta: ");
        getInput(6, pianta, false);


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

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

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

        if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
          finish_with_stmt_error(conn, prepared_stmt, "Impossibile associare i parametri.\n", true);
        }

        // Run procedure
        if (mysql_stmt_execute(prepared_stmt) != 0) {
          print_stmt_error (prepared_stmt, "Errore durante la visualizzazione dei colori.");
        } else {

      dump_result_set(conn, prepared_stmt, "-------------Visualizza Colorazioni della
pianta-------------");
    }
   mysql_stmt_next_result(prepared_stmt);
        mysql_stmt_close(prepared_stmt);

}
//13
static void V_prezzi(MYSQL *conn){
        MYSQL_STMT *prepared_stmt;
        MYSQL_BIND param[1];


        char pianta[6];


        printf("\nInserisci Pianta:");
        getInput(6, pianta, false);


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

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

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

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

        if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
          finish_with_stmt_error(conn, prepared_stmt, "Impossibile associare i parametri.\n", true);
        }

        // Run procedure
        if (mysql_stmt_execute(prepared_stmt) != 0) {
          print_stmt_error (prepared_stmt, "Errore durante la visualizzazione dello storico dei prezzi
della pianta.");
        } else{
       dump_result_set(conn, prepared_stmt, "-------------Storico Prezzo Pianta-------------");
   }

   mysql_stmt_next_result(prepared_stmt);
        mysql_stmt_close(prepared_stmt);

}
//14
static void V_piante_inserire(MYSQL *conn){
        MYSQL_STMT *prepared_stmt;
        MYSQL_BIND param[1];

        int ordine;

        printf("Inserisci num. Ordine: ");
        scanf("%d", &ordine);

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

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

        param[0].buffer_type = MYSQL_TYPE_LONG;  //IN
        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, "Impossibile associare i parametri.\n", true);
        }

        // Run procedure
        if (mysql_stmt_execute(prepared_stmt) != 0) {
          print_stmt_error (prepared_stmt, "Errore durante la visualizzazione delle piante da inserire
nel pacco.");
        } else {
       dump_result_set(conn, prepared_stmt, "-------------Piante da Inserire nel Pacco-------------");
        }
   mysql_stmt_next_result(prepared_stmt);
        mysql_stmt_close(prepared_stmt);

}
//15
static void V_piante_inserite(MYSQL *conn){
        MYSQL_STMT *prepared_stmt;
        MYSQL_BIND param[1];

        int Pacco;

        printf("Inserisci num. Pacco: ");
        scanf("%d", &Pacco);

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

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

        param[0].buffer_type = MYSQL_TYPE_LONG;  //IN
        param[0].buffer = &Pacco;
        param[0].buffer_length = sizeof(Pacco);



        if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
          finish_with_stmt_error(conn, prepared_stmt, "Impossibile associare i parametri.\n", true);
        }

        // Run procedure
        if (mysql_stmt_execute(prepared_stmt) != 0) {
          print_stmt_error (prepared_stmt, "Errore durante la visualizzazione delle piante inserite nel
pacco.");
        } else {
          dump_result_set(conn, prepared_stmt, "-------------Piante inserite nel Pacco-------------");
        }
   mysql_stmt_next_result(prepared_stmt);
```

```
        mysql_stmt_close(prepared_stmt);

}
//16
static void V_costo_ordine(MYSQL *conn){

   MYSQL_STMT *prepared_stmt;
        MYSQL_BIND param[1];

        int ordine;

        printf("Inserisci num. Ordine: ");
        scanf("%d", &ordine);

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

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

        param[0].buffer_type = MYSQL_TYPE_LONG;  //IN
        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, "Impossibile associare i parametri.\n", true);
        }

        // Run procedure
        if (mysql_stmt_execute(prepared_stmt) != 0) {
          print_stmt_error (prepared_stmt, "Errore durante la visualizzazione delle costo totale
dell'ordine.");
        } else {
          dump_result_set(conn, prepared_stmt, "-------------Costo Totale Ordine-------------");
        }
   mysql_stmt_next_result(prepared_stmt);
        mysql_stmt_close(prepared_stmt);

}


void run_as_operatore(MYSQL *conn){

   char options[17]={'1','2','3','4','5','6','7','8','9','a','b','c','d','e','f','g', 'h'};
        char op;
   int ordine, pacco;
   int i, loop;
```

```
  printf("Passo al ruolo di Operatore...\n");

  if(!parse_config("users/operatore.json", &conf)) {
    fprintf(stderr, "Impossibile caricare info Operatore\n");
    exit(EXIT_FAILURE);
  }

  if(mysql_change_user(conn, "root", "root1234", "mydb")) {
    fprintf(stderr, "mysql_change_user() failed\n");
    exit(EXIT_FAILURE);
  }

  while(true) {

    printf("*** Cosa desidera fare ? ***\n\n");
    printf("1) Aggiungi Cliente Privato\n");
    printf("2) Modifica Cliente Rivendita\n");
printf("3) Aggiungi Contatti\n");
    printf("4) Scegli Contatto Preferito del Cliente\n");
    printf("5) Aggiungi Ordine\n");
printf("6) Aggiungere Piante ad Ordine\n");
printf("7) Aggiungi Pacco\n");
printf("8) Inserisci Piante in un Pacco\n");
    printf("9) Visualizza Ordine\n");
printf("a) Visualizza Ordini Cliente\n");
    printf("b) Visualizza Pianta\n");
    printf("c) Visualizza Colorazione delle Pianta\n");
printf("d) Visualizzare Storico Prezzi di Pianta\n");
    printf("e) Visualizzare Piante da inserire nel Pacco\n");
    printf("f) Visualizza Piante del Pacco\n");
printf("g) Visualizza Costo Totale Ordine\n");
    printf("h) Esci\n");

op=multiChoice("select option", options, 17);

    switch(op) {

            case '1':
                    Add_cPrivato(conn);
                    break;
            case '2':
                    Add_cRivendita(conn);
                    break;
    case '3':
                    Add_contatti(conn);
                    break;
            case '4':
                    scegli_Contatto(conn);
                    break;
```

```
            case '5':
                    Add_ordine(conn);
                    break;
    case '6':

       printf("Inserire numero dell'ordine: ");
       scanf("%d", &ordine);
       i=0;
       printf("Quante piante si vogliono inserire: ");
       scanf("%d", &loop);
       while(loop>i){

          Aggiungi_pianta(conn, ordine);

       i++;
       }

       break;
    case '7':
       Add_pacco(conn);
                    break;
    case '8':
       printf("Inserire numero del Pacco: ");
       scanf("%d", &pacco);
       i=0;
       int loop;
       printf("Quante specie di piante si vogliono inserire nel pacco %d: ", pacco);
       scanf("%d", &loop);
       while(loop>i){
          Aggiungi_pianta_pacco(conn, pacco);
          i++;
       }
       break;

            case '9':
                    V_ordine(conn);
                    break;
    case 'a':
                    V_Ordini_Cliente(conn);
                    break;
            case 'b':
                    V_pianta(conn);
                    break;
            case 'c':
                    V_colorazione(conn);
                    break;
    case 'd':
                    V_prezzi(conn);
                    break;
            case 'e':
```

```
                        V_piante_inserire(conn);
                        break;
                case 'f':
                        V_piante_inserite(conn);
                        break;
        case 'g':
           V_costo_ordine(conn);
           break;
                case 'h':
                        return;
                default:
                        fprintf(stderr, "Invalid condition at %s:%d\n", __FILE__, __LINE__);
                        abort();
        }

      }
}
```