Basi di Dati e Conoscenza

Progetto A.A. 2019/2020

Sistema per la gestione della vendita all'ingrosso di piante

0252795

Simone Tiberi

Indice

1. Descrizione del Minimondo

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
4	giardino	esterni	Il termine *esterni* si adatta meglio al minimondo di interesse.
5	appartamento	interni	Il termine *interni* si adatta meglio al minimondo di interesse.
9	persona	privato	Il termine *privato* rappresenta meglio il concetto duale di *rivendita* nel minimondo di interesse.
9	indirizzo della persona	indirizzo di residenza	Come si capisce dalla specifica ulteriore a riga 16-17, i due indirizzi corrispondono. Inoltre il termine *residenza* meglio rappresenta un concetto generico del cliente anziché della sua specializzazione persona.
10	indirizzo della rivendita	indirizzo di residenza	Come si capisce dalla specifica ulteriore a riga 16-17, i due indirizzi corrispondono. Inoltre il termine *residenza* meglio rappresenta un concetto generico del cliente anziché della sua specializzazione rivendita.
12	mezzo di comunicazione	contatto	La frase “eventualmente associato ad altri contatti (con la possibilità, sempre, di indicarne uno preferito).” (righe 14-15) lascia intendere che i due termini sono effettivamente sinonimi.
21	ordine	richiesta di fornitura	Il termine *ordine* verrà utilizzato nella specifica disambiguata per ulteriori scopi.
24	acquisti	ordine	Il termine *ordine* è stato scelto per rappresentare l’acquisto poiché più semanticamente corretto nelle diverse frasi in cui compare.
28	destinatario	cliente	Il termine *cliente* è già presente nella specifica. Il termine *destinatario* non viene utilizzato altrove nella specifica ed è meno adatto a rappresentare il concetto.

Specifica disambiguata

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 esterni o da interni 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 di residenza, mentre 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 contatto 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, richieste di forniture ai fornitori per mantenere una giacenza di tutte le specie di piante trattate. Le specie di piante trattate sono gestite dai manager di Verde S.r.l.

Si vuole tener traccia di tutti gli ordini effettuati 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 cliente in caso di problemi nella consegna. Non è possibile aprire un ordine se non vi è disponibilità in magazzino.

Nel listino prezzi 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 pianta	Bene venduto dalla Verde S.r.l. Classificazione: • Da interni • Da esterni ▪ Verde ▪ Fiorita		Pacco, Fornitore ed Ordine

	Può infine essere esotica.		
Cliente	Acquirente dei beni venduti dalla Verde S.r.l.		Ordine e Contatto
Privato	Possibile tipologia di cliente della Verde S.r.l		
Rivendita	Possibile tipologia di cliente della Verde S.r.l		Referente
Referente	Persona di riferimento per una rivendita nei confronti di Verde S.r.l		Contatto, Rivendita, Ordine
Fornitore	Ente che fornisce piante alla Verde S.r.l.		Specie di pianta
Ordine	Richiesta di acquisto piante da parte dei clienti di Verde S.r.l.		Specie di pianta, Contatto, Pacco e Cliente, Referente
Pacco	Elemento che permette la suddivisione delle piante da inoltrare nell'ordine da evadere		Ordine e Specie di pianta
Manager	Dipendente della Verde S.r.l incaricato della gestione delle specie vendute e i loro prezzi.		Specie di pianta
Addetto dipartimento magazzino	Dipendente della Verde S.r.l. incaricato di richiedere la fornitura di piante per mantenerne la giacenza		Specie di pianta e Fornitore
Operatore pacchi	Dipendente della Verde S.r.l. incaricato della lavorazione dell'ordine		Specie di pianta, Pacco, Ordine

Raggruppamento dei requisiti in insiemi omogenei

Frasi relative a *specie di pianta*
L'azienda Verde S.r.l. gestisce la vendita all'ingrosso di piante da interni ed esterni. L'azienda tratta diverse piante, ciascuna caratterizzata sia dal nome latino che dal nome comune, e da un codice univoco alfanumerico attraverso cui viene identificata. Per ciascuna pianta è inoltre noto se sia

tipicamente da interni o da esterni e se sia una pianta esotica o meno. Le piante possono essere verdi oppure fiorite. Nel caso di piante fiorite, sono note tutte le colorazioni in cui una pianta è disponibile.
Il fornitore può fornire diverse piante. Verde S.r.l. ha un dipartimento di gestione di magazzino che tiene traccia delle giacenze ed effettua, periodicamente, richieste di fornitura ai fornitori per mantenere una giacenza di tutte le piante trattate. Le piante trattate sono gestite dai manager di Verde S.r.l.
Un ordine, effettuato in una data specifica, è relativo a una certa quantità di piante.
Nel listino prezzi si vuole tener traccia dei prezzi assunti nel tempo da ciascuna pianta. 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.
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 di piante differenti. 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.
Nel listino prezzi si vuole tener traccia dei prezzi assunti nel tempo da ciascuna specie di piante.

Frasi relative a *cliente*
L'azienda gestisce ordini massivi ed ha un parco clienti sia di rivendite che di privati.
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 mantenere un numero arbitrario di contatti, ad esempio numeri di telefono, di cellulare, di indirizzi email ed è possibile indicare qual è il contatto preferito per essere contattati.
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.
Si vuole tener traccia di tutti gli ordini eseguiti da ciascun cliente.
Nell'ambito di un ordine è di interesse sapere […] quale contatto fornire al corriere per mettersi in contatto con il cliente in caso di problemi nella consegna.

Frasi relative a *privato*
Per ciascun privato sono noti il codice fiscale, il nome e l'indirizzo di residenza.

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

Frasi relative a *referente*
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).
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 cliente in caso di problemi nella consegna.

Frasi relative a *fornitore*
I fornitori di Verde S.r.l. sono identificati attraverso un codice fornitore; per ciascun fornitore sono inoltre noti il nome, il codice fiscale ed un numero arbitrario di indirizzi. Il fornitore può fornire diverse specie di piante. Verde S.r.l. ha un dipartimento di gestione di magazzino che tiene traccia delle giacenze ed effettua, periodicamente, richiesta di fornitura ai fornitori per mantenere una giacenza di tutte le specie di piante trattate.

Frasi relative a *ordine*
Si vuole tener traccia di tutti gli ordini effettuati 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 cliente in caso di problemi nella consegna. Non è possibile aprire un ordine se non vi è disponibilità in magazzino.
Una variazione di prezzo non deve avere effetto su un ordine già aperto ma non ancora finalizzato.
Gli ordini vengono evasi in pacchi. Un ordine è associato ad un numero arbitrario di pacchi.

Frasi relative a *pacco*
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.

Frasi relative a *manager*
Le specie di piante trattate sono gestite dai manager di Verde S.r.l.
I prezzi sono gestiti dai manager di Verde S.r.l.

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

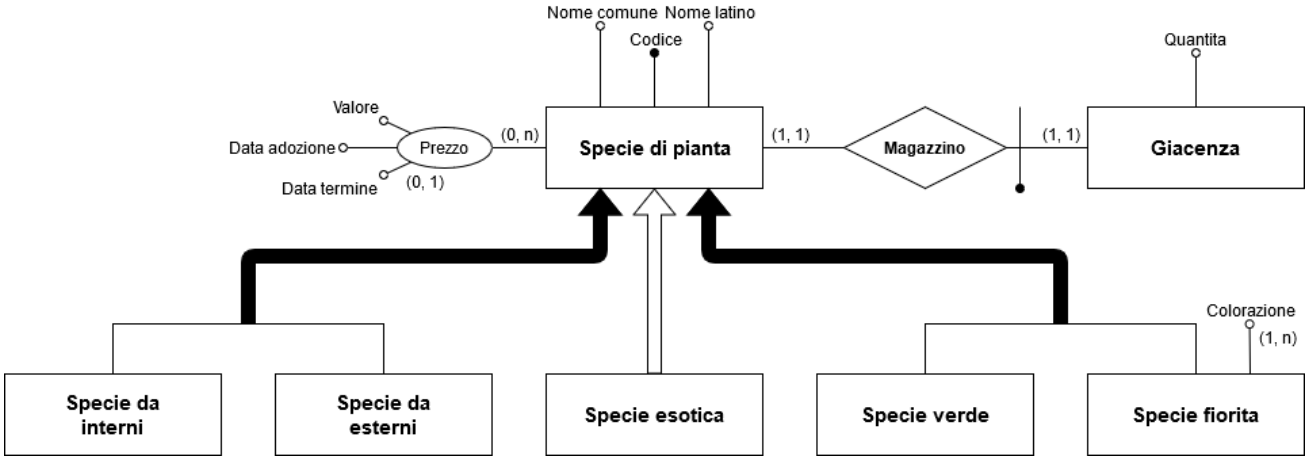
Frasi relative a *operatore pacchi*
Quando si prepara un pacco, è di interesse per l'operatore sapere quali piante devono essere ancora inserite nei pacchi, al fine di evadere correttamente l'ordine.

3. Progettazione concettuale

Costruzione dello schema E-R

Per la realizzazione dello schema E-R definitivo è stato seguito un approccio bottom-up i cui passi sono qui sotto riportati.

La modellazione delle entità durante la prima fase di analisi ha prodotto i seguenti schemi parziali:

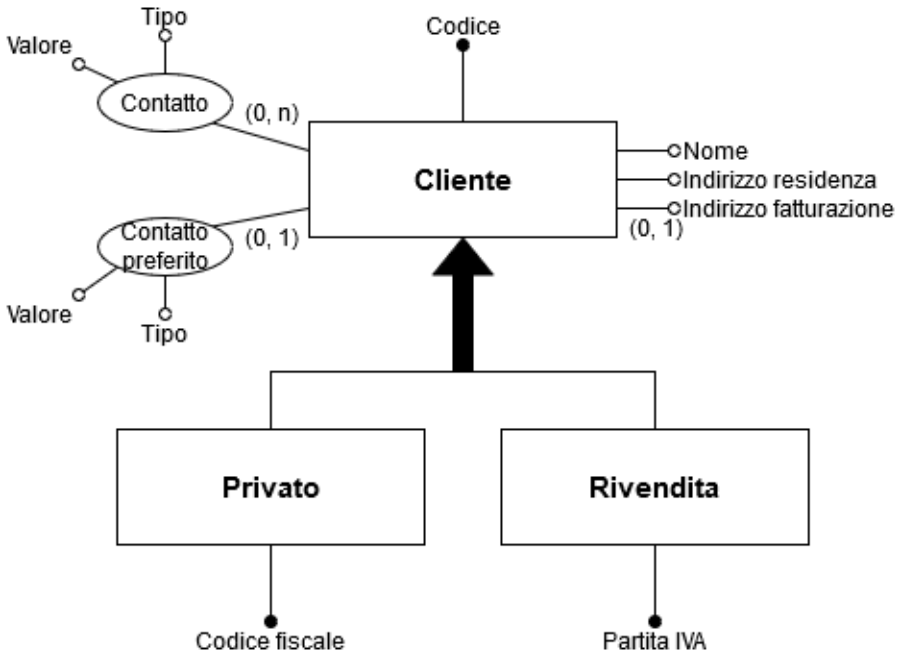


L'entità *specie di pianta* è stata sin da subito modellata mediante generalizzazioni totali e parziali per caratterizzare al meglio la classificazione evidenziata nella prima delle frasi nella sezione relativa ad essa.

È stato introdotto un attributo composto multivalore per rappresentare il prezzo, poiché come si evince nel penultimo paragrafo della specifica, tale necessità delle informazioni relative a data adozione e data termine di utilizzo.

Inoltre ai fini del progetto non è di interesse tenere traccia della singola pianta bensì soltanto della quantità per ciascuna specie motivo per cui è stata introdotta l'entità *giacenza* debole rispetto a *specie di pianta*.

In questa fase progettuale è stato scelto di scindere le entità *specie di pianta* e *giacenza* poiché intrinsecamente distinte (il primo infatti caratterizza un concetto astratto di classificazione il secondo la situazione attuale concreta del magazzino relativamente alla prima).

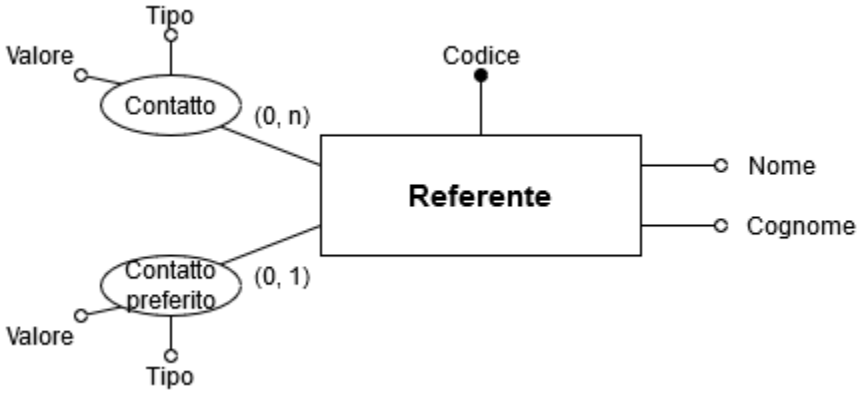


L'entità *cliente* è stata sin da subito modellata mediante generalizzazione totale per catturare al meglio la distinzione fra *privati* e *rivendite*.

Nel testo non è indicato un identificativo univoco per il *cliente* (entità padre) per questo motivo ne è stato introdotto uno ad-hoc.

Il testo suggerisce però esplicitamente degli identificativi univoci per le entità figlie (ovvero il codice fiscale e la partita iva).

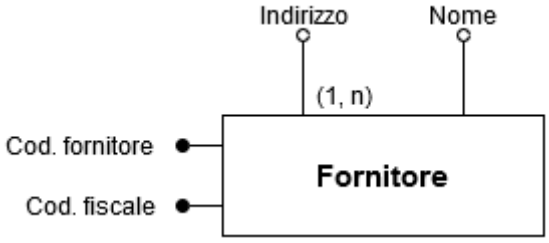
Inoltre l'indirizzo di fatturazione dato che può essere differente o meno dall'indirizzo di residenza è stato modellato come attributo opzionale.



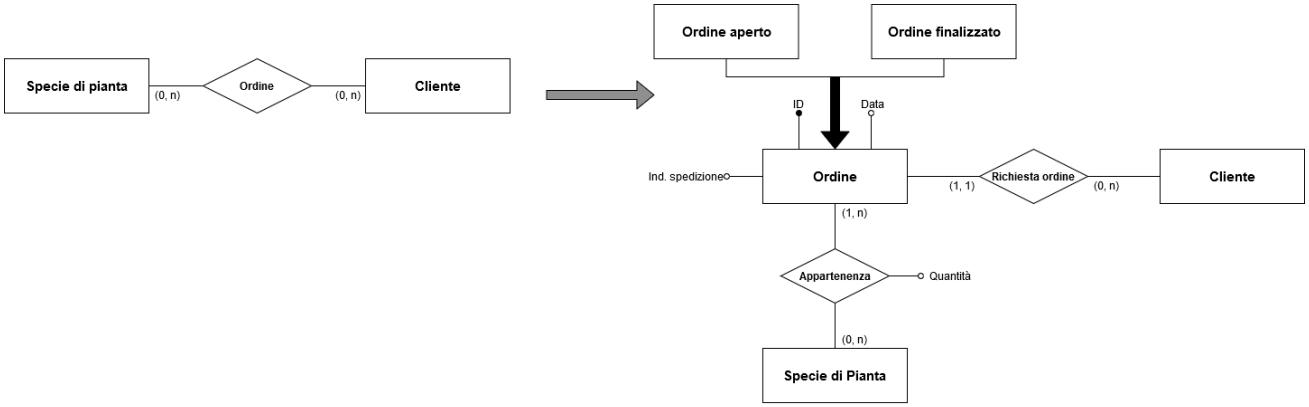
Seppure il concetto di *referente* non sia particolarmente enfatizzato all'interno del testo, sin dall'inizio dell'analisi è stato deciso di reificarlo ad entità mediante un processo di partizionamento verticale.

Tale decisione è stata presa al fine di evitare l'utilizzo di un attributo composto dell'entità *cliente* di elevata complessità per rappresentare il *referente* (di fatti avrebbe contenuto a sua volta una coppia di ulteriori attributi composti per la gestione dei contatti).

Essendo nata da un processo di partizionamento è stato scelto di identificare l'entità *referente* debolmente mediante l'identificatore esterno di cliente.



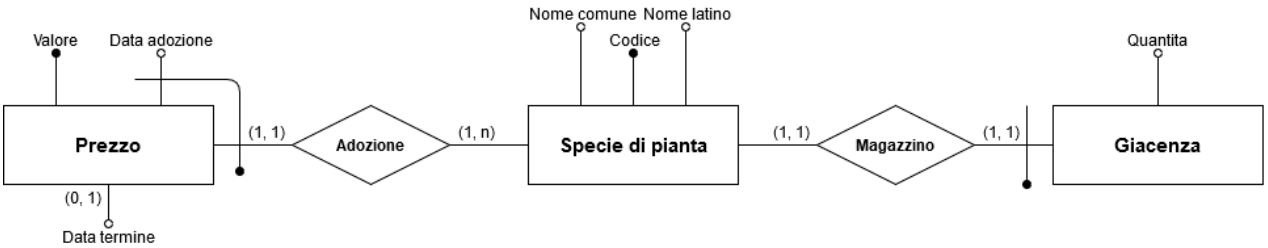
Per quanto riguarda l'entità *fornitore* la specifica fornita suggerisce due identificativi possibili entrambi riportati nello schema parziale qui riportato.



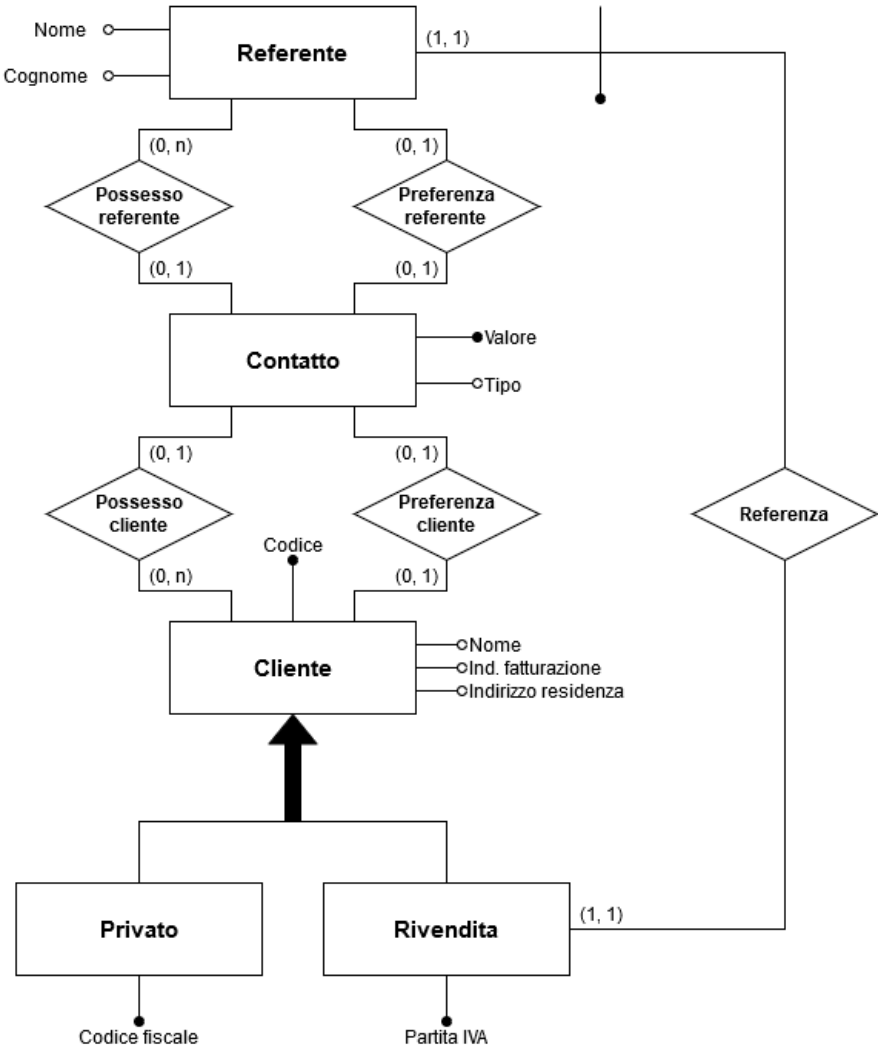
Inizialmente l'entità *ordine* è stata modellata come associazione m:n fra le entità *cliente* e *specie di pianta*. Tale è stata poi reificata ad entità per far si che un cliente potesse effettuare più ordini a cui afferisse la medesima specie di pianta. In questo scenario è stato aggiunto un attributo identificatore ad hoc per l'entità (La scelta non è ricaduta sull'identificazione mediante attributo esterno per una sorta di continuità con le altre piattaforme già esistenti in cui solitamente ciascun ordine è identificato da un codice numerico).

Per garantire infine la distinzione fra *ordine aperto* e *finalizzato* infine è stata quindi introdotta una generalizzazione totale.

In una seconda fase del processo di analisi sono state individuate le seguenti decomposizioni a partire dagli schemi sopra presentati:

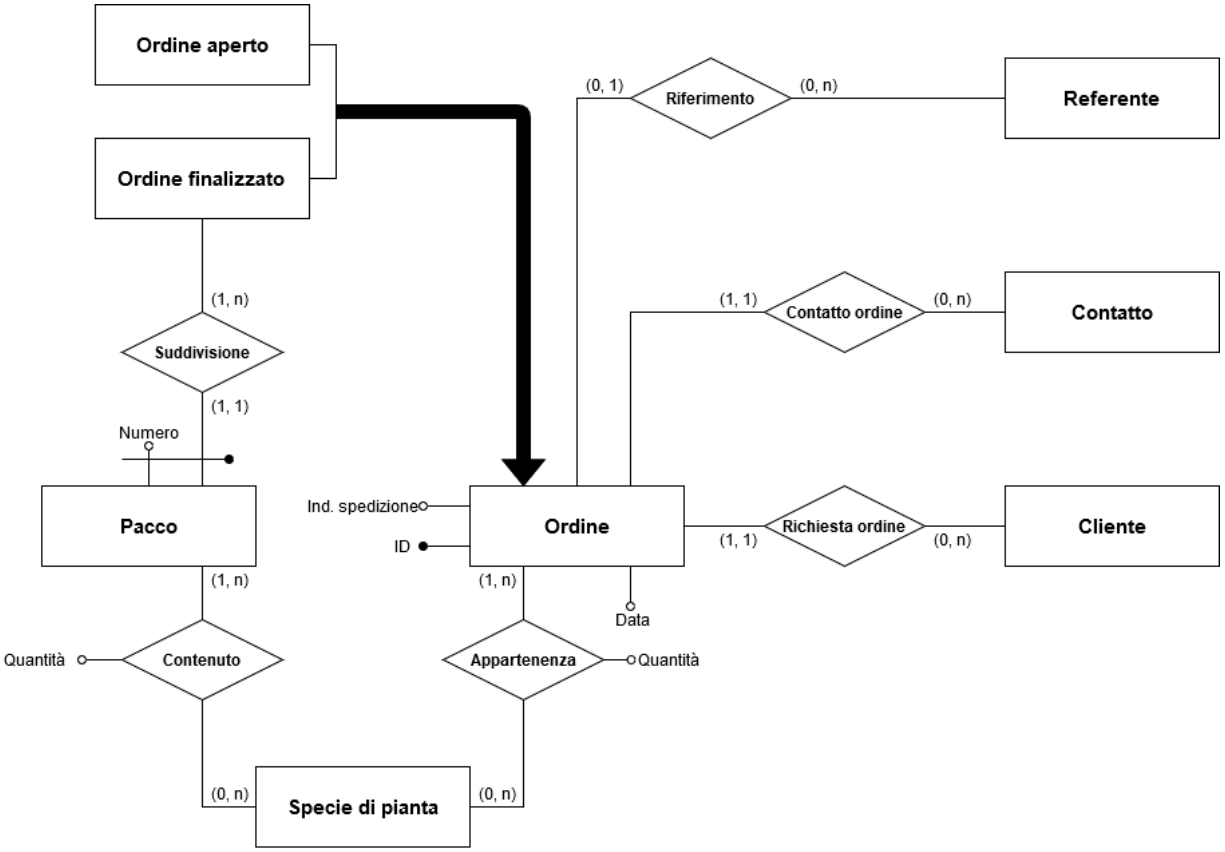


Sull'attributo composto multivalore prezzo dell'entità *specie di pianta* è stato attuato un processo di reificazione di attributo di entità, come è evidenziato nello schema parziale sopra riportato.



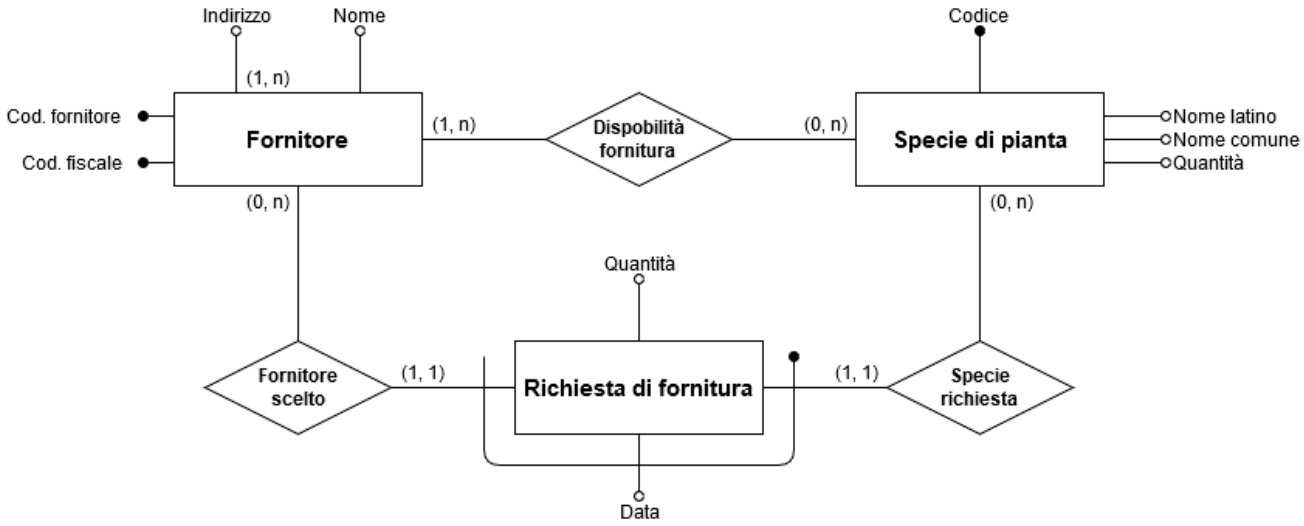
Anche l'attributo contatto delle entità *cliente* e *referente* ha subito lo stesso processo di reificazione.

In particolare la neonata entità *contatto* è stata collegata alle altre due entità mediante due coppie di associazioni distinte per mantenere la distinzione fra contatto e contatto preferito come nell'analisi precedentemente effettuata.



Lo schema parziale qui sopra riportato è relativo la struttura del core del diagramma, ovvero dell'insieme delle associazioni che legano i vari elementi che sono stati precedentemente descritti singolarmente.

A tal proposito è stata introdotta l'entità *pacco* (debole rispetto ad *ordine*) identificata da un numero progressivo unico all'interno dell'*ordine* stesso. Tale entità è stata introdotta al fine di discriminare l'appartenenza di una specie ad un ordine e l'appartenenza di una pianta ad un pacco.

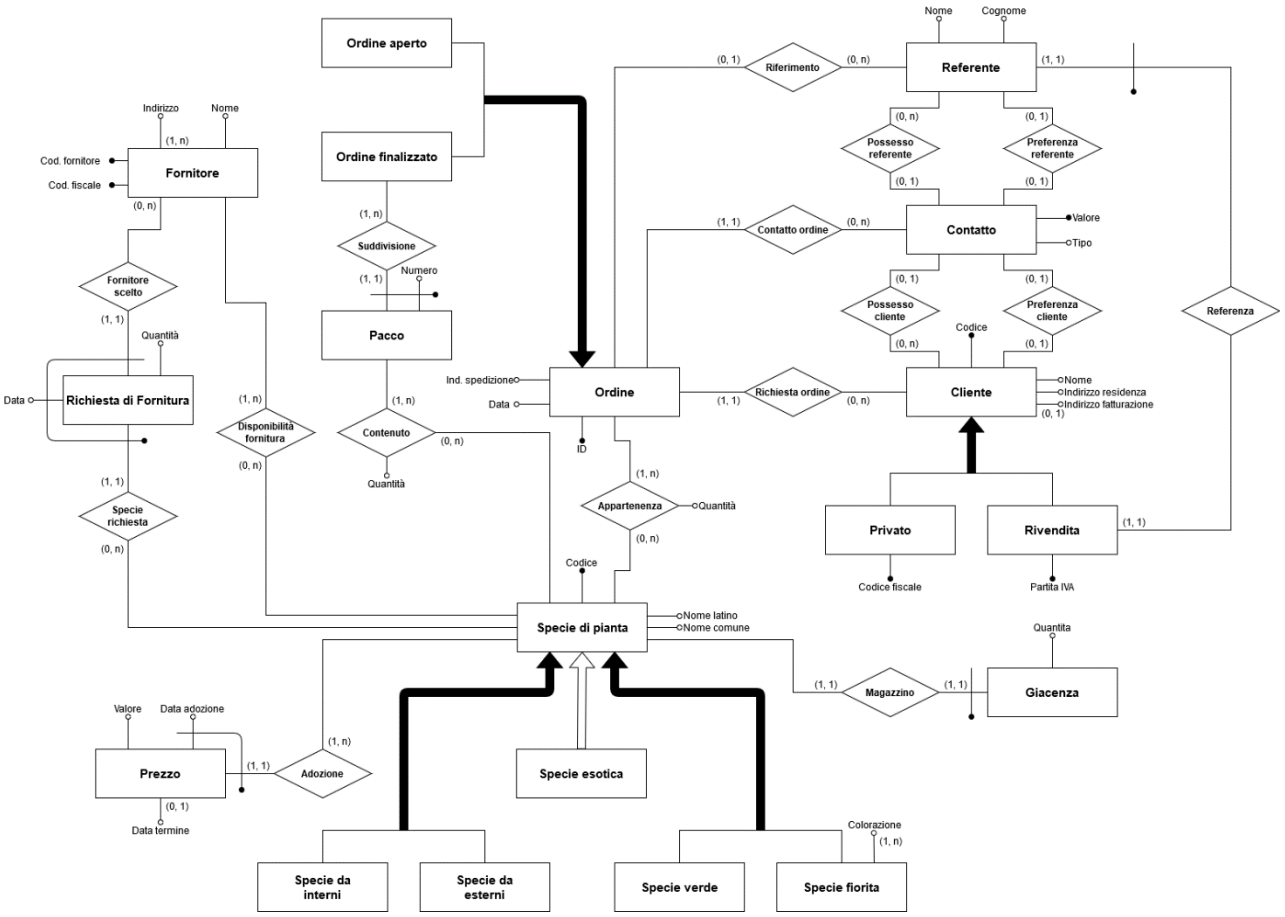


Nello schema parziale presentato qui sopra è evidenziata la scelta di collegare le entità *fornitore* e *specie di pianta* in due modi distinti:

- mediante una associazione m:n per evidenziare il fatto che un fornitore possiede una determinata specie (ovvero può rifornire Verde S.r.l.)
- reificando una associazione m:n fra i due per far si che un fornitore possa fornire la stessa specie diverse volte.

Integrazione finale

Non essendovi conflitti da risolvere di seguito è direttamente riportato lo schema definitivo.



Regole aziendali

RV01) Il prezzo attualmente adottato DEVE essere l'unico avente *data termine* nulla.

RV02) La *data* di *adozione* di un prezzo DEVE essere antecedente alla *data* di *termine.*

RV03) Il contatto preferito DEVE appartenere all'insieme dei contatti posseduti.

RV04) La quantità di piante di una specie richiesta in un ordine DEVE essere minore o uguale della giacenza della stessa.

RV05) La quantità di piante di una specie contenuta in un pacco DEVE esser minore o uguale del numero di piante della specie selezionate per l'ordine a cui esso afferisce.

RV06) Il referente associato all'ordine, se esiste, DEVE corrispondere al referente associato alla rivendita.

RV07) Il contatto associato all'ordine dal cliente DEVE appartenere all'insieme dei contatti posseduti dallo stesso.

RV08) Una specie di piante fornita da un fornitore DEVE appartenere alla lista di quelle possedute

Dizionario dei dati

Entità	Descrizione	Attributi	Identificatori
Specie di pianta	Classe d'appartenenza delle piante	Codice, Nome latino, Nome comune	Codice
Specie da interni	Particolare sottoinsieme delle specie esistenti	Codice, Nome latino, Nome comune	Codice
Specie da esterni	Particolare sottoinsieme delle specie esistenti	Codice, Nome latino, Nome comune	Codice
Specie esotica	Particolare sottoinsieme delle specie esistenti	Codice, Nome latino, Nome comune	Codice
Specie verde	Particolare sottoinsieme delle specie esistenti	Codice, Nome latino, Nome comune	Codice
Specie fiorita	Particolare sottoinsieme delle specie esistenti	Codice, Nome latino, Nome comune, Colorazione	Codice
Giacenza	Rappresentazione dello stato del magazzino relativamente ad una data specie di piante	Quantità	Codice (ext. id.)
Prezzo	Valore economico di una specie di piante vendute dalla Verde S.r.l.	Valore, Data adozione, Data termine	Codice (ext. id.) + Data adozione
Fornitore	Ente che fornisce piante alla Verde S.r.l.	Codice fornitore, Codice fiscale, Nome, Indirizzo	Codice fornitore *or* Codice fiscale

Richiesta di Fornitura	Lotto di piante richieste consegnato da un certo fornitore	Quantità, Data	Quantità + Codice fornitore (ext. id.) + Codice (ext. id.)
Cliente	Acquirente delle piante vendute dalla Verde S.r.l.	Codice, Nome, Indirizzo fatturazione, Indirizzo residenza	Codice
Privato	Possibile categoria di clienti gestita da Verde S.r.l.	Codice, Nome, Indirizzo fatturazione, Indirizzo residenza, Codice fiscale	Codice *or* Codice fiscale
Rivendita	Possibile categoria di clienti gestita da Verde S.r.l.	Codice, Nome, Indirizzo fatturazione, Indirizzo residenza, Partita IVA	Codice *or* Partita IVA
Contatto	Mezzo di comunicazione che permette di rintracciare un cliente e/o un referente della Verde S.r.l.	Valore, Tipo	Valore
Referente	Persona di riferimento per una rivendita cliente della Verde S.r.l.	Nome, Cognome	Codice cliente (ext. id.)
Ordine	Acquisto effettuato da un cliente della Verde S.r.l.	ID, Data, Indirizzo spedizione	ID
Ordine aperto	Ordine aperto dal cliente, il quale ancora non ha concluso la fase di inserimento prodotti.	ID, Data, Indirizzo spedizione	ID
Ordine finalizzato	Ordine tale per cui l'utente ha terminato l'inserimento di prodotti e che può essere processato dagli operatori.	ID, Data, Indirizzo spedizione	ID
Pacco	Unità di partizionamento di un ordine per la spedizione	Numero	Codice (ext. id.) + Data (ext. id.) + Numero

4. Progettazione logica

Volume dei dati

Concetto nello schema	Tipo[1]	Volume atteso
Specie di pianta	E	$P(t) = 2000(1 - e^{-t})$
Specie da interni	E	$25\%\ di\ P(t)$
Specie da esterni	E	$75\%\ di\ P(t)$
Specie esotica	E	$5\%\ di\ P(t)$
Specie verde	E	$60\%\ di\ P(t)$
Specie fiorita	E	$40\%\ di\ P(t)$
Giacenza	E	$P(t)$
Magazzino	R	$P(t)$
Prezzo	E	$50 * P(t)$
Adozione	R	$50 * P(t)$
Fornitore	E	$F(t) = 100(1 - e^{-t})$
Richiesta di fornitura	E	$200 * F(t)$
Specie richiesta	R	$200 * F(t)$
Fornitore scelto	R	$200 * F(t)$
Disponibilità fornitura	R	$20 * F(t)$
Cliente	E	$C(t) = 10000(1 - e^{-t})$
Privato	E	$40\%\ di\ C(t)$
Rivendita	E	$60\%\ di\ C(t)$
Referente	E	$R(t) = 60\%\ di\ C(t)$
Contatto	E	$2.5 * (C(t) + R(t))$
Possesso cliente	R	$2.5 * C(t)$
Preferenza cliente	R	$80\%\ di\ C(t)$
Referenza	R	$R(t)$
Possesso referente	R	$2.5 * R(t)$

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

Preferenza referente	R	$80\%\ di\ R(t)$
Ordine	E	$O(t) = 200 * C(t)$
Ordine finalizzato	E	$80\%\ di\ O(t)$
Ordine aperto	E	$20\%\ di\ O(t)$
Riferimento	R	$60\%\ di\ O(t)$
Pacco	E	$P(t) = 2.5 * (0.8 * O(t))$
Appartenenza	R	$5 * O(t)$
Richiesta ordine	R	$O(t)$
Contatto ordine	R	$O(t)$
Suddivisione	R	$P(t)$
Contenuto	R	$2.5 * P(t)$

Tavola delle operazioni

Cod.	Descrizione	Frequenza attesa
XCS-01	Permette, di aprire un nuovo ordine noti il codice del cliente, l'indirizzo, il contatto e il numero di piante della specie da inserire.	$C(t \to \infty)/_{week} \sim 1500/_{day}$
XCS-02	Permette di aggiungere una nuova specie ad un ordine esistente noti ambedue gli identificativi.	$3 * C(t \to \infty)/_{week} \sim 4500/_{day}$
XCS-03	Permette di rimuovere una specie da un ordine esistente noti ambedue gli identificativi.	$C(t \to \infty)/_{month} \sim 350/_{day}$
XCS-04	Permette di modificare la quantità di piante di una specie appartenente ad un ordine esistente noti ambedue gli identificativi.	$C(t \to \infty)/_{week} \sim 1500/_{day}$
XCS-05	Permette di dichiarare finalizzato un ordine di cui si conosce l'identificativo	$C(t \to \infty)/_{week} \sim 1500/_{day}$
XCS-06	Permette la visualizzazione di un report dettagliato relativo ad un ordine di cui si conosce l'identificativo.	$2 * C(t \to \infty)/_{week} \sim 3000/_{day}$
XCS-07	Permette di modificare l'indirizzo di residenza di un cliente noto il suo identificativo.	$C(t \to \infty)/_{year} \sim 30/_{day}$

XCS-08	Permette di modificare l'indirizzo di fatturazione di un cliente noto il suo identificativo.	$C(t \to \infty)/_{year} \sim 30/_{day}$
XCS-09	Permette l'aggiunta di un contatto alla lista dei posseduti di un cliente di cui si conosce il codice.	$2 * C(t \to \infty)/_{year} \sim 60/_{day}$
XCS-10	Permette la rimozione di un contatto dalla lista dei posseduti di un cliente di cui si conosce il codice.	$2 * C(t \to \infty)/_{year} \sim 60/_{day}$
XCS-11	Permette l'aggiornamento del contattto preferito di un cliente di cui si conosce il codice.	$3 * C(t \to \infty)/_{year} \sim 90/_{day}$
XCS-12	Permette di visualizzare l'insieme dei contatti di un cliente di cui si conosce il codice	$6 * C(t \to \infty)/_{year} \sim 120/_{day}$
XCS-13	Permette di visualizzare l'insieme degli ordini effettuati da un cliente di cui di conosce il codice	$8 * C(t \to \infty)/_{week} \sim 12000/_{day}$
XCS-14	Permette di visualizzare l'insieme delle specie appartenenti ad un ordine di cui è noto il codice	$C(t \to \infty)/_{week} \sim 1500/_{day}$
XWM-01	Permette la ricerca di specie di piante per nome	$50 * C(t \to \infty)/_{week} \sim 75000/_{day}$
RCS-01	Permette l'aggiunta di un contatto alla lista dei posseduti di un referente di cui si conosce l'identificativo.	$2 * R(t \to \infty)/_{year} \sim 35/_{day}$
RCS-02	Permette la rimozione di un contatto dalla lista dei posseduti di un referente di cui si conosce l'identificativo.	$2 * R(t \to \infty)/_{year} \sim 35/_{day}$
RCS-03	Permette di aggiornare il contattto preferito di un referente di cui si conosce il codice.	$3 * R(t \to \infty)/_{year} \sim 50/_{day}$
RCS-04	Permette di visualizzare l'insieme dei contatti di un referente di cui si conosce il codice	$6 * R(t \to \infty)/_{year} \sim 100/_{day}$
MNG-01	Permette l'aggiunta di un nuova specie di piante all'insieme di quelle trattate.	$0.01 * P(t \to \infty)/_{year} \sim 20/_{year}$
MNG-02	Permette la rimozione di una specie di piante dall'insieme delle piante trattate.	$0.01 * P(t \to \infty)/_{year} \sim 20/_{year}$

MNG-03	Permette l'aggiunta di una colorazione possibile per una specie fiorita trattata di cui si conosce il codice.	$[0.01 * (0.4 * P(t \to \infty))]/_{year} \sim 8/_{year}$
MNG-04	Permette la rimozione di una colorazione possibile per una specie fiorita trattata di cui si conosce il codice.	$[0.01 * (0.4 * P(t \to \infty))]/_{year} \sim 8/_{year}$
MNG-05	Permette la modifica del prezzo attualmente adottato per una specie di pianta di cui si conosce il codice.	$5 * P(t \to \infty)/_{year} \sim 30/_{day}$
MNG-06	Permette la visualizzazione di un report sul trend di vendita di una specie di cui è noto il codice.	$2 * 0.01 * P(t \to \infty)/_{year} \sim 40/_{year}$
MNG-07	Permette di visualizzare l'insieme delle colorazioni di una specie fiorita di cui si conosce il codice	$[0.02 * (0.4 * P(t \to \infty))]/_{year} \sim 16/_{year}$
EVN-01	Permette la rimuozione dall'archivio di informazioni relative ad ordini, richieste di fornitura e prezzi passati	$1/_{year}$
EVN-02	Permette l'aggiornamento della giacenza al fronte delle richieste di fornitura effettuate	$1/_{week}$
WHC-01	Permette di richiedere ad un fornitore, di cui si conosce l'identificativo una certa quantità di piante di una specie di cui è noto il codice.	$P(t \to \infty)/_{month} \sim 70/_{day}$
WHC-02	Permette di inserire un nuovo fornitore nel sistema.	$0.1 * F(t \to \infty)/_{year} \sim 10/_{year}$
WHC-03	Permette di aggiungere una specie alla lista delle disponibili di un fornitore di cui si conosce il codice	$0.1 * (20 * F(t \to \infty))/_{year} \sim 200/_{year}$
WHC-04	Permette di visualizzare un report che evidenzia le specie che necessitano un rifornimento della giacenza	$1/_{week}$
WHC-05	Permette di visualizzare l'elenco delle specie disponibili di un fornitore di cui è noto il codice	$P(t \to \infty)/_{month} \sim 70/_{day}$
WHC-06	Permette di aggiungere un indirizzo alla lista degli indirizzi di un fornitore di cui è noto il codice	$F(t \to \infty)/_{year} \sim 100/_{year}$

WHC-07	Permette di visualizzare l'elenco dei fornitori nel sistema	$P(t \to \infty)/_{month} \sim 70/_{day}$
OPC-01	Permette la visualizzazione dei dettagli sullo smistamento in pacchi di un'ordine finalizzato di cui si conosce il codice.	$(0.8 * O(t \to \infty))/_{day} \sim 1600000/_{day}$
OPC-02	Permette la creazione di un pacco (inserendovi almeno una pianta) relativo ad un ordine di cui si conosce il codice.	$(0.8 * O(t \to \infty))/_{day} \sim 1600000/_{day}$
OPC-03	Permette l'aggiunta di un certo numero di piante afferenti ad una specie ad un pacco di cui si conosce l'identificativo	$2.5 * (0.8 * O(t \to \infty))/_{day} \sim 4000000/_{day}$
OPC-04	Permette di visualizzare l'elenco delle specie ancora da impacchettare	$2.5 * (0.8 * O(t \to \infty))/_{day} \sim 4000000/_{day}$
OPC-05	Permette di visualizzare lo stato di un ordine di cui si conosce il codice	$2.5 * (0.8 * O(t \to \infty))/_{day} \sim 4000000/_{day}$
NRG-01	Permette l'inserimento di un nuovo cliente privato nella base di dati.	$4000 * e^{-\Delta t}$ (per $t \to \infty$ si assume l'operazione sia eseguiuta un numero infinitesimo di volte rispetto al valore di $C(t)$)
NRG-02	Permette l'inserimento di un nuovo cliente rivendita nella base di dati.	$6000 * e^{-\Delta t}$ (per $t \to \infty$ si assume l'operazione sia eseguiuta un numero infinitesimo di volte rispetto al valore di $C(t)$)

Costo delle operazioni

Operazione: XCS-01			
Concetto	**Costrutto**	**Accessi**	**Tipo**
Cliente	Entità	1	L
Pref/Poss cliente	Associazione	1	L
Contatto	Entità	1	L
Specie di pianta	Entità	1	L
Magazzino	Associazione	1	L
Giacenza	Entità	1	L
Rivendita	Entità	1	L
Referenza	Associazione	1	L
Referente	Entità	1	L

Ordine	Entità	1	S
Ordine aperto	Entità	1	S
Appartenenza	Associazione	1	S
Richiesta ordine	Associazione	1	S
Riferimento	Associazione	1	S
Contatto ordine	Associazione	1	S
Giacenza	Entità	1	S
Costo effettivo: 23			

Nel caso di privati il costo totale decrementa di 5 unità per via dell'assenza del referente (assenti righe 7-8-9-14 della tabella).

Operazioni: XCS-02, XCS-04			
Concetto	**Costrutto**	**Accessi**	**Tipo**
Specie di pianta	Entità	1	L
Magazzino	Associazione	1	L
Giacenza	Entità	1	L
Ordine	Entità	1	L
Appartenenza	Associazione	1	S
Giacenza	Entità	1	S
Costo effettivo: 8			

Operazione: XCS-05			
Concetto	**Costrutto**	**Accessi**	**Tipo**
Ordine aperto	Entità	1	S (rimozione)
Ordine finalizzato	Entità	1	S
Costo effettivo: 4			

Operazione: XCS-07, XCS-08			
Concetto	**Costrutto**	**Accessi**	**Tipo**
Cliente	Entità	1	S
Costo effettivo: 2			

Operazione: XCS-09			
Concetto	**Costrutto**	**Accessi**	**Tipo**
Cliente	Entità	1	L
Contatto	Entità	1	S
Possesso cliente	Associazione	1	S
Costo effettivo: 5			

Operazione: XCS-11			
Concetto	**Costrutto**	**Accessi**	**Tipo**
Cliente	Entità	1	L
Contatto	Entità	1	L
Preferenza cliente	Associazione	1	S
Costo effettivo: 4			

Operazione: RCS-01			
Concetto	**Costrutto**	**Accessi**	**Tipo**
Referente	Entità	1	L
Contatto	Entità	1	S
Possesso referente	Associazione	1	S
Costo effettivo: 5			

Operazione: RCS-03			
Concetto	**Costrutto**	**Accessi**	**Tipo**
Referente	Entità	1	L
Contatto	Entità	1	L
Preferenza referente	Associazione	1	S
Costo effettivo: 4			

Operazione: MNG-01			
Concetto	**Costrutto**	**Accessi**	**Tipo**
Specie di pianta	Entità	1	S
Entità figlie di specie di pianta	Entità	2/3	S

Magazzino	Associazione	1	S
Giacenza	Entità	1	S
Prezzo	Entità	1	S
Adozione	Associazione	1	S
Costo effettivo: 10/12			

L'entità *specie di pianta* possiede 5 entità figlie ad essa collegata. Quattro di esse a due a due formano due generalizzazioni totali a cui si aggiunge una 5 entità figlia che è in generalizzazione parziale e sovrapposta con le precedenti. Se ne deduce che ogni qual volta si procede con la scrittura di un nuovo elemento *specie di pianta*, si procede con 2 oppure 3 scritture aggiuntive per le entità figlie.

Operazione: MNG-03			
Concetto	**Costrutto**	**Accessi**	**Tipo**
Specie fiorita	Entità	1	S
Costo effettivo: 2			

Operazione: MNG-05			
Concetto	**Costrutto**	**Accessi**	**Tipo**
Specie di pianta	Entità	1	L
Adozione	Associazione	1	S
Prezzo	Entità	2	S
Costo effettivo: 7			

Operazione: WHC-01			
Concetto	**Costrutto**	**Accessi**	**Tipo**
Fornitore	Entità	1	L
Specie di pianta	Entità	1	L
Richiesta fornitura	Entità	1	S
Fornitore scelto	Associazione	1	S
Specie richiesta	Associazione	1	S
Costo effettivo: 8			

Operazione: WHC-02			
Concetto	**Costrutto**	**Accessi**	**Tipo**
Specie di pianta	Entità	1	L
Fornitore	Entità	1	S
Disponibilità fornitura	Associazione	1	S
Costo effettivo: 5			

Operazione: WHC-03			
Concetto	**Costrutto**	**Accessi**	**Tipo**
Specie di pianta	Entità	1	L
Fornitore	Entità	1	L
Disponibilità fornitura	Associazione	1	S
Costo effettivo: 4			

Operazione: WHC-6			
Concetto	**Costrutto**	**Accessi**	**Tipo**
Fornitore	Entità	1	S
Costo effettivo: 2			

Operazione: OPC-02			
Concetto	**Costrutto**	**Accessi**	**Tipo**
Specie di pianta	Entità	1	L
Ordine finalizzato	Entità	1	L
Pacco	Entita	1	S
Contenuto	Associazione	1	S
Suddivisione	Associazione	1	S
Costo effettivo: 8			

Operazione: OPC-03			
Concetto	**Costrutto**	**Accessi**	**Tipo**
Specie di pianta	Entità	1	L
Pacco	Entita	1	L

Contenuto	Associazione	1	S
Costo effettivo: 4			

Operazione: NRG-01			
Concetto	**Costrutto**	**Accessi**	**Tipo**
Cliente	Entità	1	S
Cliente privato	Entità	1	S
Costo effettivo: 4			

Operazione: NRG-02			
Concetto	**Costrutto**	**Accessi**	**Tipo**
Cliente	Entità	1	S
Cliente rivendita	Entità	1	S
Referenza	Associazione	1	S
Referente	Entità	1	S
Costo effettivo: 8			

Ristrutturazione dello schema E-R

Analisi delle ridondanze

In questa fase di ristrutturazione è stato scelto di eliminare l'associazione *riferimento* che lega *ordine* con *referente.*

Infatti l'entità *referente* è in associazione 1:1 con l'entità *rivendita* che già è associata (tramite l'entità padre) all'*ordine* mediante l'associazione *richiesta d'ordine*. Non essendo presenti operazioni che richiedono esclusivamente l'accesso al referente associato ad un'ordine senza conoscere i dettagli della rivendita corrispondente è preferibile rinunciare alla ridondanza per diminuire il costo dell'operazione XCS-01 interessata.

Operazione XCS-01: COSTO TOTALE: $23 \rightarrow 18$ $[-22\%]$ (risparmio di 3 letture ed una scrittura).

È stato inoltre aggiunto l'attributo pendente all'entità *richiesta di fornitura* per rendere più adattabile il sistema a modifichè future. Di fatti in questo scenario sarebbe possibile verificare se un rifornimento è stato realmente effettuato basandosi soltanto sulla data poiché l'operazione EVN-02 settimanalmente aggiorna le giacenze al fronte di richieste.

Se in futuro si implementasse un aggiornamento manuale asincrono delle giacenze si rischierebbe di aggiornare più volte le tuple rendendo lo stato della base di dati inconsistente rispetto al magazzino reale.

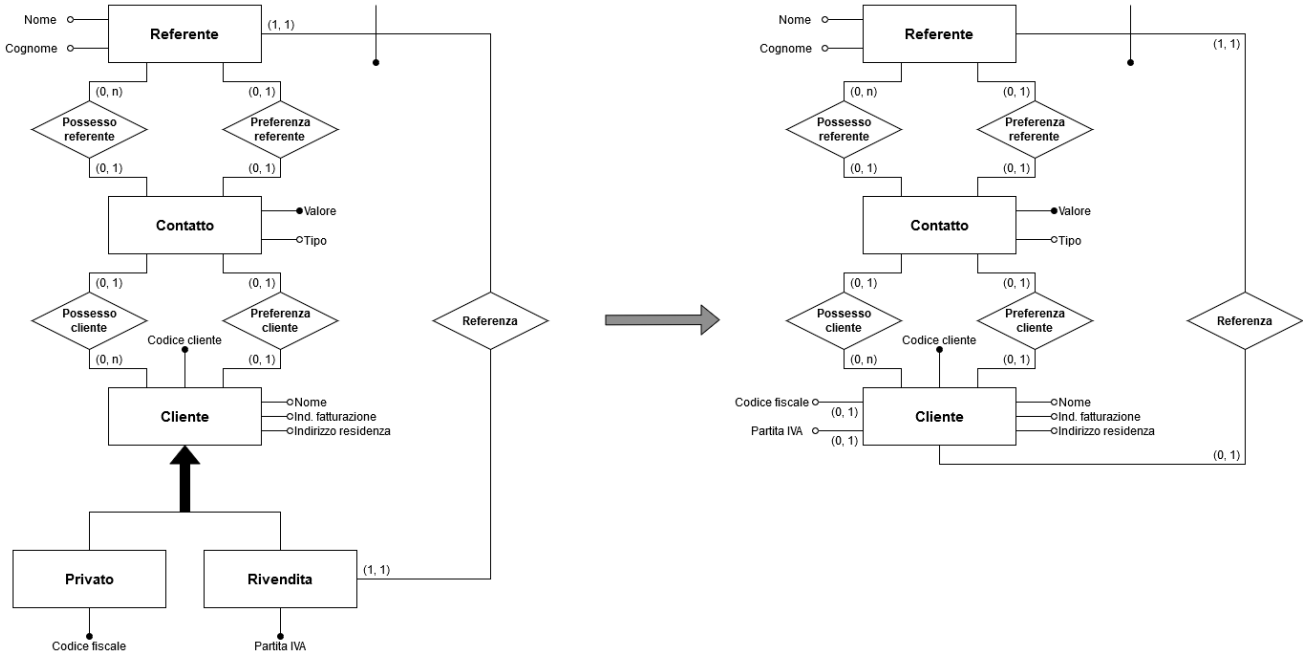
Eliminazione delle generalizzazioni

Per tutte e tre le generalizzazioni presenti nello schema E-R è stato scelto in questa fase progettuale l'accorpamento all'entità padre come strategia di eliminazione.

Per la generalizzzazione *cliente-privato-rivendita*, la scelta è legata alla mancanza di distinzione fra le due specializzazioni nelle operazioni elencate (praticamente tutte si riferiscono al cliente in generale) e nelle associazioni presenti nel diagramma E-R (fatta eccezione di *referenza*).

Per questo motivo la cardinalità minima di tale associazione rispetto a *cliente* subisce una variazione da 1 a 0 (per i privati non è richiesto di memorizzare il referente).

Non è stato introdotto l'attributo tipo in questa fase d'analisi poiché gli attributi codice fiscale e partita IVA rispettivamente di privato e rivendita sono mutalmente esclusivi in cliente (l'assenza di uno di essi determina univocamente l'una o l'altra specializzazione).

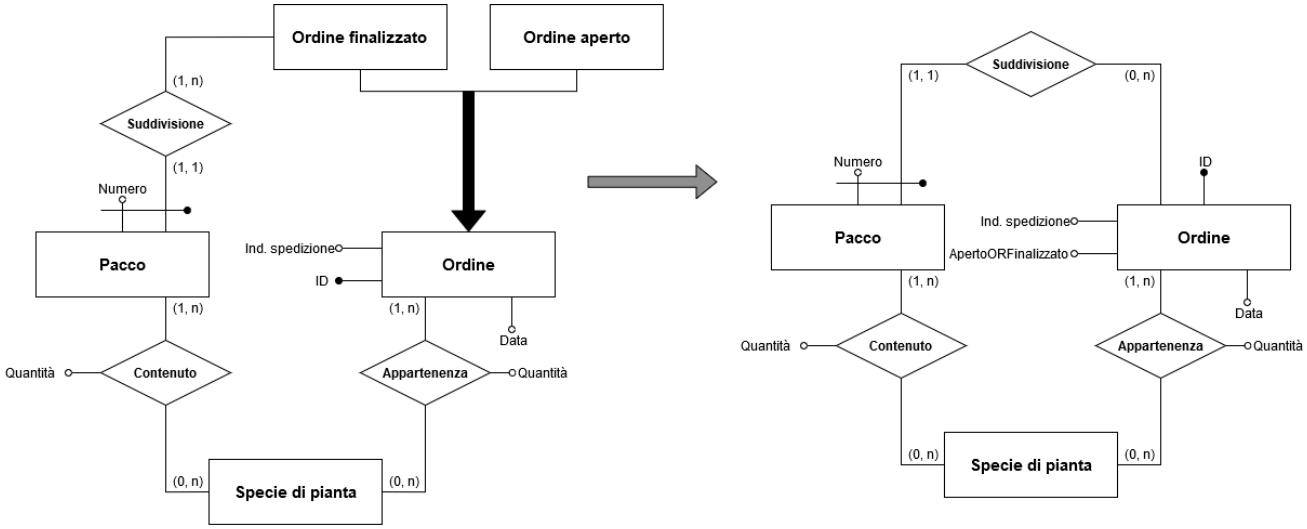


Per la generalizzazione *ordine-ordine finalizzato-ordine aperto* la scelta è legata alla natura stessa della generalizzazione: essa era stata introdotta in fase di progettazione concettuale con l'unico obiettivo di distinguere due diverse tipologie di ordini. Ai fini dell'ottimizzazione in questa fase è stato preferito inserire l'attributo *ApertoORFinalizzato* a tale scopo.

L'operazione XCS-05 è l'unica interessata dalla modifica e, come viene mostrato qui sotto, subisce un notevole miglioramento.

Operazione XCS-05: COSTO TOTALE: $4 \rightarrow 2$ $[-50\%]$ (risparmio di una scrittura).

L'associazione *suddivisione* rispetto ad *ordine finalizzato* a seguito della modifica subisce una variazione di cardinalità minima (da (1, n) a (0, n)).



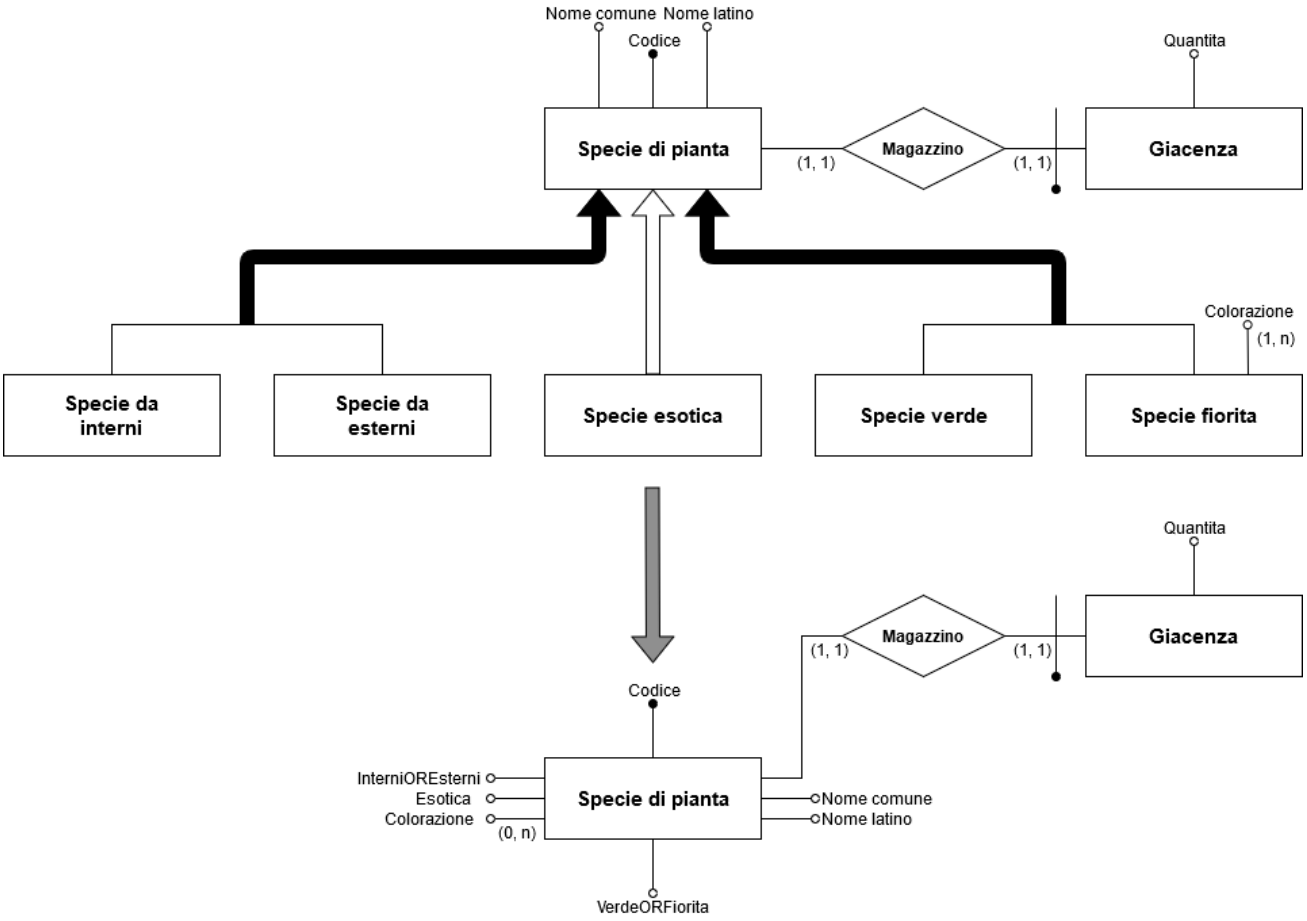
Per la generalizzazione relativa alla *specie di pianta* infine la scelta è legata a motivazioni analoghe al caso precedente.

Anche in questo caso le varie entità figlie non sono altro che diverse categorie di specie che risulta essere l'unica entità interessata da operazioni ed associazioni nel diagramma E-R.

Inoltre tale accorpamento causa una miglioria sul costo dell'operazione MN01:

Operazione MN01: COSTO TOTALE: $10/12 \rightarrow 6\ [-40/50\%]$ (risparmio di 2 o 3 scritture).

Per la rimozione della generalizzazione totale che coinvolge *specie da interni* e *specie da esterni* si introduce lì attributo *InterniOREsterni*, per la generalizzazione parziale *specie esotica* si l'attributo *Esotica*, infine per la generalizzazione totale relativa a *specie verde* o *fiorita* l'attributo *VerdeORFiorita.*



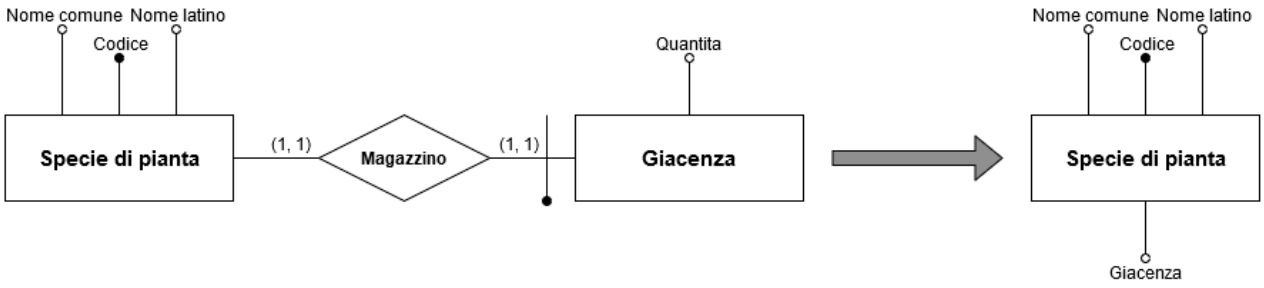
Partizionamento/Accorpamento entità e associazioni

L'unica operazione di accorpamento effettuata sullo schema prodotto fin qui è quella che coinvolge le entità *specie di pianta* e *giacenza.*

Sebbene le due furono divise in fase di progettazione concettuale per motivi semantici in questa fase, al fine di ottimizzare le prestazioni è stato scelto l'accorpamento.

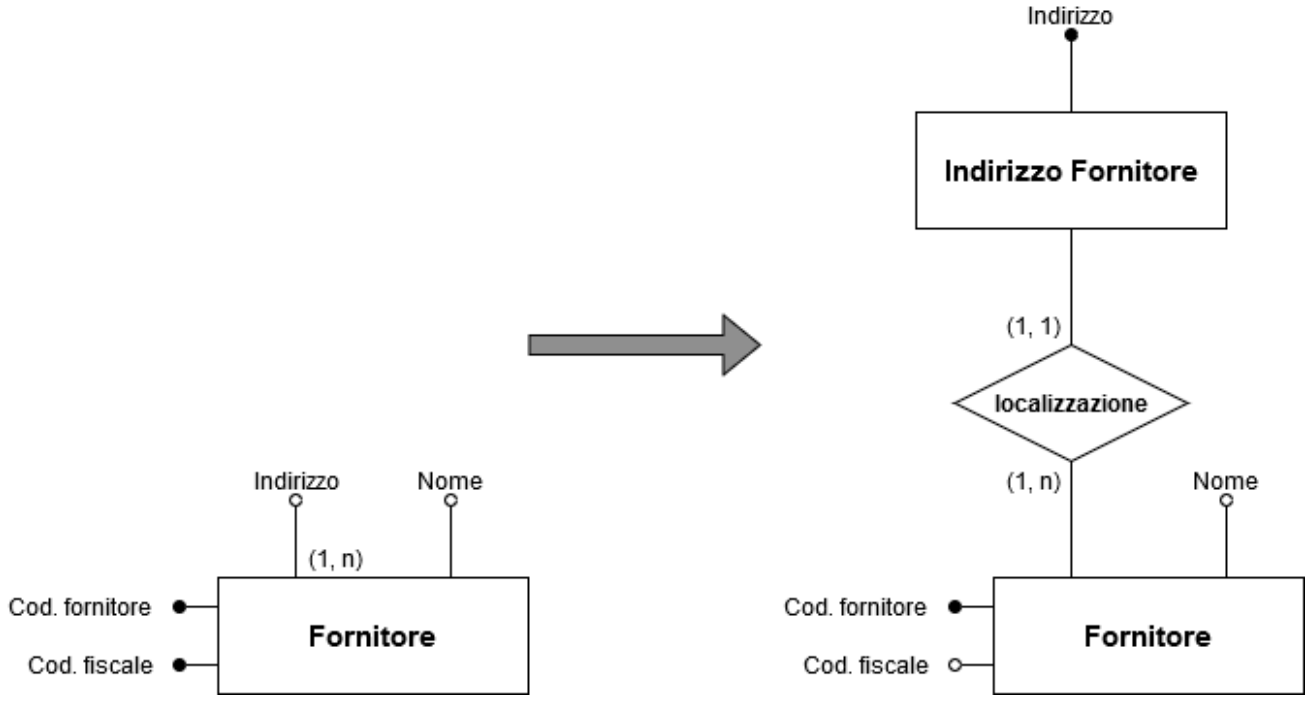
In quasi ogni operazione che coinvolge la *specie di pianta* è di interesse conoscere la giacenza in magazzino della stessa.

Ciascuna operazione che prevedeva 3 letture/scritture distinte nello schema concettuale originario, in quello di sotto riportato ora ne necessita soltanto 1.

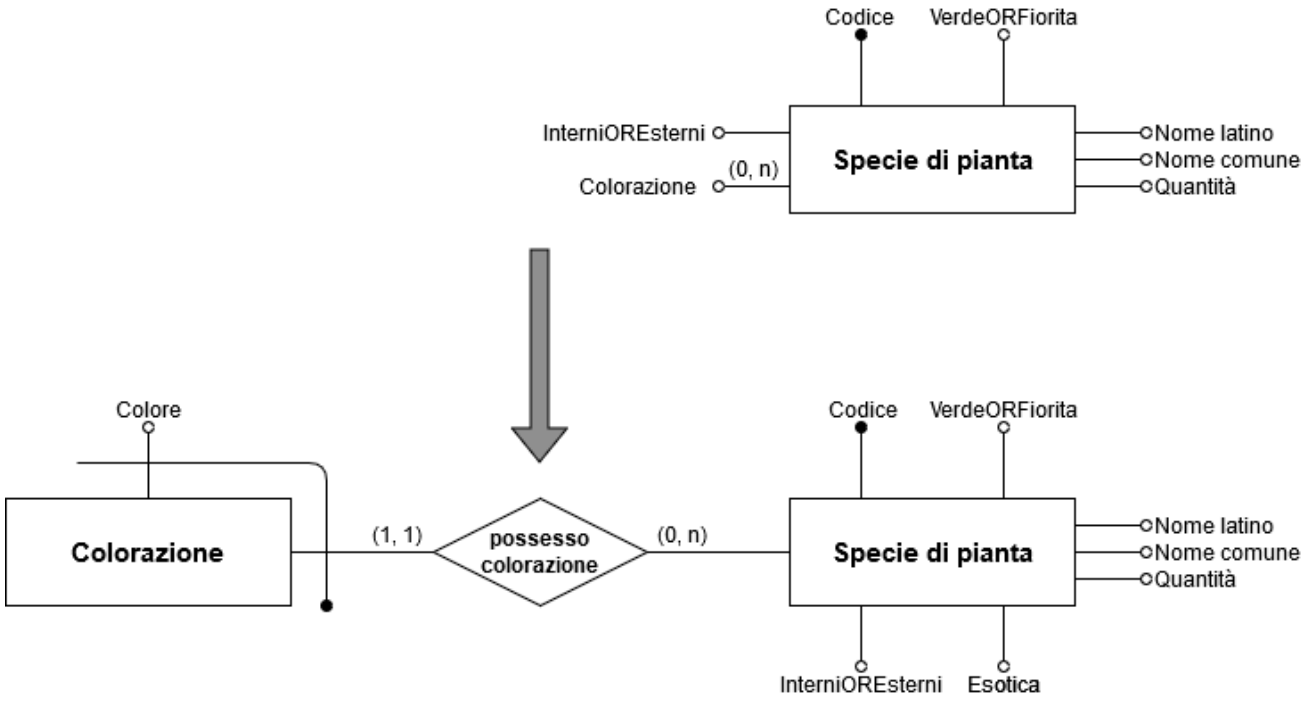


Nella fase di partizionamento di entità invece sono stati reificati gli attributi multivalore presenti nello schema per adattarli al modello relazionale.

L'attributo indirizzo del fornitore (cardinalità (1, n)) dell'entità *fornitore* viene reificato ad entità e viene collegato mediante un'associazione 1:n come viene mostrato nello schema sottostante.



L'attributo colorazione dell'entità *specie di pianta* (cardinalità (0, n)) viene reificato ad entità e viene collegato mediante un associazione 0:n come viene mostrato nello schema sottostante.



Scelta degli identificatori primari

Di seguito vengono riportate specifiche ulteriori sulle scelte degli identificatori. Le entità che non compaiono in questa lista mantengono l'identificatore presentato nel diagramma E-R.

- **Fornitore**
  per tale entità si sceglie come identificatore primario il codice fornitore poiché più pertinente al dominio di interesse rispetto al codice fiscale.

- **Cliente**
  per tale entità si accorpano gli attributi codice (che era stato inserito in fase progettuale senza dirette indicazioni provenienti dalla descrizione del minimondo) partita IVA e codice fiscale in una nuova versione dell'attributo codice. Viene in aggiunta introdotto un attributo *PrivatoORRivendita* per discriminare la categoria di cliente.
  In questo modo si passa da una configurazione con tre attributi di cui due opzionali, ad un'altra con soli 2 attributi non opzionali.
  Il nuovo identificatore corrisponderà al valore del codice fiscale nel caso di un privato o alla partita iva nel caso di una rivendita.

  ESEMPIO:

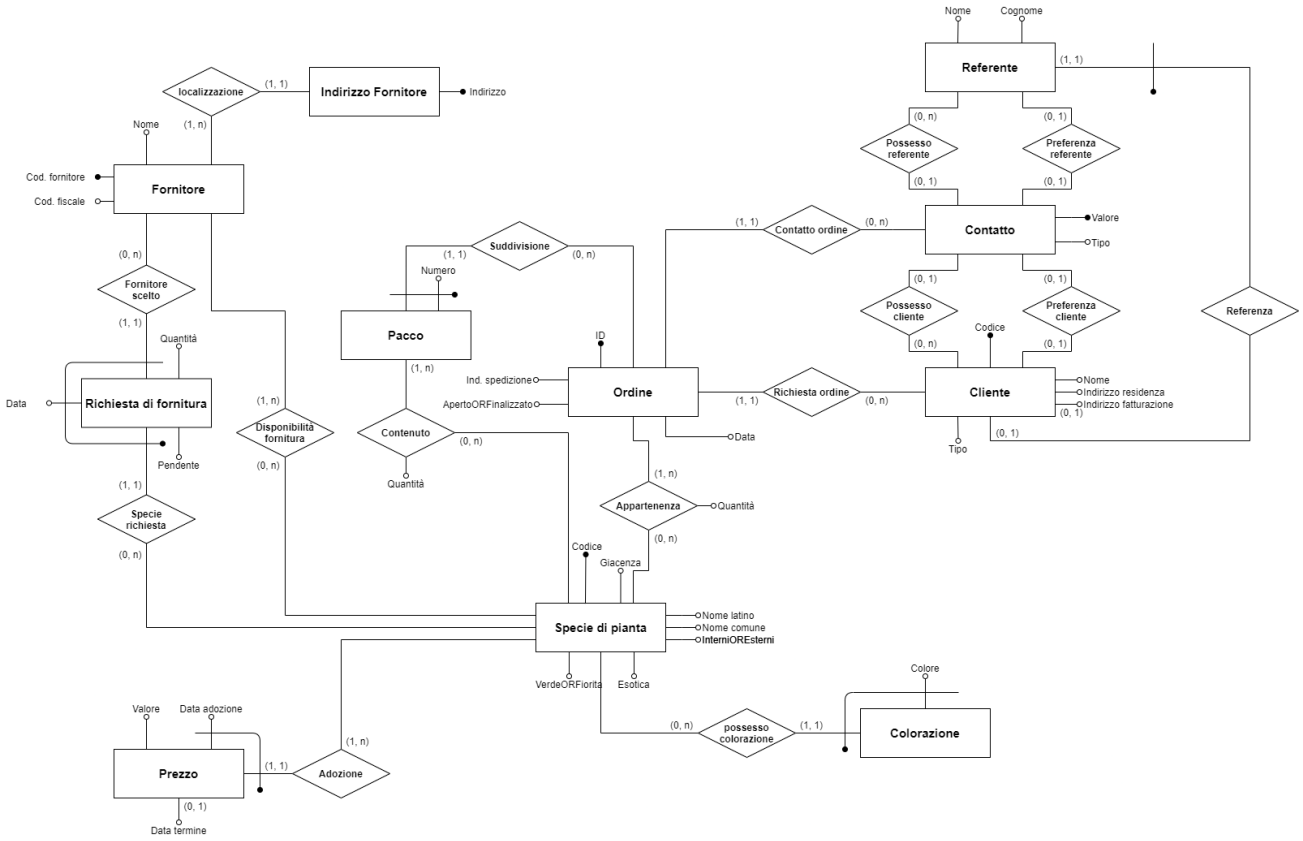
  1) elementi appartenenti all'entità cliente prima del cambio di identifiatore:
     c1(00001, "Mario", "Via Roma, 29", "Via Roma, 29", "RSSMRA80A01H501U", null)
     c2(00002, "Flora", "Via Verdi, 1", "Via Torino, 12", null, "0764352056C")

  2) elemento appartenente all'entità cliente dopo il cambio di identificatore:
     c1("RSSMRA80A01H501U", "Mario", "Via Roma, 29", "Via Roma, 29", P)
     c2("0764352056C", "Flora", "Via Verdi, 1", "Via Torino, 12", R)

Trasformazione di attributi e identificatori

Non vi sono attributi ripetuti e tutti gli identificatori esterni finora mantenuti rimangono inalterati.

Traduzione di entità e associazioni



La convenzione adottata nella traduzione è stata quella di utilizzare underscore e caratteri minuscoli non accentati per rappresentare sia relazioni che attributi, ed inoltre l'adozione del plurale per i nomi delle relazioni anziché il singolare utilizzato nella nomenclatura delle entità modello concettuale.

Inoltre per una maggiore chiarezza nella traduzione sono state attuate le seguente modifiche ai nomi:

- le associazioni *possesso cliente/referente* sono state tradotte in *contatti_cliente/referente*
- gli attributi esotica dell'entità *specie di pianta* e pendente dell'entità *richiesta di fornitura* sono stati tradotti rispettivamente in esotica_si_no e pendente_si_no .
- le associazioni *appartenenza* e *contenuto* sono state tradotte rispettivamente in appartenenza_ordini e contenuto_pacchi.

La traduzione nel modello relazionale porta al seguente schema:

appartenenza_ordini (ordine, specie_di_pianta, quantita)

clienti (codice, nome, indirizzo_residenza, indirizzo_fatturazione*, privato_or_rivendita, contatto_preferito*)

colorazioni (specie_fiorita, colore)

contatti (valore, tipo)

contatti_clienti (contatto, cliente)

contatti_referenti (contatto, referente)

contenuto_pacchi (ordine, pacco, specie_di_pianta, quantita)

disponibilita_fornitura (specie_di_pianta , fornitore)

fornitori (codice_fornitore, codice_fiscale, nome)

indirizzi_fornitori (indirizzo, fornitore)

ordini (id, cliente, data, aperto_or_finalizzato, indirizzo_spedizione, contatto)

pacchi (ordine, numero)

prezzi (specie_di_pianta, data_adozione, valore, data_termine*)

referenti (rivendita, nome, cognome, contatto_preferito*)

richieste_di_forniture (data, fornitore_scelto, specie_richiesta, quantita, pendente_si_no)

specie_di_piante (codice, nome_latino, nome_comune, interni_or_esterni, verde_or_fiorita, esotica_si_no, giacenza)

Con i seguenti vincoli di integrita refernziale:

appartenenza_ordini (ordine) ⊆ **ordini** (id)

appartennza_ordini (specie_di_pianta) ⊆ **specie_di_piante** (codice)

clienti (contatto_preferito) ⊆ **contatti** (valore)

colorazioni (specie_fiorita) ⊆ **specie_di_piante** (codice)

contatti_clienti (contatto) ⊆ **contatti** (valore)

contatti_clienti (cliente) ⊆ **clienti** (codice)

contatti_referenti (contatto) ⊆ **contatti** (valore)

contatti_referenti (cliente) ⊆ **clienti** (codice)

contenuto_pacchi (ordine, pacco) ⊆ **pacchi** (ordine, numero)

contenuto_pacchi (specie_di_pianta) ⊆ **specie_di_piante** (codice)

disponibilita_forniture (specie_di_pianta) ⊆ **specie_di_piante** (codice)

disponibilita_forniture (fornitore) ⊆ **fornitori** (codice_fornitore)

indirizzi_fornitori (fornitore) ⊆ **fornitori** (codice_fornitore)

ordini (clienti) ⊆ **clienti** (codice)

ordini (contatto) ⊆ **contatti** (valore)

pacchi (ordine) ⊆ **ordini** (id)

prezzi (specie_di_pianta) ⊆ **specie_di_piante** (codice)

referenti (rivendita) ⊆ **clienti** (codice)

referenti (contatto_preferito) ⊆ **contatti** (valore)

richieste_di_forniture (fornitore_scelto) ⊆ **fornitori** (codice_fornitore)

richieste_di_forniture (pianta_richiesta) ⊆ **specie_di_piante** (codice)

Normalizzazione del modello relazionale

Per dimostrare che lo schema prodotto soddisfa le forme 1NF, 2NF e 3NF è sufficiente mostrare come soddisfi la BCNF.

Per provare tale proprietà è necessario mostrare che per ogni dipendenza funzionale del tipo $X \rightarrow Y$, X è superchiave per la relazione.

Per la maggior parte delle relazioni, avendo esse pochi attributi, è facile dimostrarlo.

Esempio: relazione **indirizzi fornitori**

indirizzo → fornitore, ma fornitore $\nrightarrow$ indirizzo poiché ad esempio un fornitore può avere due indirizzi differenti

Per quanto riguarda le relazioni **specie_di_piante** e **fornitori** il discorso si amplia anche alla superchiave essendo presenti più chiavi candidate

Esempio: relazione **specie_di_piante**

codice → (nome_latino, nome_comune, … esotica_si_no, giacenza)

nome_latino → (codice, nome_comune, … esotica_si_no, giacenza)

(non esistono campi Y tali per cui codice → Y ma nome_latino $\nrightarrow$ Y)

Estendendo il ragionamento alle altre relazioni presenti nello schema è possibile concludere che lo schema è in BCNF.

5. Progettazione fisica

Utenti e privilegi

Dall'analisi della specifica del minimondo si ricavano le seguenti classsi d'utenti:

1. Cliente
   1.1. Cliente rivendita [RCS]
   1.2. Cliente privato [PCS]
2. Manager [MNG]
3. Operatore pacchi [OPC]
4. Adetto dipartimento magazzino [WHC]

Alle quali si aggiungono due ulteriori classi d'utenza necessarie per l'ingegnerizzazione del sistema:

5. Capo del personale [COS]
6. Non registrato [NRG]

Prima di effettuare una digressione sui privilegi delle classi d'utenza sulla base di dati è necessario introdurre due nuove relazioni:

utenti (<u>username</u>, password, ruolo, uuid)

utenze_clienti (<u>cliente</u>, utente)

con i seguenti vincoli di integrita referenziale:

utenze_clienti (cliente) ⊆ **clienti** (codice)

utenze_clienti (utente) ⊆ **utenti** (username)

Lo scopo della prima è quello di memorizzare le credenziali di accesso e la classe di appartenenza dei vari utenti fisici che si connettono al database. La seconda è invece la tabella pivot di una associazione 1:1 opzionale su cui si è adottata una traduzione non ottima.

Il suo scopo è quello di associare un utenza di un cliente alle sue informazioni effettive memorizzate nella base di dati.

In questa fase vengono introdotte anche altre 3 nuove operazioni:

1. NRG-03: permette di effettuare l'accesso al sistema
2. ALL-01: permette la modifica della password d'accesso
3. COS-01: permette di creare un'utenza dipendente (classi MNG, WHC, OPC, COS)

Anziché assegnare privilegi di lettura e/o scritture ad intere tabelle è stato preferito conferire il solo privilegio di esecuzione di alcune stored procedure a ciascuna classe d'utenza.

Nella tabella seguente vengono riassunti i privilegi d'esecuzione di ciascuna classe d'utenza presentata:

Classe d'utenza	**Privilegi (EXECUTE)**
Cliente rivendita	XCS-$$ RCS-$$ ALL-01
Cliente privato	XCS-$$ ALL-01
Manager	MNG-$$ ALL-01
Operatore pacchi	OPC-$$ ALL-01
Addetto dipartimento magazzino	WHC-$$ ALL-01
Capo del personale	COS-01 ALL-01
Non registrato	NRG-$$

n.b. I caratteri speciali $$ corrispondono ai numeri 01, 02 … presentati nella sezione relativa alle operazioni.

Strutture di memorizzazione

Tabella apparteneza_ordini		
Attributo	**Tipo di dato**	**Attributi**[2]
ordine	INT	PK. NN
specie_di_pianta	INT	PK, NN
quantita	INT	NN, UN

Tabella clienti		
Attributo	**Tipo di dato**	**Attributi**
codice	VARCHAR(16)	PK. NN
nome	VARCHAR(32)	NN
indirizzo_residenza	VARCHAR(64)	NN
indirizzo_fatturazione	VARCHAR(64)	
privato_or_rivendita	TINYINT	NN
contatto_preferito	VARCHAR(256)	

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

Tabella colorazioni		
Attributo	**Tipo di dato**	**Attributi**
specie_fiorita	INT	PK. NN
colore	VARCHAR(32)	PK, NN

Tabella contatti		
Attributo	**Tipo di dato**	**Attributi**
valore	VARCHAR(256)	PK. NN
tipo	TINYINT	NN

Tabella contatti_clienti		
Attributo	**Tipo di dato**	**Attributi**
contatto	VARCHAR(256)	PK, NN
cliente	VARCHAR(16)	NN

Tabella contatti_referenti		
Attributo	**Tipo di dato**	**Attributi**
contatto	VARCHAR(256)	PK. NN
referente	VARCHAR(16)	NN

Tabella contenuto_pacchi		
Attributo	**Tipo di dato**	**Attributi**
ordine	INT	PK. NN
pacco	INT	PK, NN, UN
specie_di_pianta	INT	PK, NN
quantita	INT	NN, UN

Tabella disponibilita_forniture		
Attributo	**Tipo di dato**	**Attributi**
specie_di_pianta	INT	PK. NN
fornitore	INT	PK, NN

Tabella fornitori		
Attributo	**Tipo di dato**	**Attributi**
codice_fornitore	INT	PK. NN, AI
codice_fiscale	VARCHAR(16)	NN, UQ
nome	VARCHAR(32)	NN

Tabella indirizzi_fornitori		
Attributo	**Tipo di dato**	**Attributi**
indirizzo	VARCHAR(64)	PK, NN
fornitore	INT	NN

Tabella ordini		
Attributo	**Tipo di dato**	**Attributi**
id	INT	PK. NN, AI
cliente	VARCHAR(16)	NN
data	DATETIME	NN
aperto_or_finalizzato	TINYINT	NN
indirizzo_spedizione	VARCHAR(64)	NN
contatto	VARCHAR(256)	NN

Tabella pacchi		
Attributo	**Tipo di dato**	**Attributi**
ordine	INT	PK. NN
numero	INT	PK, NN, UN

Tabella prezzi		
Attributo	**Tipo di dato**	**Attributi**
specie_di_pianta	INT	PK. NN
data_adozione	DATETIME	PK, NN
valore	DECIMAL(7,2)	NN, UN
data_termine	DATETIME	

Tabella referenti		
Attributo	**Tipo di dato**	**Attributi**
rivendita	VARCHAR(16)	PK, NN
nome	VARCHAR(32)	NN
cognome	VARCHAR(32)	NN
contatto_preferito	VARCHAR(256)	

Tabella richieste_di_forniture		
Attributo	**Tipo di dato**	**Attributi**
data	DATETIME	PK. NN
fornitore_scelto	INT	PK, NN
specie_richiesta	INT	PK, NN
quantita	INT	NN, UN
pendente_si_no	TINYINT	NN

Tabella specie_di_piante		
Attributo	**Tipo di dato**	**Attributi**
codice	INT	PK. NN, AI
nome_latino	VARCHAR(64)	NN, UQ
nome_comune	VARCHAR(64)	NN
interni_or_esterni	TINYINT	NN
verde_or_fiorita	TINYINT	NN
esotica_si_no	TINYINT	NN

giacenza	INT	NN, UN

Tabella utenti		
Attributo	**Tipo di dato**	**Attributi**
username	VARCHAR(128)	PK, NN
password	CHAR(128)	NN
ruolo	ENUM	NN
uuid	CHAR(36)	NN, UQ

Tabella utenze_clienti		
Attributo	**Tipo di dato**	**Attributi**
cliente	VARCHAR(16)	PK, NN
utente	VARCHAR(32)	NN

Indici

L'unico indice inserito manualmente in fase di modellazione è il seguente:

Tabella specie_di_piante	
Indice nome_comune_FULLTEXT	**Tipo[3]:**
nome_comune 1	FT

Tale indice è stato creato per velocizzare la ricerca nell'operazione XWM-01 ed inoltre è stato scelto il tipo FULLTEXT poiché tale si basa sull'utilizzo dell'operatore LIKE.

I seguenti indici sono invece stati autogenerati:

Tabella appartenenza_ordini	
Indice PRIMARY	**Tipo:**
ordine 1 specie_di_pianta 2	PRIMARY
Indice fk_appartenenza_ordini_specie_di_piante1_idx	**Tipo:**
specie_di_pianta 1	IDX

[3] IDX = index, UQ = unique, FT = full text, PR = primary.

Indice fk_appartenenza_ordini_ordini1_idx	**Tipo:**
ordine 1	IDX

Tabella clienti	
Indice PRIMARY	**Tipo:**
codice 1	PRIMARY
Indice fk_clienti_contatti1_idx	**Tipo:**
contatto_preferito 1	IDX

Tabella colorazioni	
Indice PRIMARY	**Tipo:**
specie_fiorita 1 colore 2	PRIMARY

Tabella contatti	
Indice PRIMARY	**Tipo:**
valore 1	PRIMARY

Tabella contatti_clienti	
Indice PRIMARY	**Tipo:**
contatto 1	PRIMARY
Indice fk_contatti_clienti_clienti1_idx	**Tipo:**
cliente 1	IDX

Tabella contatti_referenti	
Indice PRIMARY	**Tipo:**
contatto 1	PRIMARY
Indice fk_contatti_referenti_referenti1_idx	**Tipo:**
referente 1	IDX

Tabella contenuto_pacchi	
Indice PRIMARY	**Tipo:**
ordine 1 pacco 2 specie_di_pianta 3	PRIMARY
Indice fk_contenuto_pacchi_specie_di_piante1_idx	**Tipo:**
specie_di_pianta 1	IDX
Indice fk_contenuto_pacchi_pacchi1_idx	**Tipo:**
ordine 1 pacco 2	IDX

Tabella disponibilita_forniture	
Indice PRIMARY	**Tipo:**
specie_di_pianta 1 fornitore 2	PRIMARY
Indice fk_disponibilita_forniture_specie_di_piante1_idx	**Tipo:**
specie_di_pianta 1	IDX
Indice fk_ disponibilita_forniture_fornitori1_idx	**Tipo:**
fornitore 1	IDX

Tabella fornitori	
Indice PRIMARY	**Tipo:**
codice_fornitore 1	PRIMARY
Indice codice_fiscale_UNIQUE	**Tipo:**
codice_fiscale 1	UQ

Tabella indirizzi_fornitori	
Indice PRIMARY	**Tipo:**
indirizzo 1	PRIMARY
Indice fk_indirizzi_fornitori_fornitori1_idx	**Tipo:**
fornitore 1	IDX

Tabella ordini	
Indice PRIMARY	**Tipo:**
id 1	PRIMARY
Indice fk_ordini_contatti1_idx	**Tipo:**
contatto 1	IDX
Indice fk_ ordini_clienti1_idx	**Tipo:**
cliente 1	IDX

Tabella pacchi	
Indice PRIMARY	**Tipo:**
ordine 1 numero 2	PRIMARY
Indice fk_pacchi_ordini1_idx	**Tipo:**
ordine 1	IDX

Tabella prezzi	
Indice PRIMARY	**Tipo:**
specie_di_pianta 1 data_adozione 2	PRIMARY
Indice fk_prezzi_specie_di_piante1_idx	**Tipo:**
specie_di_pianta 1	IDX

Tabella referenti	
Indice PRIMARY	**Tipo:**
rivendita 1	PRIMARY
Indice fk_referenti_contatti1_idx	**Tipo:**
contatto_preferito 1	IDX

Tabella richieste_di_forniture	
Indice PRIMARY	**Tipo:**
data 1 fornitore_scelto 2 specie_richiesta 3	PRIMARY
Indice fk_richieste_di_forniture_specie_di_piante1_idx	**Tipo:**
specie_richiesta 1	IDX
Indice fk_ richieste_di_forniture_fornitori1_idx	**Tipo:**
fornitore_scelto 1	IDX

Tabella referenti	
Indice PRIMARY	**Tipo:**
rivendita 1	PRIMARY
Indice fk_referenti_contatti1_idx	**Tipo:**
contatto_preferito 1	IDX

Tabella specie_di_piante	
Indice PRIMARY	**Tipo:**
codice 1	PRIMARY
Indice nome_latino_UNIQUE	**Tipo:**
nome_latino 1	UQ

Tabella utenti	
Indice PRIMARY	**Tipo:**
username 1	PRIMARY
Indice uuid_UNIQUE	**Tipo:**
uuid 1	UQ

Tabella utenze_clienti	
Indice PRIMARY	**Tipo:**
cliente 1	PRIMARY
Indice fk_utenze_clienti_utenti1_idx	**Tipo:**
utente 1	IDX

Il DBMS ha quindi autogenerato:

1. degli indici di tipo PRIMARY per ciascuna chiave primaria
2. degli indici di tipo INDEX per ciascuna foreign key per velocizzare le operazioni di join.
3. degli indici di tipo UNIQUE per ciascun campo con attributo UNIQUE presente nello schema, anche in questo caso per facilitare le operazioni di ricerca.

Trigger

Tabella utenti

```
CREATE DEFINER = CURRENT_USER TRIGGER
`verdesrl`.`utenti_BEFORE_INSERT`
BEFORE INSERT ON `utenti` FOR EACH ROW
BEGIN
	IF INSTR(NEW.`username`, " ") > 0 THEN
		SIGNAL SQLSTATE '45004'
		SET MESSAGE_TEXT = "Username must not contains spaces";
	END IF;
END$$
```

Tabella specie_di_piante

```
CREATE DEFINER = CURRENT_USER TRIGGER
`verdesrl`.`specie_di_piante_BEFORE_INSERT`
BEFORE INSERT ON `specie_di_piante` FOR EACH ROW
BEGIN
	IF NEW.interni_or_esterni <> 1  AND NEW.interni_or_esterni <> 0 THEN
		SIGNAL SQLSTATE '45022'
		SET MESSAGE_TEXT = "Types of species allowed: 1 -> Indoor ~ 0 -> Outdoor";
	 END IF;
```

```
    IF NEW.verde_or_fiorita <> 1  AND NEW.verde_or_fiorita <> 0 THEN
        SIGNAL SQLSTATE '45022'
        SET MESSAGE_TEXT = "Types of species allowed: 1 -> Green ~ 0 -> Flowery";
    END IF;
    IF NEW.esotica_si_no <> 1  AND NEW.esotica_si_no <> 0 THEN
        SIGNAL SQLSTATE '45022'
        SET MESSAGE_TEXT = "Types of species allowed: 1 -> Exotic ~ 0 -> Not exotic";
     END IF;
END$$
```

Tabella clienti

```
CREATE DEFINER = CURRENT_USER TRIGGER
`verdesrl`.`clienti_BEFORE_INSERT`
BEFORE INSERT ON `clienti` FOR EACH ROW
BEGIN
    IF NEW.codice REGEXP "[A-Za-z]{6}[0-9]{2}[A-Za-z]{1}[0-9]{2}[A-Za-z]{1}[0-9]{3}[A-Za-z]{1}$" THEN
        IF NEW.privato_or_rivendita <> 1 THEN
            SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = "Mismatch between code type and customer type";
        END IF;
    ELSEIF NEW.codice REGEXP "[0-9]{11}$" THEN
        IF NEW.privato_or_rivendita <> 0 THEN
            SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = "Mismatch between code type and customer type";
        END IF;
    ELSE
        SIGNAL SQLSTATE '45001' SET MESSAGE_TEXT = "Wrong code inserted";
    END IF;
END$$

CREATE DEFINER = CURRENT_USER TRIGGER
```

```
`verdesrl`.`clienti_BEFORE_UPDATE`
BEFORE UPDATE ON `clienti` FOR EACH ROW
BEGIN
	DECLARE var_proprietario VARCHAR(16);

	IF NEW.contatto_preferito IS NOT NULL AND
		NEW.contatto_preferito <> OLD.contatto_preferito THEN
		SELECT	cliente
		FROM	contatti_clienti
		WHERE	contatto = NEW.contatto_preferito
		INTO	var_proprietario;

		IF NEW.codice <> var_proprietario THEN
			SIGNAL SQLSTATE '45009'
			SET MESSAGE_TEXT = "Selected customer does not own this contact";
		END IF;
	END IF;
END$$
```

Tabella referenti

```
CREATE DEFINER = CURRENT_USER TRIGGER
`verdesrl`.`referenti_BEFORE_UPDATE`
BEFORE UPDATE ON `referenti` FOR EACH ROW
BEGIN
	DECLARE var_proprietario VARCHAR(16);

	IF NEW.contatto_preferito IS NOT NULL AND
		NEW.contatto_preferito <> OLD.contatto_preferito THEN
		SELECT	referente
		FROM	contatti_referenti
		WHERE	contatto = NEW.contatto_preferito
		INTO	var_proprietario;
```

```
            IF NEW.rivendita <> var_proprietario THEN
                SIGNAL SQLSTATE '45009'
                SET MESSAGE_TEXT = "Selected referent does not own this contact";
            END IF;
        END IF;
END$$
```

Tabella pacchi

```
CREATE DEFINER = CURRENT_USER TRIGGER
`verdesrl`.`pacchi_BEFORE_INSERT`
BEFORE INSERT ON `pacchi` FOR EACH ROW
BEGIN
        DECLARE var_stato TINYINT;

        SELECT      aperto_or_finalizzato
        FROM        ordini
        WHERE       id = NEW.ordine
        INTO        var_stato;

        IF var_stato = 1 THEN
            SIGNAL SQLSTATE '45034' SET MESSAGE_TEXT = "Order not yet finalized";
        END IF;
END$$
```

Tabella fornitori

```
CREATE DEFINER = CURRENT_USER TRIGGER
`verdesrl`.`fornitori_BEFORE_INSERT`
BEFORE INSERT ON `fornitori` FOR EACH ROW
BEGIN
        IF NEW.codice_fiscale REGEXP "/^[A-Za-z]{6}[0-9]{2}[A-Za-z]{1}[0-9]{2}[A-Za-z]{1}[0-9]{3}[A-Za-z]{1}$/" THEN
```

```
            SIGNAL SQLSTATE '45023'
            SET MESSAGE_TEXT = "Fiscal code expression not valid";
    END IF;
END$$

CREATE DEFINER = CURRENT_USER TRIGGER
`verdesrl`.`fornitori_BEFORE_INSERT_1`
BEFORE INSERT ON `fornitori` FOR EACH ROW
BEGIN
    SET NEW.codice_fiscale = UPPER(NEW.codice_fiscale);
END$$
```

Tabella richieste_di_forniture

```
CREATE DEFINER = CURRENT_USER TRIGGER
`verdesrl`.`richieste_di_forniture_BEFORE_INSERT`
BEFORE INSERT ON `richieste_di_forniture` FOR EACH ROW
BEGIN
    DECLARE var_verifica_disponibilita INT;

    SELECT      COUNT(*)
    FROM        disponibilita_forniture
    WHERE       fornitore = NEW.fornitore_scelto AND
                specie_di_pianta = NEW.specie_richiesta
    INTO        var_verifica_disponibilita;

    IF var_verifica_disponibilita = 0 THEN
            SIGNAL SQLSTATE '45024'
            SET MESSAGE_TEXT = "Request species not available";
    END IF;
END$$
```

Tabella colorazioni

```
CREATE DEFINER = CURRENT_USER TRIGGER
`verdesrl`.`colorazioni_BEFORE_INSERT`
BEFORE INSERT ON `colorazioni` FOR EACH ROW
BEGIN
	DECLARE var_verde_or_fiorita TINYINT;

	SELECT	verde_or_fiorita
	FROM	specie_di_piante
	WHERE	codice = NEW.specie_fiorita
	INTO	var_verde_or_fiorita;

	IF var_verde_or_fiorita = 1 THEN
		SIGNAL SQLSTATE '45019'
		SET MESSAGE_TEXT = "Green species cannot be colorful";
	END IF;
END$
```

Tabella contenuto_pacchi

```
CREATE DEFINER = CURRENT_USER TRIGGER
`verdesrl`.`contenuto_pacchi_BEFORE_INSERT`
BEFORE INSERT ON `contenuto_pacchi` FOR EACH ROW
BEGIN
	DECLARE var_verifica_appartenenza INT;

	SELECT	COUNT(*)
	FROM	appartenenza_ordini
	WHERE	ordine = NEW.ordine AND
		specie_di_pianta = NEW.specie_di_pianta
	INTO	var_verifica_appartenenza;

	IF var_verifica_appartenenza = 0 THEN
		SIGNAL SQLSTATE '45020'
```

```
            SET MESSAGE_TEXT = "Species does not belong to the order";
        END IF;
END$$


CREATE DEFINER = CURRENT_USER TRIGGER
`verdesrl`.`contenuto_pacchi_BEFORE_INSERT_1`
BEFORE INSERT ON `contenuto_pacchi` FOR EACH ROW
BEGIN
        DECLARE var_quantita_ordine INT;
        DECLARE var_gia_impacchettate INT;

        SELECT      quantita
        FROM        appartenenza_ordini
        WHERE       specie_di_pianta = NEW.specie_di_pianta AND
                    ordine = NEW.ordine
        INTO        var_quantita_ordine;

        SELECT      SUM(quantita)
        FROM        contenuto_pacchi
        WHERE       ordine = NEW.ordine AND
                    specie_di_pianta = NEW.specie_di_pianta
        INTO        var_gia_impacchettate;

        IF var_gia_impacchettate IS NULL THEN
            SET var_gia_impacchettate = 0;
        END IF;

        IF NEW.quantita > var_quantita_ordine - var_gia_impacchettate THEN
            SIGNAL SQLSTATE '45015'
            SET MESSAGE_TEXT = "Selected quantity greater than that not processed yet";
        END IF;
```

```
END$$
```

Tabella appartenenza_ordini

```
CREATE DEFINER = CURRENT_USER TRIGGER
`verdesrl`.`appartenenza_ordini_BEFORE_INSERT`
BEFORE INSERT ON `appartenenza_ordini` FOR EACH ROW
BEGIN
	DECLARE var_giacenza INT UNSIGNED;

	SELECT	giacenza
	FROM	specie_di_piante
	WHERE	codice = NEW.specie_di_pianta
	INTO	var_giacenza;

	IF NEW.quantita > var_giacenza THEN
		SIGNAL SQLSTATE '45006'
		SET MESSAGE_TEXT = "Selected quantity is higher than current stock";
	END IF;
END$$

CREATE DEFINER = CURRENT_USER TRIGGER
`verdesrl`.`appartenenza_ordini_BEFORE_INSERT_1`
BEFORE INSERT ON `appartenenza_ordini` FOR EACH ROW
BEGIN
	DECLARE var_stato TINYINT;

	SELECT	aperto_or_finalizzato
	FROM	ordini
	WHERE	id = NEW.ordine
	INTO	var_stato;

	IF var_stato = 0 THEN
```

```
            SIGNAL SQLSTATE '45007'
            SET MESSAGE_TEXT = "It is not possible to modify an order already closed";
        END IF;
END$$

CREATE DEFINER = CURRENT_USER TRIGGER
`verdesrl`.`appartenenza_ordini_AFTER_INSERT`
AFTER INSERT ON `appartenenza_ordini` FOR EACH ROW
BEGIN
        UPDATE      specie_di_piante
        SET         giacenza = giacenza - NEW.quantita
        WHERE       codice = NEW.specie_di_pianta;
END$$

CREATE DEFINER = CURRENT_USER TRIGGER
`verdesrl`.`appartenenza_ordini_BEFORE_UPDATE`
BEFORE UPDATE ON `appartenenza_ordini` FOR EACH ROW
BEGIN
        DECLARE var_giacenza INT UNSIGNED;

        SELECT      giacenza
        FROM        specie_di_piante
        WHERE       codice = NEW.specie_di_pianta
        INTO        var_giacenza;

        IF CAST(NEW.quantita AS SIGNED) - CAST(OLD.quantita AS SIGNED) > var_giacenza
        THEN
            SIGNAL SQLSTATE '45006'
            SET MESSAGE_TEXT = "Selected quantity is higher than current stock";
         END IF;
END$$
```

```
CREATE DEFINER = CURRENT_USER TRIGGER
`verdesrl`.`appartenenza_ordini_BEFORE_UPDATE_1`
BEFORE UPDATE ON `appartenenza_ordini` FOR EACH ROW
BEGIN
	DECLARE var_stato TINYINT;

	SELECT	aperto_or_finalizzato
	FROM	ordini
	WHERE	id = NEW.ordine
	INTO	var_stato;

	IF var_stato = 0 THEN
		SIGNAL SQLSTATE '45007'
		SET MESSAGE_TEXT = "It is not possible to modify an order already closed";
	END IF;
END$$

CREATE DEFINER = CURRENT_USER TRIGGER
`verdesrl`.`appartenenza_ordini_AFTER_UPDATE`
AFTER UPDATE ON `appartenenza_ordini` FOR EACH ROW
BEGIN
	IF NEW.quantita > OLD.quantita THEN
		UPDATE	specie_di_piante
		SET giacenza = giacenza - (NEW.quantita - OLD.quantita)
		WHERE codice = NEW.specie_di_pianta;
	ELSE
		UPDATE	specie_di_piante
		SET giacenza = giacenza + (OLD.quantita - NEW.quantita)
		WHERE codice = NEW.specie_di_pianta;
	END IF;
END$$
```

```
CREATE DEFINER = CURRENT_USER TRIGGER
`verdesrl`.`appartenenza_ordini_BEFORE_DELETE`
BEFORE DELETE ON `appartenenza_ordini` FOR EACH ROW
BEGIN
	DECLARE var_stato TINYINT;

	SELECT	aperto_or_finalizzato
	FROM	ordini
	WHERE	id = OLD.ordine
	INTO	var_stato;

	IF var_stato = 0 THEN
		SIGNAL SQLSTATE '45007'
		SET MESSAGE_TEXT = "It is not possible to modify an order already closed";
	END IF;
END$$

CREATE DEFINER = CURRENT_USER TRIGGER
`verdesrl`.`appartenenza_ordini_AFTER_DELETE`
AFTER DELETE ON `appartenenza_ordini` FOR EACH ROW
BEGIN
	DECLARE var_stato TINYINT;

	SELECT	aperto_or_finalizzato
	FROM	ordini
	WHERE	id = OLD.ordine
	INTO	var_stato;

	IF var_stato = 1 THEN
		UPDATE	specie_di_piante
		SET	giacenza = giacenza + OLD.quantita
		WHERE	codice = OLD.specie_di_pianta;
```

```
    END IF;
END$$
```

Eventi

Entrambi gli eventi non sono attivati in fase di configurazione bensì vengono eseguiti una volta all'anno per eliminare dati non più utili al sistema.

Per quanto riguarda il secondo evento è stato supposto che lo scarico delle forniture avvenga regolarmente il lunedì sera per cui tale evento viene eseguito ogni martedì a mezzanotte.

Il codice di tali eventi è qui riportato:

```
SET GLOBAL event_scheduler = ON;

DELIMITER $$

CREATE EVENT IF NOT EXISTS pulizia_storico_dati
ON SCHEDULE EVERY 1 YEAR
STARTS "2021-01-01 00:00:00"
ON COMPLETION PRESERVE
COMMENT "Eliminazione dati relativi ad ordini prezzi e richieste di fornitura memorizzati da piu' di due anni "
DO BEGIN
    DELETE FROM `verdesrl`.`ordini`
    WHERE `data` < (NOW() - INTERVAL 2 YEAR);

    DELETE FROM `verdesrl`.`richieste_di_forniture`
    WHERE `data` < (NOW() - INTERVAL 2 YEAR);

    DELETE FROM `verdesrl`.`prezzi`
    WHERE `data_termine` < (NOW() - INTERVAL 2 YEAR);
END $$

DELIMITER ;
```

```
CREATE EVENT IF NOT EXISTS aggiornamento_giacenza
ON SCHEDULE EVERY 1 WEEK
STARTS "2020-04-21 00:00:00"
ON COMPLETION PRESERVE
COMMENT "Aggiornamento giacenza al fronte di richieste di forniture"
DO BEGIN

    DECLARE var_specie_c INT;
    DECLARE var_quantita_c INT;
    DECLARE var_data_agg DATETIME;
    DECLARE done INT DEFAULT FALSE;
    DECLARE cur CURSOR FOR
        SELECT      `specie_richiesta`, `quantita`
        FROM        `richieste_di_forniture`
        WHERE       `pendente_si_no` = 1;
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

    DROP TEMPORARY TABLE IF EXISTS da_aggiornare;
    CREATE TEMPORARY TABLE da_aggiornare (
        `specie_di_pianta`      INT,
        `quantita_richiesta`    INT UNSIGNED
    );

    SET var_data_agg = now();

    OPEN cur;
    read_loop: LOOP
        FETCH cur INTO var_specie_c, var_quantita_c;
        IF done THEN
            LEAVE read_loop;
        END IF;
```

```
            UPDATE      `specie_di_piante`
            SET         `giacenza` = `giacenza` + var_quantita_c
            WHERE       `codice` = var_specie_c;
        END LOOP;
        CLOSE cur;
        DROP TEMPORARY TABLE da_aggiornare;

        UPDATE      `richieste_di_forniture`
        SET         `pendente_si_no` = 0
        WHERE       `data` <= var_data_agg;

END $$
DELIMITER ;
```

Viste

Non sono state implementate viste

Stored Procedures e transazioni

XCS-01

Il livello di isolamento scelto e' SERIALIZABLE per via dell'utilizzo della select max per ottenere l'id autogereato.

Non e' stato possibile utilizzare la funzione LAST_INSERT_ID() poiche' dalla documentazione si evince che "The ID that was generated is maintained in the server on a per-connection basis".

Per cui dato che la connessione e' la medesima per ciascun utente afferente ad una certa classe si e' preferita questa scelta alternativa.

```
CREATE PROCEDURE `crea_ordine` (
        IN var_cliente      VARCHAR(16),
        IN var_ind_sped     VARCHAR(64),
        IN var_contatto     VARCHAR(256),
        IN var_specie       INT,
        IN var_quantita     INT,
        OUT var_id          INT
)
```

```
BEGIN
	DECLARE var_errno INT;
	DECLARE var_verifica_proprietario INT;

	DECLARE EXIT HANDLER FOR SQLEXCEPTION
	BEGIN
		ROLLBACK;
		GET DIAGNOSTICS CONDITION 1 var_errno = MYSQL_ERRNO;

		IF var_errno = 1452 THEN
			SIGNAL SQLSTATE '45014'
			SET MESSAGE_TEXT = "Selected species does not exists";
		ELSE
			RESIGNAL;
		END IF;
	END;

	DROP TABLE IF EXISTS `verifica_proprietario`;
	CREATE TEMPORARY TABLE `verifica_proprietario` (
		`Contatto`	VARCHAR(256),
		`Cliente`	CHAR(16),
		`Referente`	CHAR(11)
	);

	SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
	START TRANSACTION;

		IF var_ind_sped IS NULL THEN
			SELECT	indirizzo_residenza
			FROM	clienti
			WHERE	codice = var_cliente
			INTO	var_ind_sped;
```

```
END IF;

IF var_contatto IS NULL THEN
        SELECT      contatto_preferito
        FROM        clienti
        WHERE       codice = var_cliente
        INTO        var_contatto;

        IF var_contatto IS NULL THEN
                SIGNAL SQLSTATE '45002'
                SET MESSAGE_TEXT  = "Favourite contact not set for current user";
        END IF;
ELSE
        INSERT INTO `verifica_proprietario`
        SELECT      contatti.valore                 AS `Contatto`,
                    contatti_clienti.cliente        AS `Cliente`,
                    contatti_referenti.referente    AS `Referente`
        FROM        contatti        LEFT JOIN contatti_clienti
                                    ON contatti_clienti.contatto = valore
                                    LEFT JOIN contatti_referenti
                                    ON contatti_referenti.contatto = valore
        WHERE       contatti.valore = var_contatto;

        SELECT      COUNT(*)
        FROM        `verifica_proprietario`
        WHERE       (`Cliente` = var_cliente OR `Referente` = var_cliente)
        INTO  var_verifica_proprietario;

        IF var_verifica_proprietario = 0 THEN
                SIGNAL SQLSTATE '45009'
                SET MESSAGE_TEXT = "Neither customer nor referent
                                        (if exists) own this contact";
```

```
            END IF;
        END IF;

        DROP TEMPORARY TABLE `verifica_proprietario`;

        INSERT INTO ordini (`cliente`, `data`, `aperto_or_finalizzato`,
        `indirizzo_spedizione`, `contatto`)
            VALUES (var_cliente, NOW(), 1, var_ind_sped, var_contatto);

        SELECT MAX(id) FROM ordini INTO var_id;

        INSERT INTO appartenenza_ordini (`specie_di_pianta`, `ordine`, `quantita`)
            VALUES (var_specie, var_id, var_quantita);

    COMMIT;
END$$
```

XCS-02

Il livello di isolamento scelto e' REPEATABLE READ, al fine di garantire che la giacenza della specie (che viene letta in un trigger BI) non possa essere modificata durante l'esecuzione della transazione.

```
CREATE PROCEDURE `aggiungi_specie_ad_ordine_esistente` (
    IN var_cliente VARCHAR(16),
    IN var_specie  INT,
    IN var_ordine  INT,
    IN var_quantita INT
)
BEGIN
    DECLARE var_errno INT;

    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
```

```
            ROLLBACK;
            GET DIAGNOSTICS CONDITION 1 var_errno = MYSQL_ERRNO;

            IF var_errno = 1062 THEN
                SIGNAL SQLSTATE '45008'
                SET MESSAGE_TEXT = "Species already in order";
            ELSEIF var_errno = 1452 THEN
                SIGNAL SQLSTATE '45014'
                SET MESSAGE_TEXT = "Species does not exists";
            ELSE
                RESIGNAL;
            END IF;
        END;

        START TRANSACTION;

            IF verifica_proprietario(var_ordine, var_cliente) THEN
                INSERT INTO appartenenza_ordini (`specie_di_pianta`, `ordine`, `quantita`)
                    VALUES (var_specie, var_ordine, var_quantita);
            ELSE
                SIGNAL SQLSTATE '45005'
                SET MESSAGE_TEXT = "Only owners can make changes to an order";
            END IF;

        COMMIT;
END$$
```

XCS-03

Il livello di isolamento scelto è SERIALIZABLE per via dell'utilizzo della funzione COUNT(*) per contare le specie ancora appartenenti all'ordine. Ai fini del corretto funzionamento è necessario che non vi siano inserimenti fantasma fino al termine della transazione.

```
CREATE PROCEDURE `rimuovi_specie_da_ordine` (
    IN var_cliente                  VARCHAR(16),
```

```
        IN var_specie                   INT,
        IN var_ordine                   INT,
        OUT var_ordine_eliminato_si_no  INT,
        OUT var_eliminazione_eff        INT
)
BEGIN
        DECLARE var_capienza_ordine INT;

        DECLARE EXIT HANDLER FOR SQLEXCEPTION
        BEGIN
                ROLLBACK;
                RESIGNAL;
        END;

        SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
        START TRANSACTION;

                IF verifica_proprietario(var_ordine, var_cliente) THEN
                        SELECT      COUNT(*)
                        FROM        appartenenza_ordini
                        WHERE       specie_di_pianta = var_specie AND
                                    ordine = var_ordine
                        INTO        var_eliminazione_eff;

                        IF var_eliminazione_eff = 1 THEN
                                DELETE
                                FROM        appartenenza_ordini
                                WHERE       specie_di_pianta = var_specie AND
                                            ordine = var_ordine;

                                SELECT      COUNT(*)
                                FROM        appartenenza_ordini
```

```
                    WHERE       ordine = var_ordine
                    INTO        var_capienza_ordine;

                    IF var_capienza_ordine = 0 THEN
                        DELETE FROM ordini WHERE id = var_ordine;
                        SET var_ordine_eliminato_si_no = 1;
                    ELSE
                        SET var_ordine_eliminato_si_no = 0;
                    END IF;
                END IF;
            ELSE
                SIGNAL SQLSTATE '45005'
                SET MESSAGE_TEXT = "Only owners can make changes to an order";
            END IF;

    COMMIT;
END$$
```

XCS-04

Il livello di isolamento scelto e' REPEATABLE READ, al fine di garantire che la giacenza della specie (che viene letta in un trigger BU) non possa essere modificata durante l'esecuzione della transazione.

```
CREATE PROCEDURE `modifica_ordine` (
    IN var_cliente              VARCHAR(16),
    IN var_specie               INT,
    IN var_ordine               INT,
    IN var_quantita             INT,
    OUT var_aggiornamento_eff INT
)
BEGIN
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        ROLLBACK;
```

```
            RESIGNAL;
        END;

        START TRANSACTION;

            IF verifica_proprietario(var_ordine, var_cliente) THEN
                SELECT      COUNT(*)
                FROM        appartenenza_ordini
                WHERE       specie_di_pianta = var_specie AND
                            ordine = var_ordine
                INTO        var_aggiornamento_eff;
                IF var_aggiornamento_eff = 1 THEN
                    UPDATE      appartenenza_ordini
                    SET         quantita = var_quantita
                    WHERE       specie_di_pianta = var_specie AND
                                ordine = var_ordine;
                END IF;
            ELSE
                SIGNAL SQLSTATE '45005'
                SET MESSAGE_TEXT = "Only owners can make changes to an order";
            END IF;

        COMMIT;
END$$
```

XCS-05

```
CREATE PROCEDURE `finalizza_ordine` (
        IN var_cliente VARCHAR(16),
        IN var_ordine INT
)
BEGIN
        IF verifica_proprietario(var_ordine, var_cliente) THEN
```

```
        UPDATE      ordini
        SET         aperto_or_finalizzato = 0
        WHERE       id = var_ordine;
    ELSE
        SIGNAL SQLSTATE '45005'
        SET MESSAGE_TEXT = "Only owners can make changes to an order";
    END IF;
END$$
```

XCS-06

Il livello di isolamento scelto e' SERIALIZABLE al fine di presentare uno snapshot affidabile dell'ordine selezionato anche nel caso in cui esso sia ancora aperto e quindi potrebbe mutare concorrentemente alla produzione del report con eventuali inserimenti.

Il campo nome all'interno della tabella temporanea dettagli_specie tiene conto della somma di più lunghezze in particolare si ha 131 = 64 (nome_comune) + 64 (nome_latino) + 3 (caratteri extra).

```
CREATE PROCEDURE `report_ordine` (
    IN var_cliente VARCHAR(16),
    IN var_ordine INT
)
BEGIN
    DECLARE var_codice_cursor INT;
    DECLARE var_verifica_esistenza INT DEFAULT 0;
    DECLARE var_data_ordine DATETIME;
    DECLARE var_numero_piante INT;

    DECLARE done INT DEFAULT FALSE;
    DECLARE cur CURSOR FOR
        SELECT      codice
        FROM        specie_di_piante    INNER JOIN appartenenza_ordini
                                        ON codice = specie_di_pianta
        WHERE       ordine = var_ordine;
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;
```

```
DECLARE EXIT HANDLER FOR SQLEXCEPTION
BEGIN
	ROLLBACK;
	RESIGNAL;
END;

DROP TEMPORARY TABLE IF EXISTS dettagli_specie;
CREATE TEMPORARY TABLE dettagli_specie (
	`Code` INT,
	`Name` VARCHAR(131),
	`Unit_price_(E)` DECIMAL(7, 2),
	`Required_amount` INT,
	`Cumulative_price_(E)` DECIMAL(8, 2),
	PRIMARY KEY (`Code`)
);

SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
START TRANSACTION;

	IF NOT verifica_proprietario(var_ordine, var_cliente) THEN
		SIGNAL SQLSTATE '45005'
		SET MESSAGE_TEXT = "Only owners can view order reports";
	END IF;

	SELECT	COUNT(*)
	FROM	ordini
	WHERE	id = var_ordine
	INTO	var_verifica_esistenza;

	IF var_verifica_esistenza = 0 THEN
		SIGNAL SQLSTATE '45016'
		SET MESSAGE_TEXT = "Not existent order";
```

```
END IF;

SELECT      `data`
FROM        `ordini`
WHERE       `id` = var_ordine
INTO        var_data_ordine;

OPEN cur;
read_loop: LOOP
        FETCH cur INTO var_codice_cursor;
        IF done THEN
                LEAVE read_loop;
        END IF;

        INSERT INTO dettagli_specie
        SELECT      specie_di_piante.codice AS `Code`,
                    CONCAT(specie_di_piante.nome_comune, " (",
                    specie_di_piante.nome_latino, + ")") AS `Name`,
                    prezzi.valore AS `Unit_price_(E)`,
                    appartenenza_ordini.quantita  AS `Required_amount`,
                    prezzi.valore * appartenenza_ordini.quantita
                            AS      `Cumulative_price_(E)`
        FROM        specie_di_piante INNER JOIN prezzi
                    ON specie_di_piante.codice = prezzi.specie_di_pianta
                    INNER JOIN appartenenza_ordini ON
                    specie_di_piante.codice=appartenenza_ordini.specie_di_pianta
        WHERE       specie_di_piante.codice = var_codice_cursor  AND
                    data_adozione <= var_data_ordine
        ORDER BY    data_adozione DESC
        LIMIT 1;
END LOOP;
CLOSE cur;
```

```
        SELECT      `ordini`.`id` AS `ID`,
                    DATE(`ordini`.`data`) AS `Date`,
                    `ordini`.`indirizzo_spedizione` AS `Shipping_address`,
                    `ordini`.`contatto` AS `Chosen_contact`,
                    CASE
                        WHEN `ordini`.`aperto_or_finalizzato` = 1 THEN "Open"
                        ELSE "Finalized"
                    END AS `Status`,
                    CASE
                        WHEN `clienti`.`privato_or_rivendita` = 1 THEN
                        "Not expected"
                        ELSE concat(`referenti`.`nome`, " ", `referenti`.`cognome`)
                    END AS `Referent`
        FROM        `ordini` INNER JOIN clienti` ON `ordini`.`cliente` = `clienti`.`codice`
                    LEFT JOIN `referenti` ON `clienti`.`codice` = `referenti`.`rivendita`
        WHERE           `id` = var_ordine;

        SELECT * FROM dettagli_specie;

        SELECT      SUM(`Required_amount`) AS `Number_of_plants`,
                    SUM(`Cumulative_price_(E)`) AS `Total_spending`
        FROM        `dettagli_specie`;

        DROP TEMPORARY TABLE dettagli_specie;
    COMMIT;
END$$
```

XCS-07

```
CREATE PROCEDURE `modifica_residenza` (
    IN var_cliente      VARCHAR(16),
    IN var_residenza    VARCHAR(64)
```

```
)
BEGIN
        UPDATE      clienti
        SET         indirizzo_residenza = var_residenza
        WHERE       codice = var_cliente;
END$$
```

XCS-08

```
CREATE PROCEDURE `modifica_fatturazione` (
        IN var_cliente          VARCHAR(16),
        IN var_fatturazione     VARCHAR(64)
)
BEGIN
        UPDATE      clienti
        SET         indirizzo_fatturazione = var_fatturazione
        WHERE       codice = var_cliente;
END$$
```

XCS-09

Il livello di isolamento scelto e' READ COMMITTED poichè sufficiente per restituire una lista consistente di contatti realmente inseriti (un contatto già inserito non può essere modificato).

```
CREATE PROCEDURE `aggiungi_contatto_cliente` (
        IN var_cliente          VARCHAR(16),
        IN var_contatto         VARCHAR(256),
        IN var_tipo             CHAR(16)
)
BEGIN
        DECLARE var_errno INT;

        DECLARE EXIT HANDLER FOR SQLEXCEPTION
        BEGIN
                ROLLBACK;
```

```
        GET DIAGNOSTICS CONDITION 1 var_errno = MYSQL_ERRNO;

        IF var_errno = 1062 THEN
            SIGNAL SQLSTATE '45010'
            SET MESSAGE_TEXT = "Contact already inserted";
        ELSE
            RESIGNAL;
        END IF;
    END;

    SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
    START TRANSACTION;

        INSERT INTO contatti (`valore`, `tipo`) VALUES (var_contatto, var_tipo);
        INSERT INTO contatti_clienti (`contatto`, `cliente`)
            VALUES (var_contatto, var_cliente);

        SELECT    contatti.valore AS `Contact`,
                  contatti.tipo   AS `Type`
        FROM      contatti        INNER JOIN contatti_clienti
                                  ON contatti.valore = contatti_clienti.contatto
        WHERE     cliente = var_cliente;

    COMMIT;
END$$
```

XCS-10

```
CREATE PROCEDURE `rimuovi_contatto_cliente` (
    IN var_cliente VARCHAR(16),
    IN var_contatto VARCHAR(256)
)
```

```
BEGIN
	DECLARE var_verifica_possesso INT;
	DECLARE var_errno INT;

	DECLARE EXIT HANDLER FOR SQLEXCEPTION
	BEGIN

		GET DIAGNOSTICS CONDITION 1 var_errno = MYSQL_ERRNO;

		IF var_errno = 1451 THEN
			SIGNAL SQLSTATE '45012'
			SET MESSAGE_TEXT = "Cannot remove the contact because it is associated
					to an order";
		ELSE
			RESIGNAL;
		END IF;
	END;

	SELECT	COUNT(*)
	FROM	contatti_clienti
	WHERE	cliente = var_cliente AND
		contatto = var_contatto
	INTO	var_verifica_possesso;

	IF var_verifica_possesso = 0 THEN
		SIGNAL SQLSTATE '45009'
		SET MESSAGE_TEXT = "Selected customer does not own this contact";
	ELSE
		DELETE FROM contatti WHERE valore = var_contatto;
	END IF;
END$$
```

XCS-11

```
CREATE PROCEDURE `modifica_contatto_preferito_cliente` (
        IN var_cliente VARCHAR(16),
        IN var_contatto VARCHAR(256)
)
BEGIN
        DECLARE var_errno INT;
        DECLARE EXIT HANDLER FOR SQLEXCEPTION
        BEGIN
                GET DIAGNOSTICS CONDITION 1 var_errno = MYSQL_ERRNO;

                IF var_errno = 1452 THEN
                        SIGNAL SQLSTATE '45013'
                        SET MESSAGE_TEXT = "Cannot set this contact as favourite one because it
                                                is not owned yet";
                ELSE
                        RESIGNAL;
                END IF;
        END;

        UPDATE      clienti
        SET         contatto_preferito = var_contatto
        WHERE       codice = var_cliente;
END$$
```

XCS-12

```
CREATE PROCEDURE `visualizza_contatti_cliente` (IN var_cliente VARCHAR(16))
BEGIN
        SELECT      contatti.valore AS `Contact`,
                    contatti.tipo   AS `Type`
        FROM        contatti        INNER JOIN  contatti_clienti
                                    ON contatti.valore = contatti_clienti.contatto
```

```
    WHERE       cliente = var_cliente;
END$$
```

XCS-13

Il livello di isolamento scelto e' READ COMMITTED perché l'unico obiettivo è quello di leggere l'informazione relativa allo stato (aperto/finalizzato) di un ordine in maniera consistente. Dato che essa non viene letta nuovamente o utilizzata in altre operazioni non è necessario un livello di isolamento più alto.

```
CREATE PROCEDURE `visualizza_ordini_cliente` (
    IN var_cliente VARCHAR(16),
    IN var_status TINYINT
)
BEGIN
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        ROLLBACK;
        RESIGNAL;
    END;

    SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
    START TRANSACTION;

        IF var_status = 1 THEN
            SELECT      `id`                    AS  `Order`,
                        `indirizzo_spedizione`  AS  `Shipping_address`,
                        DATE(`data`)            AS  `Date`
            FROM        `ordini`
            WHERE       `cliente` = var_cliente AND
                        `aperto_or_finalizzato` = 1
            ORDER BY    `data` DESC;
        ELSE
            SELECT      `id`                    AS  `Order`,
                        `indirizzo_spedizione`  AS  `Shipping address`,
```

```
                DATE(`data`)            AS  `Date`,
                CASE
                    WHEN `aperto_or_finalizzato` = 1 THEN "Open"
                    ELSE "Finalized"
                END                     AS `Status`
            FROM    `ordini`
            WHERE   `cliente` = var_cliente
            ORDER BY `data` DESC;
        END IF;

    COMMIT;
END$$
```

XCS-14

Il livello di isolamento scelto e' READ COMMITTED perché l'unico obiettivo è quello di leggere l'informazione relativa alla giacenza relativa ad una certa specie in maniera consistente. Dato che essa non viene letta nuovamente o utilizzata in altre operazioni non è necessario un livello di isolamento più alto.

```
CREATE PROCEDURE `visualizza_specie_appartenenti_ad_ordine` (IN var_ordine INT)
BEGIN
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        ROLLBACK;
        RESIGNAL;
    END;

    SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
    START TRANSACTION;

        SELECT  `codice`    AS  `Species_code`,
                CONCAT(`nome_comune`, " (", `nome_latino`, + ")")
                            AS  `Species_name`,
                `quantita`  AS  `Quantity`
```

```
        FROM        `appartenenza_ordini` INNER JOIN  `specie_di_piante` ON
                    `specie_di_pianta` = `codice`
        WHERE       `ordine` = var_ordine;

    COMMIT;
END$$
```

XWM-01

Il livello di isolamento scelto e' READ COMMITTED perché l'unico obiettivo è quello di leggere l'informazione relativa alla giacenza relativa ad una certa specie in maniera consistente. Dato che essa non viene letta nuovamente o utilizzata in altre operazioni non è necessario un livello di isolamento più alto.

```
CREATE PROCEDURE `visualizza_dettagli_specie` (
    IN var_nome_comune VARCHAR(64),
    IN var_solo_fiorite TINYINT
)
BEGIN
    DECLARE var_ricerca VARCHAR(66);

    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        ROLLBACK;
        RESIGNAL;
    END;

    SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
    START TRANSACTION;

        IF var_nome_comune IS NOT NULL THEN
            SET var_ricerca = CONCAT("%", UPPER(var_nome_comune), "%");
        ELSE
            SET var_ricerca = "%";
        END IF;
```

```
IF var_solo_fiorite = 0 THEN
        SELECT      codice          AS      `Code`,
                    nome_latino     AS      `Latin_name`,
                    nome_comune AS      `Common_name`,
                    CASE
                            WHEN interni_or_esterni = 1 THEN 'Indoor'
                            ELSE 'Outdoor'
                    END             AS `Indoor/Outdoor`,
                    CASE
                            WHEN verde_or_fiorita = 1 THEN 'Green'
                            ELSE 'Flowery'
                    END             AS `Green/Flowery`,
                    CASE
                            WHEN esotica_si_no = 1 THEN 'Yes'
                            ELSE 'No'
                    END             AS `Exotic`,
                    valore          AS `Price`,
                    giacenza        AS `Stock`
        FROM        specie_di_piante INNER JOIN prezzi
                    ON specie_di_piante.codice = prezzi.specie_di_pianta
        WHERE       UPPER(nome_comune) LIKE var_ricerca AND
                    data_termine IS NULL
        ORDER BY    nome_comune ASC;
ELSE
        SELECT      codice          AS `Code`,
                    nome_latino     AS `Latin name`,
                    nome_comune AS `Common name`,
                    CASE
                            WHEN interni_or_esterni = 1 THEN 'Indoor'
                            ELSE 'Outdoor'
                    END             AS `Indoor/Outdoor`,
```

```
                    CASE
                        WHEN verde_or_fiorita = 1 THEN 'Green'
                        ELSE 'Flowery'
                    END         AS `Green/Flowery`,
                    CASE
                        WHEN esotica_si_no = 1 THEN 'Yes'
                        ELSE 'No'
                    END         AS `Exotic`,
                    valore      AS `Price`,
                    giacenza    AS `Stock`
                FROM    specie_di_piante INNER JOIN prezzi
                        ON specie_di_piante.codice = prezzi.specie_di_pianta
                WHERE   UPPER(nome_comune) LIKE var_ricerca AND
                        data_termine IS NULL AND
                        verde_or_fiorita = 0
                ORDER BY nome_comune ASC;
            END IF;

    COMMIT;
END$$
```

RCS-01

Il livello di isolamento scelto e' READ COMMITTED poichè sufficiente per restituire una lista consistente di contatti realmente inseriti (un contatto già inserito non può essere modificato).

```
CREATE PROCEDURE `aggiungi_contatto_referente` (
    IN var_referente    CHAR(11),
    IN var_contatto     VARCHAR(256),
    IN var_tipo         CHAR(16)
)
BEGIN
    DECLARE var_errno INT;
```

```
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        ROLLBACK;

        GET DIAGNOSTICS CONDITION 1 var_errno = MYSQL_ERRNO;

        IF var_errno = 1062 THEN
            SIGNAL SQLSTATE '45010'
            SET MESSAGE_TEXT = "Contact already inserted";
        ELSE
            RESIGNAL;
        END IF;
    END;

    SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
    START TRANSACTION;

        INSERT INTO contatti (`valore`, `tipo`) VALUES (var_contatto, var_tipo);
        INSERT INTO contatti_referenti (`contatto`, `referente`)
            VALUES (var_contatto, var_referente);

        SELECT      contatti.valore AS `Contact`,
                    contatti.tipo   AS `Type`
        FROM        contatti        INNER JOIN  contatti_referenti
                                    ON contatti.valore = contatti_referenti.contatto
        WHERE       referente = var_referente;

    COMMIT;
END$$
```

RCS-02

```
CREATE PROCEDURE `rimuovi_contatto_referente` (
```

```
        IN var_referente        CHAR(11),
        IN var_contatto         VARCHAR(256)
)
BEGIN
        DECLARE var_verifica_possesso INT;
        DECLARE var_errno INT;

        DECLARE EXIT HANDLER FOR SQLEXCEPTION
        BEGIN
                GET DIAGNOSTICS CONDITION 1 var_errno = MYSQL_ERRNO;

                IF var_errno = 1451 THEN
                        SIGNAL SQLSTATE '45012'
                        SET MESSAGE_TEXT = "Cannot remove the contact because it is associated
                                                to an order";
                ELSE
                        RESIGNAL;
                END IF;
        END;

        SELECT      COUNT(*)
        FROM        contatti_referenti
        WHERE       referente = var_referente AND
                    contatto = var_contatto
        INTO        var_verifica_possesso;

        IF var_verifica_possesso = 0 THEN
                SIGNAL SQLSTATE '45009'
                SET MESSAGE_TEXT = "Selected referent does not own this contact";
        ELSE
                DELETE FROM contatti WHERE valore = var_contatto;
        END IF;
```

```
END$$
```

RCS-03

```
CREATE PROCEDURE `modifica_contatto_preferito_referente` (
        IN var_referente        CHAR(11),
        IN var_contatto         VARCHAR(256)
)
BEGIN
        DECLARE var_errno INT;

        DECLARE EXIT HANDLER FOR SQLEXCEPTION
        BEGIN
                GET DIAGNOSTICS CONDITION 1 var_errno = MYSQL_ERRNO;

                IF var_errno = 1452 THEN
                        SIGNAL SQLSTATE '45013'
                        SET MESSAGE_TEXT = "Cannot set this contact as favourite one because it
                                                is not owned yet";
                ELSE
                        RESIGNAL;
                END IF;
        END;

        UPDATE      referenti
        SET         contatto_preferito = var_contatto
        WHERE       rivendita = var_referente;
END$$
```

RCS-04

```
CREATE PROCEDURE `visualizza_contatti_referente` (IN var_referente VARCHAR(16))
BEGIN
```

```
    SELECT      contatti.valore AS `Contact`,
                contatti.tipo   AS `Type`
    FROM        contatti  INNER JOINcontatti_referenti
                ON contatti.valore = contatti_referenti.contatto
    WHERE       referente = var_referente;
END$$
```

MNG-01

Il livello di isolamento scelto e' SERIALIZABLE per via dell'utilizzo della select max per ottenere l'id autogereato.

Non e' stato possibile utilizzare la funzione LAST_INSERT_ID() poiche' dalla documentazione si evince che "The ID that was generated is maintained in the server on a per-connection basis".

Per cui dato che la connessione e' la medesima per ciascun utente afferente ad una certa classe si e' preferita questa scelta alternativa.

```
CREATE PROCEDURE `inserisci_nuova_specie` (
    IN      var_nome_comune     VARCHAR(64),
    IN      var_nome_latino     VARCHAR(64),
    IN      var_int_est         TINYINT,
    IN      var_colorazione     VARCHAR(32),
    IN      var_esotica         TINYINT,
    IN      var_giacenza        INT,
    IN      var_prezzo          DECIMAL(7, 2),
    OUT     var_codice          INT
)
BEGIN
    DECLARE var_verde_or_fiorita TINYINT;
    DECLARE var_errno INT;

    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        ROLLBACK;

        GET DIAGNOSTICS CONDITION 1 var_errno = MYSQL_ERRNO;
```

```
        IF var_errno = 1062 THEN
                SIGNAL SQLSTATE '45021'
                SET MESSAGE_TEXT = "Species already stored";
        ELSE
                RESIGNAL;
        END IF;
END;

SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
START TRANSACTION;

        IF var_colorazione IS NULL THEN
                SET var_verde_or_fiorita = 1;
        ELSE
                SET var_verde_or_fiorita = 0;
        END IF;

        INSERT INTO specie_di_piante (nome_comune, nome_latino, interni_or_esterni,
        verde_or_fiorita, esotica_si_no, giacenza)
                VALUES (var_nome_comune, var_nome_latino, var_int_est,
                var_verde_or_fiorita, var_esotica, var_giacenza);

        SELECT MAX(codice) FROM specie_di_piante INTO var_codice;

        IF var_colorazione IS NOT NULL THEN
                INSERT INTO colorazioni (specie_fiorita, colore)
                        VALUES (var_codice, var_colorazione);
        END IF;

        INSERT INTO prezzi (specie_di_pianta, data_adozione, valore)
                VALUES (var_codice, NOW(), var_prezzo);
```

```
   COMMIT;
END$$
```

MNG-02

```
CREATE PROCEDURE `rimuovi_specie_di_pianta` (
        IN var_codice INT,
        OUT var_eliminazione_effettiva INT
)
BEGIN
        DECLARE var_errno INT;

        DECLARE EXIT HANDLER FOR SQLEXCEPTION
        BEGIN
                GET DIAGNOSTICS CONDITION 1 var_errno = MYSQL_ERRNO;

                IF var_errno = 1451 THEN
                        SIGNAL SQLSTATE '45029'
                        SET MESSAGE_TEXT = "Cannot remove the species because it is associated
                                                to an order";
                ELSE
                        RESIGNAL;
                END IF;
        END;

        SELECT      COUNT(*)
        FROM        specie_di_piante
        WHERE       codice = var_codice
        INTO        var_eliminazione_effettiva;

        IF var_eliminazione_effettiva = 1 THEN
                DELETE FROM specie_di_piante WHERE codice = var_codice;
```

```
    END IF;
END$$
```

MNG-03

In questo caso l'operazione non utilizza una transazione se pure il suo schema è analogo alle stored procedure di inserimento contatti. In quei casi però la transazione è indispensabile perché vengono effettuate due scritture le quali devono essere eseguite atomicamente.

In questo caso la scrittura è una sola e non vale la pena introdurre del overhead dovuto all'introduzione di una transazione soltanto per l'output della lista aggiornata (al più infatti essa può contenere un qualche colore in più che in realtà non dovrebbe esser presente).

```
CREATE PROCEDURE `aggiungi_colorazione` (
    IN var_specie_fiorita  INT,
    IN var_colore          VARCHAR(32)
)
BEGIN
    DECLARE var_errno INT;

    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        GET DIAGNOSTICS CONDITION 1 var_errno = MYSQL_ERRNO;

        IF var_errno = 1452 THEN
            SIGNAL SQLSTATE '45014'
            SET MESSAGE_TEXT = "Species does not exists";
        ELSEIF var_errno = 1062 THEN
            SIGNAL SQLSTATE '45030'
            SET MESSAGE_TEXT = "Coloring already specified for selected species";
        ELSE
            RESIGNAL;
        END IF;
    END;

    INSERT INTO colorazioni (`specie_fiorita`, `colore`)
```

```
        VALUES (var_specie_fiorita, var_colore);

    SELECT      UPPER(colore)       AS `Coloring`
    FROM        colorazioni
    WHERE       specie_fiorita = var_specie_fiorita;
END$$
```

MNG-04

```
CREATE PROCEDURE `rimuovi_colorazione` (
    IN      var_specie_fiorita          INT,
    IN      var_colore                  VARCHAR(32),
    OUT     var_eliminazione_effettiva  INT
)
BEGIN
    SELECT      COUNT(*)
    FROM        colorazioni
    WHERE       specie_fiorita = var_specie_fiorita AND
                colore = var_colore
    INTO        var_eliminazione_effettiva;

    IF var_eliminazione_effettiva = 1 THEN
        DELETE
        FROM        colorazioni
        WHERE       specie_fiorita = var_specie_fiorita AND
                    colore = var_colore;
    END IF;
END$$
```

MNG-05

Il livello scelto in questa transazione e' READ COMMITTED poiche' l'unica necessita' e' quella di leggere l'ultimo prezzo adottato in maniera consistente.

Viene inoltre fattorizzata l'impostazione della data per non aver nessuna differenza fra quella di terminazione e quella di nuova adozione

```
CREATE PROCEDURE `modifica_prezzo` (
        IN      var_specie                  INT,
        IN      var_prezzo                  DECIMAL(7, 2),
        OUT     var_aggiornamento_effettivo INT
)
BEGIN
        DECLARE var_nuova_data DATETIME;
        DECLARE var_prezzo_attuale DECIMAL(7, 2);
        DECLARE var_specie_non_trovata TINYINT DEFAULT 0;

        DECLARE EXIT HANDLER FOR SQLEXCEPTION
        BEGIN
                ROLLBACK;
                RESIGNAL;
        END;

        DECLARE CONTINUE HANDLER FOR NOT FOUND SET var_specie_non_trovata = 1;

        SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
        START TRANSACTION;

                SELECT      valore
                FROM        prezzi
                WHERE       specie_di_pianta = var_specie AND
                            data_termine IS NULL
                INTO  var_prezzo_attuale;

                IF var_specie_non_trovata = 1 THEN
                        SIGNAL SQLSTATE '45014'
                        SET MESSAGE_TEXT = "Species does not exists";
                END IF;
```

```
        IF var_prezzo_attuale <> var_prezzo THEN

            SET var_nuova_data = NOW();

            UPDATE      prezzi
            SET         data_termine = var_nuova_data
            WHERE       specie_di_pianta = var_specie AND
                        data_termine IS NULL;

            INSERT INTO prezzi (specie_di_pianta, data_adozione, valore)
                VALUES (var_specie, var_nuova_data, var_prezzo);
            SET var_aggiornamento_effettivo = 1;
        ELSE
            SET var_aggiornamento_effettivo = 0;
        END IF;

    COMMIT;
END$$
```

MNG-06

Il livello di isolamento scelto e' REPEATABLE READ poichè si suppone che le vendite avvenute in concorrenza con l'esecuzione di tale stored procedure (che quindi andrebbero in taluni casi ad aggiungere nuovi record alle tabella appartenenza_ordini) non possono influenzare fortemente un report su base mensile.

```
CREATE PROCEDURE `report_specie` (IN var_codice INT)
BEGIN
    DECLARE var_nome VARCHAR(131);
    DECLARE var_mese INT DEFAULT 1;
    DECLARE var_mese_string VARCHAR(16);
    DECLARE var_vendite INT;
    DECLARE var_verifica_esistenza INT;

    DECLARE EXIT HANDLER FOR SQLEXCEPTION
```

```
BEGIN
        ROLLBACK;
        RESIGNAL;
END;

DROP TEMPORARY TABLE IF EXISTS `report_mensile`;
CREATE TEMPORARY TABLE `report_mensile` (
        `Month` VARCHAR(16),
        `Sales` INT
);

START TRANSACTION;

        SELECT      COUNT(*)
        FROM        specie_di_piante
        WHERE       codice = var_codice
        INTO        var_verifica_esistenza;

        IF var_verifica_esistenza <> 1 THEN
                SIGNAL SQLSTATE '45014'
                SET MESSAGE_TEXT = "Species does not exists";
        END IF;

        SELECT      codice          AS `Species_code`,
                    nome_comune AS `Common_name`,
                    nome_latino     AS `Latin_name`
        FROM        specie_di_piante
        WHERE       codice = var_codice;

        for_loop: LOOP
                IF var_mese = 13 THEN
                        LEAVE for_loop;
```

```
END IF;

SELECT      SUM(`appartenenza_ordini`.`quantita`)
FROM        `appartenenza_ordini`       INNER JOIN `ordini`
            ON `appartenenza_ordini`.`ordine` = `ordini`.`id`
WHERE       `aperto_or_finalizzato` = 0 AND
            YEAR(`data`) = YEAR(CURDATE()) AND
            MONTH(`data`) = var_mese  AND
            `specie_di_pianta` = var_codice
INTO        var_vendite;

IF var_vendite IS NULL THEN
        SET var_vendite = 0;
END IF;

CASE var_mese
        WHEN 1 THEN SET var_mese_string = "January";
        WHEN 2 THEN SET var_mese_string = "February";
        WHEN 3 THEN SET var_mese_string = "March";
        WHEN 4 THEN SET var_mese_string = "April";
        WHEN 5 THEN SET var_mese_string = "May";
        WHEN 6 THEN SET var_mese_string = "June";
        WHEN 7 THEN SET var_mese_string = "July";
        WHEN 8 THEN SET var_mese_string = "August";
        WHEN 9 THEN SET var_mese_string = "September";
        WHEN 10 THEN SET var_mese_string = "October";
        WHEN 11 THEN SET var_mese_string = "November";
        WHEN 12 THEN SET var_mese_string = "December";
        ELSE SET var_mese_string = "####";
END CASE;

INSERT INTO `report_mensile` (`Month`, `Sales`)
```

```
                VALUES (var_mese_string, var_vendite);

            SET var_mese = var_mese + 1;

        END LOOP;

        SELECT * FROM `report_mensile`;
        DROP TEMPORARY TABLE `report_mensile`;

    COMMIT;
END$$
```

MNG-07

```
CREATE PROCEDURE `visualizza_colorazioni` (IN var_specie_fiorita INT)
BEGIN
    SELECT    UPPER(colore)    AS `Coloring`
    FROM      colorazioni
    WHERE     specie_fiorita = var_specie_fiorita;
END$$
```

WHC-01

```
CREATE PROCEDURE `inserisci_richiesta_fornitura` (
    IN var_fornitore    INT,
    IN var_specie       INT,
    IN var_quantita     INT
)
BEGIN
    DECLARE var_errno INT;

    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        GET DIAGNOSTICS CONDITION 1 var_errno = MYSQL_ERRNO;
```

```
        IF var_errno = 1452 THEN
            SIGNAL SQLSTATE '45025'
            SET MESSAGE_TEXT = "Either species or supplier does not exists";
        ELSE
            RESIGNAL;
        END IF;
    END;

    INSERT INTO richieste_di_forniture (`data`, `fornitore_scelto`, `specie_richiesta`,
    `quantita`)
        VALUES (NOW(), var_fornitore, var_specie, var_quantita);

    SELECT    DATE(`richieste_di_forniture`.`data`)    AS    `Date`,
              `fornitori`.`codice_fornitore`           AS    `Supplier_code`,
              `fornitori`.`nome`                       AS    `Supplier_name`,
              `richieste_di_forniture`.`quantita`      AS    `Quantity`

    FROM      `richieste_di_forniture` INNER JOIN `fornitori`
              ON `richieste_di_forniture`.`fornitore_scelto` = `fornitori`.`codice_fornitore`

    WHERE     `richieste_di_forniture`.`specie_richiesta` = var_specie AND
              `richieste_di_forniture`.`pendente_si_no` = 1

    ORDER BY  `richieste_di_forniture`.`data` DESC;
END$$
```

WHC-02

Il livello di isolamento scelto e' SERIALIZABLE per via dell'utilizzo della select max per ottenere l'id autogereato.

Non e' stato possibile utilizzare la funzione LAST_INSERT_ID() poiche' dalla documentazione si evince che "The ID that was generated is maintained in the server on a per-connection basis".

Per cui dato che la connessione e' la medesima per ciascun utente afferente ad una certa classe si e' preferita questa scelta alternativa.

```
CREATE PROCEDURE `inserisci_fornitore` (
        IN      var_codice_fiscale          CHAR(16),
        IN      var_nome                        VARCHAR(32),
        IN      var_specie                      INT,
        IN      var_indirizzo               VARCHAR(64),
        OUT var_codice_fornitore    INT
)
BEGIN
        DECLARE var_errno INT;

        DECLARE EXIT HANDLER FOR SQLEXCEPTION
        BEGIN
                ROLLBACK;

                GET DIAGNOSTICS CONDITION 1 var_errno = MYSQL_ERRNO;

                IF var_errno = 1062 THEN
                        SIGNAL SQLSTATE '45035'
                        SET MESSAGE_TEXT = "Supplier already inserted";
                ELSE
                        RESIGNAL;
                END IF;
        END;

        SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
        START TRANSACTION;

                INSERT INTO fornitori (`codice_fiscale`, `nome`)
                        VALUES (var_codice_fiscale, var_nome);
                SELECT MAX(codice_fornitore) FROM fornitori INTO var_codice_fornitore;
```

```
        INSERT INTO disponibilita_forniture (`fornitore`, `specie_di_pianta`)
            VALUES (var_codice_fornitore, var_specie);
        INSERT INTO indirizzi_fornitori (`indirizzo`, `fornitore`)
            VALUES (var_indirizzo, var_codice_fornitore);

    COMMIT;
END$$
```

WHC-03

Anche in questo caso vale lo stesso discorso fatto per l'operazione MNG-03 per quanto riguarda l'assenza di una transazione

```
CREATE PROCEDURE `aggiungi_disponibilita_fornitura` (
    IN var_fornitore    INT,
    IN var_specie       INT
)
BEGIN
    DECLARE var_errno INT;

    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        GET DIAGNOSTICS CONDITION 1 var_errno = MYSQL_ERRNO;

        IF var_errno = 1452 THEN
            SIGNAL SQLSTATE '45025'
            SET MESSAGE_TEXT = "Either species or supplier does not exists";
        ELSEIF var_errno = 1062 THEN
            SIGNAL SQLSTATE '45018'
            SET MESSAGE_TEXT = "Supply availability already inserted";
        ELSE
            RESIGNAL;
        END IF;
    END;
```

```
    INSERT INTO disponibilita_forniture (`fornitore`, `specie_di_pianta`)
        VALUES (var_fornitore, var_specie);

    SELECT    codice          AS    `Species_code`,
              nome_comune     AS    `Species_common_name`,
              nome_latino     AS    `Species_latin_name`
    FROM      disponibilita_forniture    INNER JOIN  specie_di_piante
                                         ON          codice = specie_di_pianta
    WHERE     fornitore = var_fornitore;
END$$
```

WHC-04

Il livello di isolamento scelto è REPEATABLE READ poiché sufficiente affinchè i dati presentati nel report siano affidabili. Non è necessario infatti evitare eventuali inserimenti fantasma in questo caso. Si tratterebbe infatti dell'aggiunta di una nuova specie e si può supporre che appena inserita nel sistema non richieda fornitura immediata.

```
CREATE PROCEDURE `report_giacenza` (IN var_range INT)
BEGIN
    DECLARE var_codice_cursor INT;
    DECLARE done INT DEFAULT FALSE;
    DECLARE cur CURSOR FOR
        SELECT    codice
        FROM      specie_di_piante
        ORDER BY  giacenza ASC
        LIMIT     var_range;
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        ROLLBACK;
        RESIGNAL;
    END;
```

```
START TRANSACTION;

OPEN cur;
read_loop: LOOP
        FETCH cur INTO var_codice_cursor;
        IF done THEN
                LEAVE read_loop;
        END IF;

        SELECT      codice AS `Species_code`,
                    CONCAT(nome_comune, " (", nome_latino, ")") AS  `Species_name`,
                    giacenza AS `Stock`
        FROM        specie_di_piante
        WHERE       codice = var_codice_cursor;

        SELECT      codice_fornitore AS `Supplier_code`,
                    nome AS `Supplier_name`,
                    COUNT(fornitore_scelto) AS `No_request_made`
        FROM        fornitori       INNER JOIN disponibilita_forniture
                                    ON codice_fornitore = fornitore
                                    LEFT JOIN     richieste_di_forniture
                                    ON fornitore_scelto = codice_fornitore
        WHERE       specie_di_pianta = var_codice_cursor AND
                    (specie_richiesta = var_codice_cursor OR specie_richiesta IS NULL)
        GROUP BY    `Supplier_code`, `Supplier_name`
        ORDER BY    `No_request_made` DESC
        LIMIT       5;

END LOOP;
CLOSE cur;
```

```
    COMMIT;
END$$
```

WHC-05

```
CREATE PROCEDURE `visualizza_specie_disponibili` (IN var_fornitore INT)
BEGIN
    SELECT    codice          AS    `Code`,
              nome_latino AS        `Latin_name`,
              nome_comune AS        `Common_name`
    FROM      specie_di_piante INNER JOIN disponibilita_forniture
              ON codice = specie_di_pianta
    WHERE     fornitore = var_fornitore
    ORDER BY  nome_comune ASC;
END$$
```

WHC-06

Anche in questo caso vale lo stesso discorso fatto per l'operazione MNG-03 per quanto riguarda l'assenza di una transazione

```
CREATE PROCEDURE `aggiungi_indirizzo_fornitore` (
    IN var_fornitore      INT,
    IN var_indirizzo      VARCHAR(64)
)
BEGIN
    DECLARE var_errno INT;

    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        GET DIAGNOSTICS CONDITION 1 var_errno = MYSQL_ERRNO;

        IF var_errno = 1452 THEN
            SIGNAL SQLSTATE '45032'
            SET MESSAGE_TEXT = "Supplier does not exists";
```

```
            ELSEIF var_errno = 1062 THEN
                SIGNAL SQLSTATE '45031'
                SET MESSAGE_TEXT = "Address already inserted";
            ELSE
                RESIGNAL;
            END IF;
        END;

        INSERT INTO indirizzi_fornitori (`indirizzo`, `fornitore`)
            VALUES (var_indirizzo, var_fornitore);

        SELECT  UPPER(indirizzo)  AS  `Address`
        FROM    indirizzi_fornitori
        WHERE   fornitore = var_fornitore;
END$$
```

WHC-07

```
CREATE PROCEDURE `visualizza_fornitori` (IN var_nome VARCHAR(32))
BEGIN
        DECLARE var_ricerca VARCHAR(34);

        IF var_nome IS NOT NULL THEN
            SET var_ricerca = CONCAT("%", var_nome, "%");
        ELSE
            SET var_ricerca = "%";
        END IF;

        SELECT  codice_fornitore  AS  `Supplier_code`,
                codice_fiscale    AS  `Fiscal_code`,
                nome              AS  `Name`
        FROM    fornitori
        WHERE   nome LIKE var_ricerca
```

```
    ORDER BY    codice_fornitore DESC;
END$$
```

OPC-01

Il livello di isolamento scelto e' SERIALIZABLE al fine di presentare uno snapshot affidabile sull'insieme dei pacchi relativo all'ordine selezionato anche nel caso in cui esso sia ancora in lavorazione e quindi potrebbe mutare concorrentemente alla produzione del report con eventuali inserimenti.

```
CREATE PROCEDURE `report_pacchi` (IN var_ordine INT)
BEGIN
    DECLARE var_numero_pacco INT;

    DECLARE done INT DEFAULT FALSE;
    DECLARE cur CURSOR FOR
        SELECT      numero
        FROM        pacchi
        WHERE       ordine = var_ordine;
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        ROLLBACK;
        RESIGNAL;
    END;

    DROP TEMPORARY TABLE IF EXISTS dettagli_pacchi;
    CREATE TEMPORARY TABLE dettagli_pacchi (
        `Pack_number`       INT UNSIGNED,
        `Species_code`      INT,
        `Latin_name`        VARCHAR(64),
        `Common_name`       VARCHAR(64),
        `Quantity`          INT UNSIGNED
    );
```

```
    SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
    START TRANSACTION;

        OPEN cur;
        read_loop: LOOP
            FETCH cur INTO var_numero_pacco;
            IF done THEN
                LEAVE read_loop;
            END IF;

            INSERT INTO dettagli_pacchi
                SELECT  pacco           AS  `Pack_number`,
                        codice          AS  `Species_code`,
                        nome_latino     AS  `Latin_name`,
                        nome_comune     AS  `Common_name`,
                        quantita        AS  `Quantity`
                FROM    specie_di_piante    INNER JOIN  contenuto_pacchi
                                            ON codice = specie_di_pianta
                WHERE   ordine = var_ordine AND
                        pacco = var_numero_pacco;
        END LOOP;
        CLOSE cur;

        SELECT  *
        FROM    dettagli_pacchi
        ORDER BY `Pack_number`;

        DROP TEMPORARY TABLE dettagli_pacchi;

    COMMIT;
END$$
```

OPC-02

Il livello di isolamento scelto e' SERIALIZABLE sia per via dell'utilizzo della select max per ottenere il numero identificativo del pacco relativamente all'ordine a cui afferisce, sia per via dell'utilizzo della funzione aggregata sum utilizzata nel trigger BI di contenuto_pacchi.

```
CREATE PROCEDURE `crea_pacco` (
        IN      var_ordine      INT,
        IN      var_specie      INT,
        IN      var_quantita    INT,
        OUT     var_numero      INT
)
BEGIN
        DECLARE var_errno INT;

        DECLARE EXIT HANDLER FOR SQLEXCEPTION
        BEGIN
                ROLLBACK;

                GET DIAGNOSTICS CONDITION 1 var_errno = MYSQL_ERRNO;

                IF var_errno = 1452 THEN
                        SIGNAL SQLSTATE '45027'
                        SET MESSAGE_TEXT = "Either species or order does not exists";
                ELSE
                        RESIGNAL;
                END IF;
        END;

   SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
   START TRANSACTION;

                SELECT      MAX(numero) + 1
                FROM        pacchi
```

```
        WHERE       ordine = var_ordine
        INTO        var_numero;

        IF var_numero IS NULL THEN
            SET var_numero = 1;
        END IF;

        INSERT INTO pacchi (`ordine`, `numero`) VALUES (var_ordine, var_numero);
        INSERT INTO contenuto_pacchi(`ordine`, `pacco`, `specie_di_pianta`, `quantita`)
            VALUES (var_ordine, var_numero, var_specie, var_quantita);

    COMMIT;
END$$
```

OPC-03

Il livello di isolamento scelto e' SERIALIZABLE per via dell'utilizzo della funzione aggregata sum utilizzata nel trigger BI di contenuto_pacchi.

```
CREATE PROCEDURE `aggiungi_specie_a_pacco` (
    IN var_specie       INT,
    IN var_ordine       INT,
    IN var_pacco        INT,
    IN var_quantita     INT
)
BEGIN
    DECLARE var_errno INT;

    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        ROLLBACK;

        GET DIAGNOSTICS CONDITION 1 var_errno = MYSQL_ERRNO;
```

```
        IF var_errno = 1452 THEN
            SIGNAL SQLSTATE '45026'
            SET MESSAGE_TEXT = "Either species or pack does not exists";
        ELSE
            RESIGNAL;
        END IF;
    END;

    SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
    START TRANSACTION;

        INSERT INTO contenuto_pacchi(`ordine`, `pacco`, `specie_di_pianta`, `quantita`)
            VALUES (var_ordine, var_pacco, var_specie, var_quantita);

    COMMIT;
END$$
```

OPC-04

Il livello di isolamento scelto è READ COMMITTED poiché è di interesse soltanto leggere dati consistenti in quanto i dati letti non vengono riutilizzati ma semplicemente restituiti all'utente.

```
CREATE PROCEDURE `visualizza_piante_rimanenti_da_impacchettare` (IN var_ordine INT)
BEGIN
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        ROLLBACK;
        RESIGNAL;
    END;

    SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
    START TRANSACTION;
```

```
        SELECT    specie_di_piante.codice              AS    `Species_code`,
                  CONCAT(specie_di_piante.nome_comune, " (",
                  specie_di_piante.nome_latino, + ")") AS    `Species_name`,
                  appartenenza_ordini.quantita         AS    `Quantity`,
                  CASE
                        WHEN SUM(contenuto_pacchi.quantita) IS NULL THEN 0
                        ELSE SUM(contenuto_pacchi.quantita)
                  END                                  AS    `Already_processed`

        FROM      specie_di_piante LEFT JOIN contenuto_pacchi
                  ON contenuto_pacchi.specie_di_pianta = specie_di_piante.codice
                  INNER JOIN  appartenenza_ordini
                  ON appartenenza_ordini.specie_di_pianta = specie_di_piante.codice

        WHERE     appartenenza_ordini.ordine = var_ordine AND
                  (contenuto_pacchi.ordine = var_ordine OR
                  contenuto_pacchi.ordine IS NULL)

        GROUP BY  `Species_code`, `Species_name`, `Quantity`;

    COMMIT;
END$$
```

OPC-05

Il livello di isolamento scelto è serializable per via dell'utilizzo della funzione verifica_completamento che fa uso di funzioni aggregate.

```
CREATE PROCEDURE `visualizza_stato_ordine` (IN var_ordine INT)
BEGIN
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        ROLLBACK;
        RESIGNAL;
```

```
    END;

    SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
    START TRANSACTION;
        SELECT      `ordini`.`id` AS `Order_ID`,
                    DATE(`ordini`.`data`) AS `Date`,
                    `ordini`.`indirizzo_spedizione` AS `Shipping_address`,
                    `ordini`.`contatto` AS `Chosen_contact`,
                    CASE
                        WHEN verifica_completamento(`id`)  THEN "Completed"
                        WHEN aperto_or_finalizzato = 1 THEN "Already open"
                        ELSE "Not yet completed"
                    END AS `Status`,
                    CASE
                        WHEN `clienti`.`privato_or_rivendita` = 1
                        THEN "Not expected"
                        ELSE concat(`referenti`.`nome`, " ", `referenti`.`cognome`)
                    END AS `Referent`
        FROM        `ordini`  INNER JOIN `clienti`
                    ON `ordini`.`cliente` = `clienti`.`codice`
                    LEFT JOIN `referenti`
                    ON `clienti`.`codice` = `referenti`.`rivendita`
        WHERE       `id` = var_ordine;


    COMMIT;
END$$
```

NRG-01

La transazione e' stata utilizzata in questo caso per effettuare un insieme di scritture in modalità "all or nothing" per cui non è necessario fissare un livello di isolamento (non avvengono letture).

```
CREATE PROCEDURE `registra_privato` (
    IN var_codice CHAR(16),
```

```
        IN var_nome   VARCHAR(32),
        IN var_res    VARCHAR(64),
        IN var_fat    VARCHAR(64),
        IN var_user   VARCHAR(128),
        IN var_pass   VARCHAR(128)
)
BEGIN
        DECLARE var_uuid CHAR(36);
        DECLARE var_errno INT;

        DECLARE EXIT HANDLER FOR SQLEXCEPTION
        BEGIN
                ROLLBACK;

                GET DIAGNOSTICS CONDITION 1 var_errno = MYSQL_ERRNO;

                IF var_errno = 1062 THEN
                        SIGNAL SQLSTATE '45011'
                        SET MESSAGE_TEXT = "User already exists";
                ELSE
                        RESIGNAL;
                END IF;
        END;

        START TRANSACTION;

                SELECT UUID() INTO var_uuid;

                IF verifica_password(var_pass) = 1 THEN
                        SIGNAL SQLSTATE '45003'
                        SET MESSAGE_TEXT = "Minimum password length is 8 characters";
                ELSEIF verifica_password(var_pass) = 2 THEN
```

```
                SIGNAL SQLSTATE '45004'
                SET MESSAGE_TEXT = "Password must not contains spaces";
        ELSE
                INSERT INTO utenti (`username`, `password`, `ruolo`, `uuid`)
                        VALUES (var_user, SHA2(CONCAT(var_pass, var_uuid), 512),
                        "PCS", var_uuid);
                INSERT INTO clienti (`codice`, `nome`, `indirizzo_residenza`,
                `indirizzo_fatturazione`, `privato_or_rivendita`)
                        VALUES (UPPER(var_codice), var_nome, var_res, var_fat, 1);
                INSERT INTO utenze_clienti (`utente`, `cliente`)
                        VALUES (var_user, UPPER(var_codice));
        END IF;

    COMMIT;
END$$
```

NRG-02

La transazione e' stata utilizzata in questo caso per effettuare un insieme di scritture in modalità "all or nothing" per cui non è necessario fissare un livello di isolamento (non avvengono letture).

```
CREATE PROCEDURE `registra_rivendita` (
    IN var_codice       CHAR(11),
    IN var_nome_riv     VARCHAR(32),
    IN var_res          VARCHAR(64),
    IN var_fat          VARCHAR(64),
    IN var_user         VARCHAR(128),
    IN var_pass         VARCHAR(128),
    IN var_nome_ref     VARCHAR(32),
    IN var_cognome_ref  VARCHAR(32)
)
BEGIN
    DECLARE var_uuid CHAR(36);
    DECLARE var_errno INT;
```

```
DECLARE EXIT HANDLER FOR SQLEXCEPTION
BEGIN
	ROLLBACK;
	GET DIAGNOSTICS CONDITION 1 var_errno = MYSQL_ERRNO;
	IF var_errno = 1062 THEN
		SIGNAL SQLSTATE '45011'
		SET MESSAGE_TEXT = "User already exists";
	ELSE
		RESIGNAL;
	END IF;
END;

START TRANSACTION;

	SELECT UUID() INTO var_uuid;

	IF verifica_password(var_pass) = 1 THEN
		SIGNAL SQLSTATE '45003'
		SET MESSAGE_TEXT = "Minimum password length is 8 characters";
	ELSEIF verifica_password(var_pass) = 2 THEN
		SIGNAL SQLSTATE '45004'
		SET MESSAGE_TEXT = "Password must not contains spaces";
	ELSE
		INSERT INTO utenti (`username`, `password`, `ruolo`, `uuid`)
			VALUES (var_user, SHA2(CONCAT(var_pass, var_uuid), 512),
			"RCS", var_uuid);
		INSERT INTO clienti (`codice`, `nome`, `indirizzo_residenza`,
		`indirizzo_fatturazione`, `privato_or_rivendita`)
			VALUES (var_codice, var_nome_riv, var_res, var_fat, 0);
		INSERT INTO utenze_clienti (`utente`, `cliente`)
			VALUES (var_user, var_codice);
```

```
                INSERT INTO referenti (`rivendita`, `nome`, `cognome`)
                        VALUES (var_codice, var_nome_ref, var_cognome_ref);
        END IF;

    COMMIT;
END$$
```

NRG-03

È stato necessario introdurre l'handler al fine di evitare il warning 1329: No data - zero rows fetched, selected, or processed

```
CREATE PROCEDURE `login` (
    IN  var_username        VARCHAR(128),
    IN  var_password        VARCHAR(128),
    OUT var_ruolo           INT,
    OUT var_codice_cliente  VARCHAR(16)
)
BEGIN
    DECLARE var_ruolo_enum ENUM("PCS", "RCS", "WHC",
                                "OPC", "MNG", "COS", "ERR");

    DECLARE CONTINUE HANDLER FOR NOT FOUND SET var_ruolo_enum = "ERR";

    SELECT  ruolo
    FROM    utenti
    WHERE   `username` = var_username and
            `password` = SHA2(CONCAT(var_password, `uuid`), 512)
    INTO    var_ruolo_enum;

    IF var_ruolo_enum = "PCS" OR var_ruolo_enum = "RCS" THEN
        SELECT  cliente
        FROM    utenze_clienti
        WHERE   utente = var_username
```

```
            INTO        var_codice_cliente;
    END IF;

    CASE var_ruolo_enum
        WHEN "PCS" THEN SET var_ruolo = 0;
        WHEN "RCS" THEN SET var_ruolo = 1;
        WHEN "WHC" THEN SET var_ruolo = 2;
        WHEN "OPC" THEN SET var_ruolo = 3;
        WHEN "MNG" THEN SET var_ruolo = 4;
        WHEN "COS" THEN SET var_ruolo = 5;
        ELSE SET var_ruolo = 6;
    END CASE;
END$$
```

ALL-01

Il livello di isolamento scelto per la transazione è READ COMMITTED per via del fatto che mediante tale stored procedure e' possibile modificare la password associata ad un'account e per effettuare un confronto corretto e' necessario leggere un dato gia' committato

```
CREATE PROCEDURE `modifica_password` (
    IN var_username             VARCHAR(32),
    IN var_vecchia_password     VARCHAR(128),
    IN var_nuova_password       VARCHAR(128)
)
BEGIN
    DECLARE var_vecchia_password_memorizzata VARCHAR(128);
    DECLARE var_uuid CHAR(36);

    DECLARE CONTINUE HANDLER FOR NOT FOUND
    SIGNAL SQLSTATE '45028' SET MESSAGE_TEXT = "User not exists";

    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
```

```
            ROLLBACK;
            RESIGNAL;
        END;

        SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
        START TRANSACTION;
            SELECT      `password`, `uuid`
            FROM        `utenti`
            WHERE       `username` = var_username
            INTO  var_vecchia_password_memorizzata, var_uuid;

            IF var_vecchia_password_memorizzata <>
            SHA2(CONCAT(var_vecchia_password, var_uuid), 512) THEN
                SIGNAL SQLSTATE '45017'
                SET MESSAGE_TEXT = 'Old password inserted does not match
                                        with the stored one';
            END IF;

            UPDATE      `utenti`
            SET         `password` = SHA2(CONCAT(var_nuova_password, var_uuid), 512)
            WHERE       `username` = var_username;
        COMMIT;
END$$
```

COS-01

La transazione e' stata utilizzata in questo caso per effettuare un insieme di scritture in modalità "all or nothing" per cui non è necessario fissare un livello di isolamento (non avvengono letture).

```
CREATE PROCEDURE `crea_utenza_dipendente` (
        IN var_username     VARCHAR(128),
        IN var_password     VARCHAR(128),
        IN var_ruolo        CHAR(3)
)
```

```
BEGIN
	DECLARE var_uuid CHAR(36);
	DECLARE var_errno INT;

	DECLARE EXIT HANDLER FOR SQLEXCEPTION
	BEGIN
		ROLLBACK;
		GET DIAGNOSTICS CONDITION 1 var_errno = MYSQL_ERRNO;
		IF var_errno = 1062 THEN
			SIGNAL SQLSTATE '45011'
			SET MESSAGE_TEXT = "User already exists";
		ELSE
			RESIGNAL;
		END IF;
	END;

	START TRANSACTION;
		IF var_ruolo = "PCS" OR var_ruolo = "RCS" THEN
			SIGNAL SQLSTATE '45033'
			SET MESSAGE_TEXT = "Selected role is not an employee";
		END IF;

		SELECT UUID() INTO var_uuid;

		IF verifica_password(var_password) = 1 THEN
			SIGNAL SQLSTATE '45003'
			SET MESSAGE_TEXT = "Minimum password length is 8 characters";
		ELSEIF verifica_password(var_password) = 2 THEN
			SIGNAL SQLSTATE '45004'
			SET MESSAGE_TEXT = "Password must not contains spaces";
		ELSE
			INSERT INTO utenti (`username`, `password`, `ruolo`, `uuid`)
```

```
                    VALUES (var_username,
                    SHA2(CONCAT(var_password, var_uuid), 512), var_ruolo, var_uuid);
            END IF;
        COMMIT;
END$$
```

Appendice: Implementazione

Codice SQL per instanziare il database

Creazione tabelle

```
-- -----------------------------------------------------
-- Schema verdesrl
-- -----------------------------------------------------
DROP SCHEMA IF EXISTS `verdesrl` ;


-- -----------------------------------------------------
-- Schema verdesrl
-- -----------------------------------------------------
CREATE SCHEMA IF NOT EXISTS `verdesrl` ;
USE `verdesrl` ;


-- -----------------------------------------------------
-- Table `verdesrl`.`utenti`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `verdesrl`.`utenti` ;

CREATE TABLE IF NOT EXISTS `verdesrl`.`utenti` (
  `username` VARCHAR(128) NOT NULL,
  `password` CHAR(128) NOT NULL,
  `ruolo` ENUM("PCS", "RCS", "WHC", "OPC", "MNG", "COS") NOT NULL,
  `uuid` CHAR(36) NOT NULL,
  PRIMARY KEY (`username`),
  UNIQUE INDEX `uuid_UNIQUE` (`uuid` ASC))
ENGINE = InnoDB;



-- -----------------------------------------------------
-- Table `verdesrl`.`specie_di_piante`
```

```
-- -----------------------------------------------------
DROP TABLE IF EXISTS `verdesrl`.`specie_di_piante` ;

CREATE TABLE IF NOT EXISTS `verdesrl`.`specie_di_piante` (
  `codice` INT NOT NULL AUTO_INCREMENT,
  `nome_latino` VARCHAR(64) NOT NULL,
  `nome_comune` VARCHAR(64) NOT NULL,
  `interni_or_esterni` TINYINT NOT NULL,
  `verde_or_fiorita` TINYINT NOT NULL,
  `esotica_si_no` TINYINT NOT NULL,
  `giacenza` INT UNSIGNED NOT NULL DEFAULT 0,
  PRIMARY KEY (`codice`),
  UNIQUE INDEX `nome_latino_UNIQUE` (`nome_latino` ASC),
  FULLTEXT INDEX `nome_comune_FULLTEXT` (`nome_comune`))
ENGINE = InnoDB;



-- -----------------------------------------------------
-- Table `verdesrl`.`prezzi`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `verdesrl`.`prezzi` ;

CREATE TABLE IF NOT EXISTS `verdesrl`.`prezzi` (
  `specie_di_pianta` INT NOT NULL,
  `data_adozione` DATETIME NOT NULL,
  `valore` DECIMAL(7,2) UNSIGNED NOT NULL,
  `data_termine` DATETIME NULL,
  PRIMARY KEY (`specie_di_pianta`, `data_adozione`),
  INDEX `fk_prezzi_specie_di_piante1_idx` (`specie_di_pianta` ASC),
  CONSTRAINT `fk_prezzi_specie_di_piante1`
    FOREIGN KEY (`specie_di_pianta`)
    REFERENCES `verdesrl`.`specie_di_piante` (`codice`)
```

```
    ON DELETE CASCADE
    ON UPDATE NO ACTION)
ENGINE = InnoDB;



-- -----------------------------------------------------
-- Table `verdesrl`.`contatti`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `verdesrl`.`contatti` ;

CREATE TABLE IF NOT EXISTS `verdesrl`.`contatti` (
  `valore` VARCHAR(256) NOT NULL,
  `tipo` ENUM("telefono", "cellulare", "email") NOT NULL,
  PRIMARY KEY (`valore`))
ENGINE = InnoDB;



-- -----------------------------------------------------
-- Table `verdesrl`.`clienti`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `verdesrl`.`clienti` ;

CREATE TABLE IF NOT EXISTS `verdesrl`.`clienti` (
  `codice` VARCHAR(16) NOT NULL,
  `nome` VARCHAR(32) NOT NULL,
  `indirizzo_residenza` VARCHAR(64) NOT NULL,
  `indirizzo_fatturazione` VARCHAR(64) NULL,
  `privato_or_rivendita` TINYINT NOT NULL,
  `contatto_preferito` VARCHAR(256) NULL,
  PRIMARY KEY (`codice`),
  INDEX `fk_clienti_contatti1_idx` (`contatto_preferito` ASC),
  CONSTRAINT `fk_clienti_contatti1`
```

```
    FOREIGN KEY (`contatto_preferito`)
    REFERENCES `verdesrl`.`contatti` (`valore`)
    ON DELETE SET NULL
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `verdesrl`.`referenti`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `verdesrl`.`referenti` ;

CREATE TABLE IF NOT EXISTS `verdesrl`.`referenti` (
  `rivendita` VARCHAR(16) NOT NULL,
  `nome` VARCHAR(32) NOT NULL,
  `cognome` VARCHAR(32) NOT NULL,
  `contatto_preferito` VARCHAR(256) NULL,
  INDEX `fk_referenti_contatti1_idx` (`contatto_preferito` ASC),
  PRIMARY KEY (`rivendita`),
  CONSTRAINT `fk_referenti_contatti1`
    FOREIGN KEY (`contatto_preferito`)
    REFERENCES `verdesrl`.`contatti` (`valore`)
    ON DELETE SET NULL
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_referenti_clienti1`
    FOREIGN KEY (`rivendita`)
    REFERENCES `verdesrl`.`clienti` (`codice`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

```
-- -----------------------------------------------------
-- Table `verdesrl`.`ordini`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `verdesrl`.`ordini` ;

CREATE TABLE IF NOT EXISTS `verdesrl`.`ordini` (
  `id` INT NOT NULL AUTO_INCREMENT,
  `cliente` VARCHAR(16) NOT NULL,
  `data` DATETIME NOT NULL,
  `aperto_or_finalizzato` TINYINT NOT NULL DEFAULT 1,
  `indirizzo_spedizione` VARCHAR(64) NOT NULL,
  `contatto` VARCHAR(256) NOT NULL,
  PRIMARY KEY (`id`),
  INDEX `fk_ordini_contatti1_idx` (`contatto` ASC),
  INDEX `fk_ordini_clienti1_idx` (`cliente` ASC),
  CONSTRAINT `fk_ordini_contatti1`
    FOREIGN KEY (`contatto`)
    REFERENCES `verdesrl`.`contatti` (`valore`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_ordini_clienti1`
    FOREIGN KEY (`cliente`)
    REFERENCES `verdesrl`.`clienti` (`codice`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `verdesrl`.`pacchi`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `verdesrl`.`pacchi` ;
```

```
CREATE TABLE IF NOT EXISTS `verdesrl`.`pacchi` (
  `ordine` INT NOT NULL,
  `numero` INT UNSIGNED NOT NULL,
  PRIMARY KEY (`ordine`, `numero`),
  INDEX `fk_pacchi_ordini1_idx` (`ordine` ASC),
  CONSTRAINT `fk_pacchi_ordini1`
    FOREIGN KEY (`ordine`)
    REFERENCES `verdesrl`.`ordini` (`id`)
    ON DELETE CASCADE
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `verdesrl`.`fornitori`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `verdesrl`.`fornitori` ;

CREATE TABLE IF NOT EXISTS `verdesrl`.`fornitori` (
  `codice_fornitore` INT NOT NULL AUTO_INCREMENT,
  `codice_fiscale` CHAR(16) NOT NULL,
  `nome` VARCHAR(32) NOT NULL,
  PRIMARY KEY (`codice_fornitore`),
  UNIQUE INDEX `codice_fiscale_UNIQUE` (`codice_fiscale` ASC))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `verdesrl`.`indirizzi_fornitori`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `verdesrl`.`indirizzi_fornitori` ;
```

```
CREATE TABLE IF NOT EXISTS `verdesrl`.`indirizzi_fornitori` (
  `indirizzo` VARCHAR(64) NOT NULL,
  `fornitore` INT NOT NULL,
  PRIMARY KEY (`indirizzo`),
  INDEX `fk_indirizzi_fornitori_fornitori1_idx` (`fornitore` ASC),
  CONSTRAINT `fk_indirizzi_fornitori_fornitori1`
    FOREIGN KEY (`fornitore`)
    REFERENCES `verdesrl`.`fornitori` (`codice_fornitore`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `verdesrl`.`richieste_di_forniture`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `verdesrl`.`richieste_di_forniture` ;

CREATE TABLE IF NOT EXISTS `verdesrl`.`richieste_di_forniture` (
  `data` DATETIME NOT NULL,
  `fornitore_scelto` INT NOT NULL,
  `specie_richiesta` INT NOT NULL,
  `quantita` INT UNSIGNED NOT NULL DEFAULT 0,
  `pendente_si_no` TINYINT NOT NULL DEFAULT 1,
  PRIMARY KEY (`data`, `fornitore_scelto`, `specie_richiesta`),
  INDEX `fk_richieste_di_forniture_specie_di_piante1_idx` (`specie_richiesta` ASC),
  INDEX `fk_richieste_di_forniture_fornitori1_idx` (`fornitore_scelto` ASC),
  CONSTRAINT `fk_richieste_di_forniture_specie_di_piante1`
    FOREIGN KEY (`specie_richiesta`)
    REFERENCES `verdesrl`.`specie_di_piante` (`codice`)
    ON DELETE NO ACTION
```

```
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_richieste_di_forniture_fornitori1`
    FOREIGN KEY (`fornitore_scelto`)
    REFERENCES `verdesrl`.`fornitori` (`codice_fornitore`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;



-- -----------------------------------------------------
-- Table `verdesrl`.`colorazioni`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `verdesrl`.`colorazioni` ;

CREATE TABLE IF NOT EXISTS `verdesrl`.`colorazioni` (
  `specie_fiorita` INT NOT NULL,
  `colore` VARCHAR(32) NOT NULL,
  PRIMARY KEY (`specie_fiorita`, `colore`),
  CONSTRAINT `fk_colorazioni_specie_di_piante1`
    FOREIGN KEY (`specie_fiorita`)
    REFERENCES `verdesrl`.`specie_di_piante` (`codice`)
    ON DELETE CASCADE
    ON UPDATE NO ACTION)
ENGINE = InnoDB;



-- -----------------------------------------------------
-- Table `verdesrl`.`contatti_clienti`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `verdesrl`.`contatti_clienti` ;

CREATE TABLE IF NOT EXISTS `verdesrl`.`contatti_clienti` (
```

```
  `contatto` VARCHAR(256) NOT NULL,
  `cliente` VARCHAR(16) NOT NULL,
  PRIMARY KEY (`contatto`),
  INDEX `fk_contatti_clienti_clienti1_idx` (`cliente` ASC),
  CONSTRAINT `fk_contatti_clienti_contatti1`
    FOREIGN KEY (`contatto`)
    REFERENCES `verdesrl`.`contatti` (`valore`)
    ON DELETE CASCADE
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_contatti_clienti_clienti1`
    FOREIGN KEY (`cliente`)
    REFERENCES `verdesrl`.`clienti` (`codice`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;



-- -----------------------------------------------------
-- Table `verdesrl`.`contatti_referenti`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `verdesrl`.`contatti_referenti` ;

CREATE TABLE IF NOT EXISTS `verdesrl`.`contatti_referenti` (
  `contatto` VARCHAR(256) NOT NULL,
  `referente` VARCHAR(16) NOT NULL,
  PRIMARY KEY (`contatto`),
  INDEX `fk_contatti_referenti_referenti1_idx` (`referente` ASC),
  CONSTRAINT `fk_contatti_referenti_contatti1`
    FOREIGN KEY (`contatto`)
    REFERENCES `verdesrl`.`contatti` (`valore`)
    ON DELETE CASCADE
    ON UPDATE NO ACTION,
```

```
  CONSTRAINT `fk_contatti_referenti_referenti1`
    FOREIGN KEY (`referente`)
    REFERENCES `verdesrl`.`referenti` (`rivendita`)
    ON DELETE CASCADE
    ON UPDATE NO ACTION)
ENGINE = InnoDB;



-- -----------------------------------------------------
-- Table `verdesrl`.`contenuto_pacchi`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `verdesrl`.`contenuto_pacchi` ;

CREATE TABLE IF NOT EXISTS `verdesrl`.`contenuto_pacchi` (
  `ordine` INT NOT NULL,
  `pacco` INT UNSIGNED NOT NULL,
  `specie_di_pianta` INT NOT NULL,
  `quantita` INT UNSIGNED NOT NULL DEFAULT 0,
  PRIMARY KEY (`ordine`, `pacco`, `specie_di_pianta`),
  INDEX `fk_contenuto_pacchi_specie_di_piante1_idx` (`specie_di_pianta` ASC),
  INDEX `fk_contenuto_pacchi_pacchi1_idx` (`ordine` ASC, `pacco` ASC),
  CONSTRAINT `fk_contenuto_pacchi_specie_di_piante1`
    FOREIGN KEY (`specie_di_pianta`)
    REFERENCES `verdesrl`.`specie_di_piante` (`codice`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_contenuto_pacchi_pacchi1`
    FOREIGN KEY (`ordine` , `pacco`)
    REFERENCES `verdesrl`.`pacchi` (`ordine` , `numero`)
    ON DELETE CASCADE
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

```
-- -----------------------------------------------------
-- Table `verdesrl`.`disponibilita_forniture`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `verdesrl`.`disponibilita_forniture` ;

CREATE TABLE IF NOT EXISTS `verdesrl`.`disponibilita_forniture` (
  `specie_di_pianta` INT NOT NULL,
  `fornitore` INT NOT NULL,
  PRIMARY KEY (`specie_di_pianta`, `fornitore`),
  INDEX `fk_disponibilta_forniture_specie_di_piante1_idx` (`specie_di_pianta` ASC),
  INDEX `fk_disponibilta_forniture_fornitori1_idx` (`fornitore` ASC),
  CONSTRAINT `fk_disponibilta_forniture_specie_di_piante1`
    FOREIGN KEY (`specie_di_pianta`)
    REFERENCES `verdesrl`.`specie_di_piante` (`codice`)
    ON DELETE CASCADE
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_disponibilta_forniture_fornitori1`
    FOREIGN KEY (`fornitore`)
    REFERENCES `verdesrl`.`fornitori` (`codice_fornitore`)
    ON DELETE CASCADE
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `verdesrl`.`utenze_clienti`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `verdesrl`.`utenze_clienti` ;

CREATE TABLE IF NOT EXISTS `verdesrl`.`utenze_clienti` (
```

```
  `cliente` VARCHAR(128) NOT NULL,
  `utente` VARCHAR(32) NOT NULL,
  PRIMARY KEY (`cliente`),
  INDEX `fk_utenze_clienti_utenti1_idx` (`utente` ASC),
  CONSTRAINT `fk_utenze_clienti_clienti1`
    FOREIGN KEY (`cliente`)
    REFERENCES `verdesrl`.`clienti` (`codice`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_utenze_clienti_utenti1`
    FOREIGN KEY (`utente`)
    REFERENCES `verdesrl`.`utenti` (`username`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;



-- -----------------------------------------------------
-- Table `verdesrl`.`appartenenza_ordini`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `verdesrl`.`appartenenza_ordini` ;

CREATE TABLE IF NOT EXISTS `verdesrl`.`appartenenza_ordini` (
  `ordine` INT NOT NULL,
  `specie_di_pianta` INT NOT NULL,
  `quantita` INT UNSIGNED NOT NULL DEFAULT 0,
  PRIMARY KEY (`ordine`, `specie_di_pianta`),
  INDEX `fk_appartenenza_ordini_specie_di_piante1_idx` (`specie_di_pianta` ASC),
  INDEX `fk_appartenenza_ordini_ordini1_idx` (`ordine` ASC),
  CONSTRAINT `fk_appartenenza_ordini_specie_di_piante1`
    FOREIGN KEY (`specie_di_pianta`)
    REFERENCES `verdesrl`.`specie_di_piante` (`codice`)
```

```
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_appartenenza_ordini_ordini1`
    FOREIGN KEY (`ordine`)
    REFERENCES `verdesrl`.`ordini` (`id`)
    ON DELETE CASCADE
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

Utenti e privilegi

```
DROP USER IF EXISTS non_registrato;

SET
SQL_MODE='ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO
_ZERO_DATE,ERROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION';

CREATE USER 'non_registrato' IDENTIFIED BY 'verdesrl';


GRANT EXECUTE ON procedure `verdesrl`.`login` TO 'non_registrato';

GRANT EXECUTE ON procedure `verdesrl`.`registra_privato` TO 'non_registrato';

GRANT EXECUTE ON procedure `verdesrl`.`registra_rivendita` TO 'non_registrato';

SET SQL_MODE = '';

DROP USER IF EXISTS cliente_privato;

SET
SQL_MODE='ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO
_ZERO_DATE,ERROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION';

CREATE USER 'cliente_privato' IDENTIFIED BY 'verdesrl';


GRANT EXECUTE ON procedure `verdesrl`.`modifica_password` TO 'cliente_privato';

GRANT EXECUTE ON procedure `verdesrl`.`crea_ordine` TO 'cliente_privato';

GRANT EXECUTE ON procedure `verdesrl`.`rimuovi_specie_da_ordine` TO 'cliente_privato';

GRANT EXECUTE ON procedure `verdesrl`.`modifica_ordine` TO 'cliente_privato';

GRANT EXECUTE ON procedure `verdesrl`.`aggiungi_specie_ad_ordine_esistente` TO
'cliente_privato';

GRANT EXECUTE ON procedure `verdesrl`.`finalizza_ordine` TO 'cliente_privato';

GRANT EXECUTE ON procedure `verdesrl`.`report_ordine` TO 'cliente_privato';

GRANT EXECUTE ON procedure `verdesrl`.`modifica_residenza` TO 'cliente_privato';
```

GRANT EXECUTE ON procedure `verdesrl`.`modifica_fatturazione` TO 'cliente_privato';

GRANT EXECUTE ON procedure `verdesrl`.`aggiungi_contatto_cliente` TO 'cliente_privato';

GRANT EXECUTE ON procedure `verdesrl`.`rimuovi_contatto_cliente` TO 'cliente_privato';

GRANT EXECUTE ON procedure `verdesrl`.`modifica_contatto_preferito_cliente` TO 'cliente_privato';

GRANT EXECUTE ON procedure `verdesrl`.`visualizza_ordini_cliente` TO 'cliente_privato';

GRANT EXECUTE ON procedure `verdesrl`.`visualizza_contatti_cliente` TO 'cliente_privato';

GRANT EXECUTE ON procedure `verdesrl`.`visualizza_dettagli_specie` TO 'cliente_privato';

GRANT EXECUTE ON procedure `verdesrl`.`visualizza_specie_appartenenti_ad_ordine` TO 'cliente_privato';

SET SQL_MODE = '';

DROP USER IF EXISTS cliente_rivendita;

SET SQL_MODE='ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZERO_DATE,ERROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION';

CREATE USER 'cliente_rivendita' IDENTIFIED BY 'verdesrl';

GRANT EXECUTE ON procedure `verdesrl`.`modifica_password` TO 'cliente_rivendita';

GRANT EXECUTE ON procedure `verdesrl`.`crea_ordine` TO 'cliente_rivendita';

GRANT EXECUTE ON procedure `verdesrl`.`rimuovi_specie_da_ordine` TO 'cliente_rivendita';

GRANT EXECUTE ON procedure `verdesrl`.`modifica_ordine` TO 'cliente_rivendita';

GRANT EXECUTE ON procedure `verdesrl`.`aggiungi_specie_ad_ordine_esistente` TO 'cliente_rivendita';

GRANT EXECUTE ON procedure `verdesrl`.`finalizza_ordine` TO 'cliente_rivendita';

GRANT EXECUTE ON procedure `verdesrl`.`report_ordine` TO 'cliente_rivendita';

GRANT EXECUTE ON procedure `verdesrl`.`modifica_residenza` TO 'cliente_rivendita';

GRANT EXECUTE ON procedure `verdesrl`.`modifica_fatturazione` TO 'cliente_rivendita';

GRANT EXECUTE ON procedure `verdesrl`.`aggiungi_contatto_cliente` TO 'cliente_rivendita';

GRANT EXECUTE ON procedure `verdesrl`.`rimuovi_contatto_cliente` TO 'cliente_rivendita';

GRANT EXECUTE ON procedure `verdesrl`.`modifica_contatto_preferito_cliente` TO 'cliente_rivendita';

GRANT EXECUTE ON procedure `verdesrl`.`aggiungi_contatto_referente` TO 'cliente_rivendita';

GRANT EXECUTE ON procedure `verdesrl`.`rimuovi_contatto_referente` TO 'cliente_rivendita';

```
GRANT EXECUTE ON procedure `verdesrl`.`modifica_contatto_preferito_referente` TO 'cliente_rivendita';

GRANT EXECUTE ON procedure `verdesrl`.`visualizza_ordini_cliente` TO 'cliente_rivendita';

GRANT EXECUTE ON procedure `verdesrl`.`visualizza_contatti_cliente` TO 'cliente_rivendita';

GRANT EXECUTE ON procedure `verdesrl`.`visualizza_contatti_referente` TO 'cliente_rivendita';

GRANT EXECUTE ON procedure `verdesrl`.`visualizza_dettagli_specie` TO 'cliente_rivendita';

GRANT EXECUTE ON procedure `verdesrl`.`visualizza_specie_appartenenti_ad_ordine` TO 'cliente_rivendita';

SET SQL_MODE = '';

DROP USER IF EXISTS addetto_diparimento_magazzino;

SET SQL_MODE='ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZERO_DATE,ERROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION';

CREATE USER 'addetto_diparimento_magazzino' IDENTIFIED BY 'verdesrl';


GRANT EXECUTE ON procedure `verdesrl`.`modifica_password` TO 'addetto_diparimento_magazzino';

GRANT EXECUTE ON procedure `verdesrl`.`inserisci_richiesta_fornitura` TO 'addetto_diparimento_magazzino';

GRANT EXECUTE ON procedure `verdesrl`.`inserisci_fornitore` TO 'addetto_diparimento_magazzino';

GRANT EXECUTE ON procedure `verdesrl`.`aggiungi_disponibilita_fornitura` TO 'addetto_diparimento_magazzino';

GRANT EXECUTE ON procedure `verdesrl`.`report_giacenza` TO 'addetto_diparimento_magazzino';

GRANT EXECUTE ON procedure `verdesrl`.`visualizza_dettagli_giacenza` TO 'addetto_diparimento_magazzino';

GRANT EXECUTE ON procedure `verdesrl`.`visualizza_dettagli_specie` TO 'addetto_diparimento_magazzino';

GRANT EXECUTE ON procedure `verdesrl`.`visualizza_fornitori` TO 'addetto_diparimento_magazzino';

GRANT EXECUTE ON procedure `verdesrl`.`visualizza_specie_disponibili` TO 'addetto_diparimento_magazzino';

GRANT EXECUTE ON procedure `verdesrl`.`aggiungi_indirizzo_fornitore` TO 'addetto_diparimento_magazzino';

SET SQL_MODE = '';

DROP USER IF EXISTS operatore_pacchi;
```

```
SET
SQL_MODE='ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO
_ZERO_DATE,ERROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION';

CREATE USER 'operatore_pacchi' IDENTIFIED BY 'verdesrl';


GRANT EXECUTE ON procedure `verdesrl`.`modifica_password` TO 'operatore_pacchi';

GRANT EXECUTE ON procedure `verdesrl`.`aggiungi_specie_a_pacco` TO 'operatore_pacchi';

GRANT EXECUTE ON procedure `verdesrl`.`report_pacchi` TO 'operatore_pacchi';

GRANT EXECUTE ON procedure `verdesrl`.`visualizza_stato_ordine` TO 'operatore_pacchi';

GRANT EXECUTE ON procedure `verdesrl`.`visualizza_piante_rimanenti_da_impacchettare` TO
'operatore_pacchi';

GRANT EXECUTE ON procedure `verdesrl`.`crea_pacco` TO 'operatore_pacchi';

SET SQL_MODE = '';

DROP USER IF EXISTS manager;

SET
SQL_MODE='ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO
_ZERO_DATE,ERROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION';

CREATE USER 'manager' IDENTIFIED BY 'verdesrl';


GRANT EXECUTE ON procedure `verdesrl`.`modifica_password` TO 'manager';

GRANT EXECUTE ON procedure `verdesrl`.`modifica_prezzo` TO 'manager';

GRANT EXECUTE ON procedure `verdesrl`.`inserisci_nuova_specie` TO 'manager';

GRANT EXECUTE ON procedure `verdesrl`.`rimuovi_specie_di_pianta` TO 'manager';

GRANT EXECUTE ON procedure `verdesrl`.`aggiungi_colorazione` TO 'manager';

GRANT EXECUTE ON procedure `verdesrl`.`rimuovi_colorazione` TO 'manager';

GRANT EXECUTE ON procedure `verdesrl`.`report_specie` TO 'manager';

GRANT EXECUTE ON procedure `verdesrl`.`visualizza_colorazioni` TO 'manager';

GRANT EXECUTE ON procedure `verdesrl`.`visualizza_dettagli_specie` TO 'manager';

SET SQL_MODE = '';

DROP USER IF EXISTS capo_personale;

SET
SQL_MODE='ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO
_ZERO_DATE,ERROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION';

CREATE USER 'capo_personale' IDENTIFIED BY 'verdesrl';
```

```
GRANT EXECUTE ON procedure `verdesrl`.`modifica_password` TO 'capo_personale';

GRANT EXECUTE ON procedure `verdesrl`.`crea_utenza_dipendente` TO 'capo_personale';
```

Insert predefinite

In fase di configurazione è previsto l'inserimento di un record all'interno della tabella utenti per il login dell'amministratore del sistema. Di seguito sono portati dettagli e codice:

username: admin

password: admin

ruolo: COS (Chief Of Staff)

```
START TRANSACTION;

USE `verdesrl`;

INSERT INTO `verdesrl`.`utenti` (`username`, `password`, `ruolo`, `uuid`) VALUES ('admin', '7603e55f21a11a3c9e71a00110dd97cc1552d7954bf90685eaa91c98bbfac26e24d4e9af41a122a212ccd423a7e305820cb57c7cfaa88409c78ff832859532b3', 'COS', '058fa777-84c0-11ea-8c72-d0039b002ee5');

COMMIT;
```

Funzioni

```
CREATE FUNCTION `verifica_proprietario`(id_ordine INT, codice_cliente VARCHAR(16))
RETURNS BOOL
DETERMINISTIC
BEGIN

    DECLARE var_cliente_ordine VARCHAR(16);

    SELECT    cliente
    FROM      ordini
    WHERE     id = id_ordine
    INTO      var_cliente_ordine;

    RETURN (var_cliente_ordine = codice_cliente);
```

```
END $$

CREATE FUNCTION `verifica_password`(var_password VARCHAR(128))
RETURNS TINYINT
DETERMINISTIC
BEGIN

	IF LENGTH(var_password) < 8 THEN
		RETURN 1;
	END IF;

	IF INSTR(var_password, " ") > 0 THEN
		RETURN 2;
	END IF;

	RETURN 0;

END $$

CREATE FUNCTION `verifica_completamento`(var_ordine INT)
RETURNS BOOL
DETERMINISTIC
BEGIN
	DECLARE var_quantita_in_pacchi INT UNSIGNED;
	DECLARE var_quantita_in_ordine INT UNSIGNED;

	SELECT	SUM(quantita)
	FROM	appartenenza_ordini
	WHERE	ordine = var_ordine
	INTO	var_quantita_in_ordine;

	SELECT	SUM(quantita)
```

```
        FROM        contenuto_pacchi
        WHERE       ordine = var_ordine
        INTO        var_quantita_in_pacchi;

        RETURN var_quantita_in_pacchi = var_quantita_in_ordine;
END $$
```

Codice del Front-End

FILE: chiefofstaff.c

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



static char curr_user[BUFFSIZE_L];



static bool attempt_add_employee_account(char *username, char *password, char *role)
{
   MYSQL_STMT *stmt;
   MYSQL_BIND param[3];

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

   param[0].buffer_type = MYSQL_TYPE_VAR_STRING; // IN var_username VARCHAR(128)
   param[0].buffer = username;
   param[0].buffer_length = strlen(username);
```

```
    param[1].buffer_type = MYSQL_TYPE_VAR_STRING; // IN var_username VARCHAR(128)
    param[1].buffer = password;
    param[1].buffer_length = strlen(password);

    param[2].buffer_type = MYSQL_TYPE_STRING; // IN var_username VARCHAR(128)
    param[2].buffer = role;
    param[2].buffer_length = strlen(role);

    if (!exec_sp(&stmt, param, "call crea_utenza_dipendente(?, ?, ?)"))
        return false;

    mysql_stmt_close(stmt);
    return true;
}

static void add_employee_account(void)
{
    char username[BUFFSIZE_L];
    char password[BUFFSIZE_L];
    char password_check[BUFFSIZE_L];
    char role[4];
    char choice;

    memset(username, 0, sizeof(username));
    memset(password, 0, sizeof(password));
    memset(password_check, 0, sizeof(password_check));

    init_screen(false);

    printf("Insert username: ");
    get_input(BUFFSIZE_L, username, false, true);
```

```
retype_pass:
    printf("Insert password: ");
    get_input(BUFFSIZE_L, password, true, true);

    printf("Retype password: ");
    get_input(BUFFSIZE_L, password_check, true, true);

    if (strcmp(password, password_check) != 0) {
        printf("Mismatch password, please retry!\n");
        goto retype_pass;
    }

    printf("\nWhat will be her/his role in the company?\n");
    printf("1) Warehouse clerk\n");
    printf("2) Order processor\n");
    printf("3) Manager\n");

    choice = multi_choice("Pick an option", "123", 3);

    switch (choice) {
    case '1': snprintf(role, 4, "WHC"); break;
    case '2': snprintf(role, 4, "OPC"); break;
    case '3': snprintf(role, 4, "MNG"); break;
    default:
        fprintf(stderr, "Invalid condition at %s:%d\n", __FILE__, __LINE__);
        abort();
    }

    putchar('\n');

    if (attempt_add_employee_account(username, password, role))
        printf("Employee account for %s [%s] was succesfylly created\n", username, role);
```

```
    else
        printf("Operation failed\n");

    printf("Press enter key to get back to menu ...\n");
    getchar();
}

void run_as_chief_of_staff(char *username)
{
    struct configuration cnf;
    char choice;

    memset(&cnf, 0, sizeof(cnf));
    memset(curr_user, 0, sizeof(curr_user));

    strncpy(curr_user, username, BUFFSIZE_L);

    if (parse_config("config/cos.user", &cnf, "=")) {
        fprintf(stderr, "Invalid configuration file selected (COS)\n");
        exit(EXIT_FAILURE);
    }

    if(mysql_change_user(conn, cnf.username, cnf.password, cnf.database)) {
        fprintf(stderr, "Unable to switch privileges\n");
        exit(EXIT_FAILURE);
    }

    while (true) {
        init_screen(true);
        printf("Welcome %s\n\n", curr_user);
        printf("*** What do you wanna do? ***\n\n");
        printf("a) Add an employee account\n");
```

```
        printf("p) Change password\n");
        printf("q) Quit\n");

        choice = multi_choice("Pick an option", "apq", 3);

        switch (choice) {
            case 'a': add_employee_account(); break;
            case 'p': change_password(curr_user); break;
            case 'q': printf("Bye bye!\n\n\n"); return;
            default:
                fprintf(stderr, "Invalid condition at %s:%d\n", __FILE__, __LINE__);
                abort();
        }
    }
}
```

FILE: customer.c

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

struct customer_info {
    char username[BUFFSIZE_L];
    char code[BUFFSIZE_XS];
    bool is_private;
};

struct create_order_sp_params {
    char shipping_address[BUFFSIZE_M];
```

```
    char contact[BUFFSIZE_XL];
    unsigned int species;
    unsigned int quantity;
};

struct order_info {
    unsigned int id;
    char date[BUFFSIZE_S];
    char shipping_address[BUFFSIZE_M];
    char chosen_contact[BUFFSIZE_XL];
    char status[BUFFSIZE_XS];
    /*
    * nel report viene presentato come <nome> <cognome>
    * -> 32 + 1 + 32 + 1 = 66
    */
    char referent[2 * BUFFSIZE_S];
};



static struct customer_info curr_customer;



static bool attempt_search_species_belonging_to_order(unsigned int order_id)
{
    MYSQL_STMT *stmt;
    MYSQL_BIND param[1];

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

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

```
    if (!exec_sp(&stmt, param, "call visualizza_specie_appartenenti_ad_ordine(?)"))
        return false;

    if (!dump_result_set(stmt, "Species belonging to selected order:",
LEADING_ZERO_BITMASK_IDX_0)) {
        CLOSE_AND_RETURN(false, stmt);
    }

    mysql_stmt_close(stmt);
    return true;
}

static void search_species(void)
{
    char name[BUFFSIZE_M];

    memset(name, 0, sizeof(name));

    init_screen(false);

    printf("*** Search species by name ***\n");
    printf("Insert the name to filter on (default all): ");
    get_input(BUFFSIZE_M, name, false, false);

    putchar('\n');

    if (!attempt_search_species(false, name))
        printf("Operation failed\n");

    printf("\nPress enter key to get back to menu ...\n");
    getchar();
```

```
}

static bool attempt_report_orders_short(bool only_open)
{
   MYSQL_STMT *stmt;
   MYSQL_BIND param[2];

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

   param[0].buffer_type = MYSQL_TYPE_VAR_STRING; // IN var_cliente       VARCHAR(16)
   param[0].buffer = curr_customer.code;
   param[0].buffer_length = strlen(curr_customer.code);

   param[1].buffer_type = MYSQL_TYPE_TINY; // IN var_status TINYINT
   param[1].buffer = &(only_open);
   param[1].buffer_length = sizeof(only_open);

   if (!exec_sp(&stmt, param, "call visualizza_ordini_cliente(?, ?)"))
      return false;

   if (!dump_result_set(stmt, "Open orders:", LEADING_ZERO_BITMASK_IDX_0)) {
      CLOSE_AND_RETURN(false, stmt);
   }

   mysql_stmt_close(stmt);
   return true;
}

static unsigned int attempt_open_order(struct create_order_sp_params *input)
{
   MYSQL_STMT *stmt;
   MYSQL_BIND param[6];
```

```
unsigned int id;

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

param[0].buffer_type = MYSQL_TYPE_VAR_STRING; // IN var_cliente       VARCHAR(16)
param[0].buffer = curr_customer.code;
param[0].buffer_length = strlen(curr_customer.code);

if (strlen(input->shipping_address) > 0) {
   param[1].buffer_type = MYSQL_TYPE_VAR_STRING; // IN var_ind_sped VARCHAR(64)
   param[1].buffer = input->shipping_address;
   param[1].buffer_length = strlen(input->shipping_address);
} else {
   param[1].buffer_type = MYSQL_TYPE_NULL; // IN var_ind_sped VARCHAR(64)
   param[1].buffer = NULL;
   param[1].buffer_length = 0;
}

if (strlen(input->contact) > 0) {
   param[2].buffer_type = MYSQL_TYPE_VAR_STRING; // IN var_contatto VARCHAR(64)
   param[2].buffer = input->contact;
   param[2].buffer_length = strlen(input->contact);
} else {
   param[2].buffer_type = MYSQL_TYPE_NULL; // IN var_contatto VARCHAR(64)
   param[2].buffer = NULL;
   param[2].buffer_length = 0;
}

param[3].buffer_type = MYSQL_TYPE_LONG; // IN var_specie INT
param[3].buffer = &(input->species);
param[3].buffer_length = sizeof(input->species);
```

```
    param[4].buffer_type = MYSQL_TYPE_LONG; // IN var_quantita INT
    param[4].buffer = &(input->quantity);
    param[4].buffer_length = sizeof(input->quantity);

    param[5].buffer_type = MYSQL_TYPE_LONG; // OUT var_id INT
    param[5].buffer = &id;
    param[5].buffer_length = sizeof(id);

    if (!exec_sp(&stmt, param, "call crea_ordine(?, ?, ?, ?, ?, ?)"))
        return 0;

    param[0].buffer_type = MYSQL_TYPE_LONG; // OUT var_id INT
    param[0].buffer = &id;
    param[0].buffer_length = sizeof(id);

    if (!fetch_res_sp(stmt, param))
        return 0;

    mysql_stmt_close(stmt);
    return id;
}

static void open_order(void)
{
    struct create_order_sp_params params;
    char buffer_for_integer[BUFFSIZE_XS];
    unsigned int order_id;
    char spec_name[BUFFSIZE_M];

    memset(&params, 0, sizeof(params));
    memset(buffer_for_integer, 0, sizeof(buffer_for_integer));
    memset(spec_name, 0, sizeof(spec_name));
```

```
init_screen(false);

printf("*** Open a new order ***\n");
printf("Customer code.....: %s\n", curr_customer.code);

printf("Insert shipping address (default residential address).....: ");
get_input(BUFFSIZE_M, params.shipping_address, false, false);

printf("Insert contact (default favourite one).....: ");
get_input(BUFFSIZE_XL, params.contact, false, false);

if (ask_for_tips("Do you wanna search species by name to find the right code", 0)) {
    printf("\nInsert the name to filter on (default all).....: ");
    get_input(BUFFSIZE_M, spec_name, false, false);

    if (!attempt_search_species(false, spec_name))
        printf("Operation failed\n");

    putchar('\n');
}

printf("Insert species code.....: ");
get_input(BUFFSIZE_XS, buffer_for_integer, false, true);
params.species = strtol(buffer_for_integer, NULL, 10);

printf("Insert relative quantity.....: ");
get_input(BUFFSIZE_XS, buffer_for_integer, false, true);
params.quantity = strtol(buffer_for_integer, NULL, 10);

putchar('\n');
```

```
    order_id = attempt_open_order(&params);
    if (order_id > 0)
        printf("New order opened (ID: %010u)\n", order_id);
    else
        printf("No order has opened\n");

    printf("Press enter key to get back to menu ...\n");
    getchar();
}

static bool attempt_to_add_spec_to_order(unsigned int order_id, unsigned int species_code, unsigned int quantity)
{
    MYSQL_STMT *stmt;
    MYSQL_BIND param[4];

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

    param[0].buffer_type = MYSQL_TYPE_VAR_STRING; // IN var_cliente       VARCHAR(16)
    param[0].buffer = curr_customer.code;
    param[0].buffer_length = strlen(curr_customer.code);

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

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

    param[3].buffer_type = MYSQL_TYPE_LONG; // IN var_quantita INT
    param[3].buffer = &quantity;
```

```
    param[3].buffer_length = sizeof(quantity);

    if (!exec_sp(&stmt, param, "call aggiungi_specie_ad_ordine_esistente(?, ?, ?, ?)"))
        return false;

    mysql_stmt_close(stmt);
    return true;
}

static int attempt_to_modify_order(unsigned int order_id, unsigned int species_code, unsigned int quantity)
{
    MYSQL_STMT *stmt;
    MYSQL_BIND param[5];
    int affected_rows;

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

    param[0].buffer_type = MYSQL_TYPE_VAR_STRING; // IN var_cliente       VARCHAR(16)
    param[0].buffer = curr_customer.code;
    param[0].buffer_length = strlen(curr_customer.code);

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

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

    param[3].buffer_type = MYSQL_TYPE_LONG; // IN var_quantita INT
    param[3].buffer = &quantity;
```

```
    param[3].buffer_length = sizeof(quantity);

    param[4].buffer_type = MYSQL_TYPE_LONG; // OUT var_aggiornamento_eff INT
    param[4].buffer = &affected_rows;
    param[4].buffer_length = sizeof(affected_rows);

    if (!exec_sp(&stmt, param, "call modifica_ordine(?, ?, ?, ?, ?)"))
        return -1;

    param[0].buffer_type = MYSQL_TYPE_LONG; // OUT var_aggiornamento_eff INT
    param[0].buffer = &affected_rows;
    param[0].buffer_length = sizeof(affected_rows);

    if (!fetch_res_sp(stmt, param))
        return -1;

    mysql_stmt_close(stmt);
    return affected_rows;
}

static void exec_op_on_order(bool is_add)
{
    char buffer_for_integer[BUFFSIZE_XS];
    char spec_name[BUFFSIZE_M];
    unsigned int order_id;
    unsigned int species_code;
    unsigned int quantity;
    int ret;

    memset(buffer_for_integer, 0, sizeof(buffer_for_integer));
    memset(spec_name, 0, sizeof(spec_name));
```

```
init_screen(false);

if (is_add)
   printf("*** Add a species to already opened order ***\n");
else
   printf("*** Change the number of plants belonging to a species in an order ***\n");



printf("Customer code.....................................................%s: %s\n",
      (is_add) ? "" : "....", curr_customer.code);

if (ask_for_tips("Do you wanna see a report of your open orders", (is_add) ? 13 : 17)) {
   if (!attempt_report_orders_short(true))
      printf("Operation failed\n");

   putchar('\n');
}

printf("Insert order id.....................................................%s: ", (is_add) ? "" : "....");
get_input(BUFFSIZE_XS, buffer_for_integer, false, true);
order_id = strtol(buffer_for_integer, NULL, 10);

if (is_add) {
   if (ask_for_tips("Do you wanna search species by name to find the right code", 0)) {
      printf("\nInsert the name to filter on (default all).....: ");
      get_input(BUFFSIZE_M, spec_name, false, false);

      if (!attempt_search_species(false, spec_name))
         printf("Operation failed\n");

      putchar('\n');
   }
```

```
    } else {
        if (ask_for_tips("Do you wanna see a list of species belonging to selected order", 0)) {
            if (!attempt_search_species_belonging_to_order(order_id))
                printf("Operation failed\n");

            putchar('\n');
        }
    }

    printf("Insert species code.....%s: ", (is_add) ? "" : "....");
    get_input(BUFFSIZE_XS, buffer_for_integer, false, true);
    species_code = strtol(buffer_for_integer, NULL, 10);

    printf("Insert relative quantity.....%s: ", (is_add) ? "" : "....");
    get_input(BUFFSIZE_XS, buffer_for_integer, false, true);
    quantity = strtol(buffer_for_integer, NULL, 10);

    putchar('\n');

    if (is_add) {
        if (attempt_to_add_spec_to_order(order_id, species_code, quantity))
            printf("Species %u succesfully added to your order (ID %010u)\n", species_code, order_id);
        else
            printf("Operation failed\n");
    } else {
        ret = attempt_to_modify_order(order_id, species_code, quantity);
        if (ret == 0)
            printf("Nothing has changed (species %u not in order [ID %010u])\n", species_code,
order_id);
        else if (ret > 0)
            printf("Species %u succesfully updated in your order (ID %010u)\n", species_code,
order_id);
        else
```

```
        printf("Operation failed\n");
    }

    printf("Press enter key to get back to menu ...\n");
    getchar();
}

static int attempt_to_remove_spec_from_order(unsigned int order_id, unsigned int species_code, int
*order_status)
{
    MYSQL_STMT *stmt;
    MYSQL_BIND param[5];
    int affected_rows;

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

    param[0].buffer_type = MYSQL_TYPE_VAR_STRING; // IN var_cliente       VARCHAR(16)
    param[0].buffer = curr_customer.code;
    param[0].buffer_length = strlen(curr_customer.code);

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

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

    param[3].buffer_type = MYSQL_TYPE_LONG; // OUT var_ordine_eliminato_si_no INT
    param[3].buffer = order_status;
    param[3].buffer_length = sizeof(*order_status);
```

```
    param[4].buffer_type = MYSQL_TYPE_LONG; // OUT var_eliminazione_eff INT
    param[4].buffer = &affected_rows;
    param[4].buffer_length = sizeof(affected_rows);

    if (!exec_sp(&stmt, param, "call rimuovi_specie_da_ordine(?, ?, ?, ?, ?)"))
        return -1;

    param[0].buffer_type = MYSQL_TYPE_LONG; // OUT var_ordine_eliminato_si_no INT
    param[0].buffer = order_status;
    param[0].buffer_length = sizeof(*order_status);

    param[1].buffer_type = MYSQL_TYPE_LONG; // OUT var_eliminazione_eff INT
    param[1].buffer = &affected_rows;
    param[1].buffer_length = sizeof(affected_rows);

    if (!fetch_res_sp(stmt, param))
        return -1;

    mysql_stmt_close(stmt);
    return affected_rows;
}

static void remove_spec_from_order(void)
{
    char buffer_for_integer[BUFFSIZE_XS];
    unsigned int order_id;
    unsigned int species_code;
    int order_status;
    int ret;

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

```
init_screen(false);
printf("*** Remove a species from an order not closed yet ***\n");
printf("Customer code.....: %s\n",
      curr_customer.code);

if (ask_for_tips("Do you wanna see a report of your open orders", 17)) {
   if (!attempt_report_orders_short(true))
      printf("Operation failed\n");

   putchar('\n');
}

printf("Insert order id.....: ");
get_input(BUFFSIZE_XS, buffer_for_integer, false, true);
order_id = strtol(buffer_for_integer, NULL, 10);

if (ask_for_tips("Do you wanna see a list of species belonging to selected order", 0)) {
   if (!attempt_search_species_belonging_to_order(order_id))
      printf("Operation failed\n");

   putchar('\n');
}

printf("Insert species code.....: ");
get_input(BUFFSIZE_XS, buffer_for_integer, false, true);
species_code = strtol(buffer_for_integer, NULL, 10);

putchar('\n');

ret = attempt_to_remove_spec_from_order(order_id, species_code, &order_status);

if (ret > 0) {
```

```
        printf("Species %u succesfully deleted from your order (ID %010u)\n", species_code, order_id);


        if (order_status == 1)
            printf("Order (ID %010u) has been deleted (there were no more plants belonging to it)\n", order_id);
    } else if (ret == 0) {
        printf("Nothing has changed (species %u not in order [ID %010u])\n", species_code, order_id);
    } else {
        printf("Operation failed\n");
    }

    printf("Press enter key to get back to menu ...\n");
    getchar();
}

static int attempt_finalize_order(unsigned int order_id)
{
    MYSQL_STMT *stmt;
    MYSQL_BIND param[2];

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

    param[0].buffer_type = MYSQL_TYPE_VAR_STRING; // IN var_cliente       VARCHAR(16)
    param[0].buffer = curr_customer.code;
    param[0].buffer_length = strlen(curr_customer.code);

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

    if (!exec_sp(&stmt, param, "call finalizza_ordine(?, ?)"))
        return false;
```

```
    mysql_stmt_close(stmt);
    return true;
}

static void finalize_order(void)
{
    char buffer_for_integer[BUFFSIZE_XS];
    unsigned int order_id;

    memset(buffer_for_integer, 0, sizeof(buffer_for_integer));

    init_screen(false);

    printf("*** Finalize an order ***\n");
    printf("Customer code.....: %s\n", curr_customer.code);

    if (ask_for_tips("Do you wanna see a report of your open orders", 0)) {
        if (!attempt_report_orders_short(true))
            printf("Operation failed\n");

        putchar('\n');
    }

    printf("Insert order id.....: ");
    get_input(BUFFSIZE_XS, buffer_for_integer, false, true);
    order_id = strtol(buffer_for_integer, NULL, 10);

    putchar('\n');

    if (attempt_finalize_order(order_id))
        printf("Order %010u has been finalized\n", order_id);
```

```
    else
        printf("Operation failed\n");

    printf("Press enter key to get back to menu ...\n");
    getchar();
}

static bool attempt_update_addr(char *addr, bool is_res)
{
    char sp_str[BUFFSIZE_L];
    MYSQL_STMT *stmt;
    MYSQL_BIND param[2];

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

    param[0].buffer_type = MYSQL_TYPE_VAR_STRING; // IN var_cliente        VARCHAR(16)
    param[0].buffer = curr_customer.code;
    param[0].buffer_length = strlen(curr_customer.code);

    if (strlen(addr) != 0) {
        param[1].buffer_type = MYSQL_TYPE_VAR_STRING; // IN var_indirizzo VARCHAR(64)
        param[1].buffer = addr;
        param[1].buffer_length = strlen(addr);
    } else {
        param[1].buffer_type = MYSQL_TYPE_NULL; // IN var_indirizzo VARCHAR(64)
        param[1].buffer = NULL;
        param[1].buffer_length = 0;
    }

    snprintf(sp_str, BUFFSIZE_L, "call modifica_%s(?, ?)", (is_res) ? "residenza" : "fatturazione");
    if (!exec_sp(&stmt, param, sp_str))
```

```
        return false;

    mysql_stmt_close(stmt);
    return true;
}

static void update_addr(bool is_res)
{
    char addr[BUFFSIZE_M];

    memset(addr, 0, sizeof(addr));

    init_screen(false);

    printf("*** Update your %s address ***\n", (is_res) ? "residential" : "billing");
    printf("Customer code.....%s: %s\n", (is_res) ? "" : ".....", curr_customer.code);
    printf("Insert new address %s: ", (is_res) ? "" : " (default null)");
    get_input(BUFFSIZE_M, addr, false, is_res);

    putchar('\n');

    if (attempt_update_addr(addr, is_res))
        printf("Address succesfully updated\n");
    else
        printf("Operation failed\n");

    printf("Press enter key to get back to menu ...\n");
    getchar();
}

static bool attempt_show_contact_list(bool is_customer)
{
```

```
    MYSQL_STMT *stmt;
    MYSQL_BIND param[1];
    char sp_str[BUFFSIZE_L];

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

    param[0].buffer_type = MYSQL_TYPE_VAR_STRING; // IN var_cliente       VARCHAR(16)
    param[0].buffer = curr_customer.code;
    param[0].buffer_length = strlen(curr_customer.code);

    snprintf(sp_str, BUFFSIZE_L, "call visualizza_contatti_%s(?)", (is_customer) ? "cliente" : "referente");
    if (!exec_sp(&stmt, param, sp_str))
        return false;

    if (!dump_result_set(stmt, "\nContact list:", LEADING_ZERO_BITMASK_IDX_0)) {
        CLOSE_AND_RETURN(false, stmt);
    }

    mysql_stmt_close(stmt);
    return true;
}

static bool attempt_to_modify_contact_list(char *contact, bool is_customer, bool to_delete)
{
    char sp_str[BUFFSIZE_L];
    MYSQL_STMT *stmt;
    MYSQL_BIND param[2];

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

```
    param[0].buffer_type = MYSQL_TYPE_VAR_STRING; // IN var_cliente       VARCHAR(16)
    param[0].buffer = curr_customer.code;
    param[0].buffer_length = strlen(curr_customer.code);

    param[1].buffer_type = MYSQL_TYPE_VAR_STRING; // IN var_contatto VARCHAR(256)
    param[1].buffer = contact;
    param[1].buffer_length = strlen(contact);

    if (to_delete)  {
        if (is_customer)
            snprintf(sp_str, BUFFSIZE_L, "call rimuovi_contatto_cliente(?, ?)");
        else
            snprintf(sp_str, BUFFSIZE_L, "call rimuovi_contatto_referente(?, ?)");
    } else {
        if (is_customer)
            snprintf(sp_str, BUFFSIZE_L, "call modifica_contatto_preferito_cliente(?, ?)");
        else
            snprintf(sp_str, BUFFSIZE_L, "call modifica_contatto_preferito_referente(?, ?)");
    }

    if (!exec_sp(&stmt, param, sp_str))
        return false;

    mysql_stmt_close(stmt);
    return true;
}

static void modify_contact_list(bool is_customer, bool to_delete)
{
    char contact[BUFFSIZE_XL];
    char message[BUFFSIZE_L];
```

```
   memset(contact, 0, sizeof(contact));
   memset(message, 0, sizeof(message));

   init_screen(false);

   if (to_delete)
      printf("*** Remove a contact from your %slist ***\n", (is_customer) ? "" : "referent");
   else
      printf("*** Change %sfavourite contact %s***\n", (is_customer) ? "your" : "", (is_customer) ? "" : "of your referent ");

   printf("%s code.....%s: %s\n",
         (is_customer) ? "Customer" : "Referent",
         (is_customer) ? "": ".....",
         curr_customer.code);

   snprintf(message, BUFFSIZE_L, "Do you wanna see a report of your %scontacts",
          (is_customer) ? "" : "referent ");

   if (ask_for_tips(message, 0)) {
      if (!attempt_show_contact_list(is_customer))
         printf("Operation failed\n");

      putchar('\n');
   }

   printf("Insert contact.....%s: ", (is_customer) ? "": ".....");
   get_input(BUFFSIZE_XL, contact, false, true);

   putchar('\n');
```

```
    if (attempt_to_modify_contact_list(contact, is_customer, to_delete))
        printf("Contact succesfully %s\n", (to_delete) ? "removed" : "changed");
    else
        printf("Operation failed\n");

    printf("Press enter key to get back to menu ...\n");
    getchar();
}

static bool attempt_add_contact(char *contact, char *type, bool is_customer)
{
    char sp_str[BUFFSIZE_L];
    MYSQL_STMT *stmt;
    MYSQL_BIND param[3];

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

    param[0].buffer_type = MYSQL_TYPE_VAR_STRING; // IN var_cliente       VARCHAR(16)
    param[0].buffer = curr_customer.code;
    param[0].buffer_length = strlen(curr_customer.code);

    param[1].buffer_type = MYSQL_TYPE_VAR_STRING; // IN var_contatto VARCHAR(256)
    param[1].buffer = contact;
    param[1].buffer_length = strlen(contact);

    param[2].buffer_type = MYSQL_TYPE_VAR_STRING; // IN var_tipo CHAR(16)
    param[2].buffer = type;
    param[2].buffer_length = strlen(type);

    snprintf(sp_str, BUFFSIZE_L, "call aggiungi_contatto_%s(?, ?, ?)",
            (is_customer) ? "cliente" : "referente");
```

```
    if (!exec_sp(&stmt, param, sp_str))
        return false;

    if (!dump_result_set(stmt, "\nUpdated list:", 0)) {
        CLOSE_AND_RETURN(false, stmt);
    }

    mysql_stmt_close(stmt);
    return true;
}

static void add_contact(bool is_customer, bool show_prompt)
{
    char contact[BUFFSIZE_XL];
    char type[BUFFSIZE_XS];
    char message[BUFFSIZE_L];
    char choice;

    memset(contact, 0, sizeof(contact));
    memset(type, 0, sizeof(type));
    memset(message, 0, sizeof(message));

    init_screen(false);

    printf("*** Add a contact to your %slist ***\n", (is_customer) ? "" : "referent ");
    printf("%s code.....%s: %s\n",
           (is_customer) ? "Customer" : "Referent",
           (is_customer) ? "": ".....",
           curr_customer.code);
```

```
    snprintf(message, BUFFSIZE_L, "Do you wanna see a report of your %scontacts", (is_customer)
? "" : "referent ");

    if (ask_for_tips(message, 0)) {
        if (!attempt_show_contact_list(is_customer))
            printf("Operation failed\n");

        putchar('\n');
    }
    printf("Insert new contact.....%s: ", (is_customer) ? "": ".....");
    get_input(BUFFSIZE_XL, contact, false, true);

    snprintf(message, BUFFSIZE_L, "Select type [m]obile, [l]andline, [e]mail%s",
            (is_customer) ? "": ".....");

    choice = multi_choice(message, "mle", 3);

    switch (choice) {
    case 'm': snprintf(type, BUFFSIZE_XS, "cellulare"); break;
    case 'l': snprintf(type, BUFFSIZE_XS, "telefono"); break;
    case 'e': snprintf(type, BUFFSIZE_XS, "email"); break;
    default:
        fprintf(stderr, "Invalid condition at %s:%d\n", __FILE__, __LINE__);
        abort();
    }

    putchar('\n');

    if (attempt_add_contact(contact, type, is_customer)) {
        choice = multi_choice("\nDo you wanna set this as favourite contact?", "yn", 2);

        if (choice == 'y') {
```

```
            if (attempt_to_modify_contact_list(contact, is_customer, false))
                printf("Contact succesfully changed\n");
            else
                printf("Operation failed\n");
        } else {
            return;
        }
    } else {
        printf("Operation failed\n");
    }

    if (show_prompt) {
        printf("Press enter key to get back to menu ...\n");
        getchar();
    }
}

static bool attempt_report_order(unsigned int order_id)
{
    MYSQL_STMT *stmt;
    MYSQL_BIND param[2];
    struct order_info order;
    int i = 0;
    int status;

    int flags[] = {
        LEADING_ZERO_BITMASK_IDX_0,
        LEADING_ZERO_BITMASK_IDX_0,
        0
    };

    char *messages[] = {
```

```
    "\nOrder info:",
    "\n\nInvolved species:",
    "\n\nEconiomic details:"
  };

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

  param[0].buffer_type = MYSQL_TYPE_VAR_STRING; // IN var_cliente       VARCHAR(16)
  param[0].buffer = curr_customer.code;
  param[0].buffer_length = strlen(curr_customer.code);

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

  if(!exec_sp(&stmt, param, "call report_ordine(?, ?)"))
    return false;

  do {
    if(conn->server_status & SERVER_PS_OUT_PARAMS)
      goto next;

    if (!dump_result_set(stmt, messages[i], flags[i])) {
      CLOSE_AND_RETURN(false, stmt);
    }

    ++i;
next:
    status = mysql_stmt_next_result(stmt);
    if (status > 0) {
      print_stmt_error(stmt, "Unexpected condition");
```

```
            CLOSE_AND_RETURN(false, stmt);
        }

    } while (status == 0);

    mysql_stmt_close(stmt);
    return true;
}

static void report_order(void)
{
    char buffer_for_integer[BUFFSIZE_XS];
    unsigned int order_id;

    memset(buffer_for_integer, 0, sizeof(buffer_for_integer));

    init_screen(false);

    printf("*** View order details ***\n");
    printf("Customer code.....: %s\n", curr_customer.code);

    if (ask_for_tips("Do you wanna see a report of your orders", 0)) {
        if (!attempt_report_orders_short(false))
            printf("Operation failed\n");

        putchar('\n');
    }

    printf("Insert order id.....: ");
    get_input(BUFFSIZE_XS, buffer_for_integer, false, true);
    order_id = strtol(buffer_for_integer, NULL, 10);
```

```
    putchar('\n');

    if (!attempt_report_order(order_id))
        printf("Operation failed\n");

    printf("\nPress enter key to get back to menu ...\n");
    getchar();
}

static void order_management_menu(void)
{
    char choice;

    while (true) {
        init_screen(false);

        printf("*** [ORDER MANAGEMENT] What do you wanna do? ***\n\n");
        printf("1) Open a new order\n");
        printf("2) Add a species to already opened order\n");
        printf("3) Remove a species from an order not closed yet\n");
        printf("4) Change the number of plants belonging to a species in an order\n");
        printf("5) Finalize an order\n");
        printf("6) Search a species by common name\n");
        printf("7) View order details\n");
        printf("q) Back to main menu\n");

    choice = multi_choice("Pick an option", "1234567q", 8);

        switch (choice) {
        case '1': open_order(); break;
        case '2': exec_op_on_order(true); break;
        case '3': remove_spec_from_order(); break;
```

```
        case '4': exec_op_on_order(false); break;
        case '5': finalize_order(); break;
        case '6': search_species(); break;
        case '7': report_order(); break;
        case 'q': return;
        default:
            fprintf(stderr, "Invalid condition at %s:%d\n", __FILE__, __LINE__);
            abort();
        }
    }
}

static void profile_management_menu(void)
{
    char choice;

    while (true) {
        init_screen(false);

        printf("*** [PROFILE MANAGEMENT] What do you wanna do? ***\n\n");
        printf("1) Update your residential address\n");
        printf("2) Update your billing address\n");
        printf("3) Add a contact to your list\n");
        printf("4) Remove contact from your list\n");
        printf("5) Change your favourite contact\n");
        printf("q) Back to main menu\n");

        choice = multi_choice("Pick an option", "12345q", 6);

        switch (choice) {
        case '1': update_addr(true); break;
        case '2': update_addr(false); break;
```

```
        case '3': add_contact(true, true); break;
        case '4': modify_contact_list(true, true); break;
        case '5': modify_contact_list(true, false); break;
        case 'q': return;
        default:
            fprintf(stderr, "Invalid condition at %s:%d\n", __FILE__, __LINE__);
            abort();
        }
    }
}

static void referent_management_menu(void)
{
    char choice;

    while (true) {
        init_screen(false);

        printf("*** [REFERENT MANAGEMENT] What do you wanna do? ***\n\n");
        printf("1) Add a contact to your referent list\n");
        printf("2) Remove contact from your referent list\n");
        printf("3) Change favourite contact of your referent\n");
        printf("q) Back to main menu\n");

        choice = multi_choice("Pick an option", "1234", 4);

        switch (choice){
        case '1': add_contact(false, true); break;
        case '2': modify_contact_list(false, true); break;
        case '3': modify_contact_list(false, false); break;
        case 'q': return;
        default:
```

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

static void main_menu(void)
{
    char choice;
    if (curr_customer.is_private) {
        while (true) {
            init_screen(true);
            printf("Welcome %s (%s)\n\n", curr_customer.username, curr_customer.code);
            printf("*** What do you wanna do? ***\n\n");
            printf("1) Orders management\n");
            printf("2) Profile management\n");
            printf("p) Change password\n");
            printf("q) Quit\n");

            choice = multi_choice("Pick an option", "12pq", 4);

            switch (choice) {
            case '1': order_management_menu(); break;
            case '2': profile_management_menu(); break;
            case 'p': change_password(curr_customer.username); break;
            case 'q': printf("Bye bye!\n\n\n"); return;
            default:
                fprintf(stderr, "Invalid condition at %s:%d\n", __FILE__, __LINE__);
                abort();
            }
        }
    } else {
```

```
        while (true) {
            init_screen(true);
            printf("Welcome %s (%s)\n\n", curr_customer.username, curr_customer.code);
            printf("*** What do you wanna do? ***\n\n");
            printf("1) Orders management\n");
            printf("2) Profile management\n");
            printf("3) Referent management\n");
            printf("4) Change password\n");
            printf("q) Quit\n");

            choice = multi_choice("Pick an option", "1234q", 5);

            switch (choice) {
            case '1': order_management_menu(); break;
            case '2': profile_management_menu(); break;
            case '3': referent_management_menu(); break;
            case '4': change_password(curr_customer.username); break;
            case 'q': printf("Bye bye!\n\n\n"); return;
            default:
                fprintf(stderr, "Invalid condition at %s:%d\n", __FILE__, __LINE__);
                abort();
            }
        }
    }
}

void run_as_customer(char *username, char *customer_code, bool is_private, bool first_access)
{
    char path[BUFFSIZE_M];
    struct configuration cnf;
    char choice;
```

```
    memset(path, 0, sizeof(path));
    memset(&curr_customer, 0, sizeof(curr_customer));
    memset(&cnf, 0, sizeof(cnf));

    memcpy(curr_customer.code, customer_code, BUFFSIZE_XS);
    memcpy(curr_customer.username, username, BUFFSIZE_L);
    curr_customer.is_private = is_private;

    snprintf(path, BUFFSIZE_M, "config/%s.user", (is_private) ? "pcs" : "rcs");

    if (parse_config(path, &cnf, "=")) {
        fprintf(stderr, "Invalid configuration file selected (%s)\n", (is_private) ? "PCS" : "RCS");
        exit(EXIT_FAILURE);
    }

    if(mysql_change_user(conn, cnf.username, cnf.password, cnf.database)) {
        fprintf(stderr, "Unable to switch privileges\n");
        exit(EXIT_FAILURE);
    }

    if (first_access) {
        choice = multi_choice("Do you wanna insert a contact.....?", "yn", 2);

        if (choice == 'y') {
            add_contact(true, false);
            sleep(1.5);
        }
    }

    main_menu();
}
```

FILE: defines.h

```
#pragma once

#include <stdio.h>
#include <mysql.h>
#include <stdbool.h>

/* dimensione effettiva +1 (per '\0') */
#define BUFFSIZE_XS 17
#define BUFFSIZE_S  33
#define BUFFSIZE_M  65
#define BUFFSIZE_L  129
#define BUFFSIZE_XL 257

/* bitmask */
#define LEADING_ZERO_BITMASK_IDX_0 1
#define LEADING_ZERO_BITMASK_IDX_1 2
#define LEADING_ZERO_BITMASK_IDX_2 4

#define CLOSE_AND_RETURN(retval, obj) \
    mysql_stmt_close(obj); \
            return (retval);



extern MYSQL *conn;



struct configuration {
    char host[BUFFSIZE_L];
    char username[BUFFSIZE_L];
    char password[BUFFSIZE_L];
    unsigned int port;
```

```
    char database[BUFFSIZE_L];
};



bool parse_config(const char *path, struct configuration *conf, const char *delimiter);
void print_error(MYSQL *conn, char *message);
size_t get_input(unsigned int length, char *string, bool hide, bool not_null);
void init_screen(bool);
bool setup_prepared_stmt(MYSQL_STMT **stmt, char *statement, MYSQL *conn);
void print_stmt_error (MYSQL_STMT *stmt, char *message);
char multi_choice(const char *question, const char *choices, int no_choices);
void run_as_customer(char *username, char *customer_code, bool is_private, bool first_access);
void run_as_manager(char *username);
void run_as_chief_of_staff(char *username);
void run_as_warehouse_clerk(char *username);
void run_as_order_processor(char *username);
bool dump_result_set(MYSQL_STMT *stmt, char *title, int leading_zeros_bitmask);
void change_password(char *username);
void species_tips(unsigned int dots);
bool ask_for_tips(const char *message, unsigned int dots);
int format_prompt(char *dest, size_t length, const char *src, unsigned int dots);
bool attempt_search_species(bool only_flowery, char *name);
bool exec_sp(MYSQL_STMT **stmt_ptr, MYSQL_BIND *param, char *sp_name);
bool fetch_res_sp(MYSQL_STMT *stmt, MYSQL_BIND *param);
```

FILE: inout.c

```
#include <unistd.h>
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include <termios.h>
#include <signal.h>
```

```
#include <stdbool.h>

static volatile sig_atomic_t signo;

static void handler(int s) { signo = s; }

size_t get_input(unsigned int length, char *string, bool hide, bool not_null)
{
   char c;
   unsigned int i;
   struct sigaction sa, savealrm, saveint, savehup, savequit, saveterm, savetstp, savettin, savettou;
   struct termios term, oterm;

   if (hide) {
      (void) fflush(stdout);
      (void) sigemptyset(&sa.sa_mask);
      sa.sa_flags = SA_INTERRUPT;
      sa.sa_handler = handler;

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

      if (tcgetattr(fileno(stdin), &oterm) == 0) {
         (void) memcpy(&term, &oterm, sizeof(struct termios));
         term.c_lflag &= ~(ECHO|ECHONL);
         (void) tcsetattr(fileno(stdin), TCSAFLUSH, &term);
```

```
        } else {
            (void) memset(&term, 0, sizeof(struct termios));
            (void) memset(&oterm, 0, sizeof(struct termios));
        }
    }

read_loop:
    for (i = 0; i < length; ++i) {
        (void) fread(&c, sizeof(char), 1, stdin);
        if (c == '\n') {
            string[i] = '\0';
            break;
        } else {
            string[i] = c;
        }
    }

    if (not_null && i == 0)
        goto read_loop;

    if (i == length - 1)
        string[i] = '\0';

    if (strlen(string) >= length) {
        do {
            c = getchar();
        } while (c != '\n');
    }

    if (hide) {
        (void) write(fileno(stdout), "\n", 1);
```

```
        (void) tcsetattr(fileno(stdin), TCSAFLUSH, &oterm);
        (void) sigaction(SIGALRM, &savealrm, NULL);
        (void) sigaction(SIGINT, &saveint, NULL);
        (void) sigaction(SIGHUP, &savehup, NULL);
        (void) sigaction(SIGQUIT, &savequit, NULL);
        (void) sigaction(SIGTERM, &saveterm, NULL);
        (void) sigaction(SIGTSTP, &savetstp, NULL);
        (void) sigaction(SIGTTIN, &savettin, NULL);
        (void) sigaction(SIGTTOU, &savettou, NULL);

        if (signo)
            (void) raise(signo);
    }

    return i;
}

char multi_choice(const char *question, const char *choices, int no_choices)
{
    char choices_str[2 * no_choices * sizeof(char)];
    int i, j = 0;

    for (i = 0; i < no_choices; ++i) {
        choices_str[j++] = choices[i];
        choices_str[j++] = '/';
    }

    choices_str[j-1] = '\0';

    while (true) {
        printf("%s [%s]: ", question, choices_str);
```

```
        char c;
        get_input(1, &c, false, true);
        c = tolower(c);

        for (i = 0; i < no_choices; ++i) {
            if(c == tolower(choices[i]))
                return c;
        }

        printf("Sorry not compliant input, please retry!\n");
    }
}
```

FILE: main.c

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



enum role {
    PCS,    // Customer (private)
    RCS,    // Customer (retailer)
    WHC,    // Warehouse clerk
    OPC,    // Order processor
    MNG,    // Manager
    COS,    // Chief of staff
    ERR     // ERROR
};
```

```
struct credentials {
    char username[BUFFSIZE_L];
    char password[BUFFSIZE_L];
};

struct customer_signup_params {
    struct credentials *creds;
    char code[BUFFSIZE_XS];
    char name[BUFFSIZE_S];
    char residential_address[BUFFSIZE_M];
    char billing_address[BUFFSIZE_M];
    char referent_first_name[BUFFSIZE_S];
    char referent_last_name[BUFFSIZE_S];
};



MYSQL *conn;



static enum role attempt_login(struct credentials *cred, char *identifier)
{
    MYSQL_STMT *stmt;
    MYSQL_BIND param[4];
    int role;

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

    param[0].buffer_type = MYSQL_TYPE_VAR_STRING; // IN var_username VARCHAR(128)
    param[0].buffer = cred->username;
    param[0].buffer_length = strlen(cred->username);

    param[1].buffer_type = MYSQL_TYPE_VAR_STRING; // IN var_password VARCHAR(128)
```

```
    param[1].buffer = cred->password;
    param[1].buffer_length = strlen(cred->password);

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

    param[3].buffer_type = MYSQL_TYPE_VAR_STRING; // OUT var_codice_cliente VARCHAR(16)
    param[3].buffer = identifier;
    param[3].buffer_length = BUFFSIZE_XS * sizeof(char);

    if (!exec_sp(&stmt, param, "call login(?, ?, ?, ?)"))
        return ERR;

    param[0].buffer_type = MYSQL_TYPE_LONG; // OUT var_ruolo INT
    param[0].buffer = &role;
    param[0].buffer_length = sizeof(role);

    param[1].buffer_type = MYSQL_TYPE_VAR_STRING; // OUT var_codice_cliente VARCHAR(16)
    param[1].buffer = identifier;
    param[1].buffer_length = BUFFSIZE_XS * sizeof(char);

    if (!fetch_res_sp(stmt, param))
        return ERR;

    mysql_stmt_close(stmt);
    return role;
}

static bool attempt_signup(struct customer_signup_params *cst, bool is_private)
{
```

```
    MYSQL_STMT *stmt;
    MYSQL_BIND param[(is_private ? 6 : 8)];

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

    param[0].buffer_type = MYSQL_TYPE_VAR_STRING; // IN var_codice CHAR(16) / CHAR(11)
    param[0].buffer = cst->code;
    param[0].buffer_length = strlen(cst->code);

    param[1].buffer_type = MYSQL_TYPE_VAR_STRING; // IN var_nome VARCHAR(32)
    param[1].buffer = cst->name;
    param[1].buffer_length = strlen(cst->name);

    param[2].buffer_type = MYSQL_TYPE_VAR_STRING; // IN var_res VARCHAR(64)
    param[2].buffer = cst->residential_address;
    param[2].buffer_length = strlen(cst->residential_address);

    if (strlen(cst->billing_address) > 0) {
        param[3].buffer_type = MYSQL_TYPE_VAR_STRING; // IN var_fat VARCHAR(64)
        param[3].buffer = cst->billing_address;
        param[3].buffer_length = strlen(cst->billing_address);
    } else {
        param[3].buffer_type = MYSQL_TYPE_NULL; // IN var_fat VARCHAR(64)
        param[3].buffer = NULL;
        param[3].buffer_length = 0;
    }

    param[4].buffer_type = MYSQL_TYPE_VAR_STRING; // IN var_user VARCHAR(128)
    param[4].buffer = (cst->creds)->username;
    param[4].buffer_length = strlen((cst->creds)->username);
```

```
    param[5].buffer_type = MYSQL_TYPE_VAR_STRING; // IN var_pass VARCHAR(128)
    param[5].buffer = (cst->creds)->password;
    param[5].buffer_length = strlen((cst->creds)->password);

    if (!is_private) {
        param[6].buffer_type = MYSQL_TYPE_VAR_STRING; // IN var_nome_ref VARCHAR(32)
        param[6].buffer = cst->referent_first_name;
        param[6].buffer_length = strlen(cst->referent_first_name);

        param[7].buffer_type = MYSQL_TYPE_VAR_STRING; // IN var_cognome_ref VARCHAR(32)
        param[7].buffer = cst->referent_last_name;
        param[7].buffer_length = strlen(cst->referent_last_name);
    }

    if (is_private) {
        if (!exec_sp(&stmt, param, "call registra_privato(?, ?, ?, ?, ?, ?)"))
            return false;
    } else {
        if (!exec_sp(&stmt, param, "call registra_rivendita(?, ?, ?, ?, ?, ?, ?, ?)"))
            return false;
    }

    mysql_stmt_close(stmt);
    return true;
}

static bool attempt_change_password(char *username, char *old_passwd, char *new_passwd)
{
    MYSQL_STMT *stmt;
    MYSQL_BIND param[3];
```

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

    param[0].buffer_type = MYSQL_TYPE_VAR_STRING; // IN var_username VARCHAR(128)
    param[0].buffer = username;
    param[0].buffer_length = strlen(username);

    param[1].buffer_type = MYSQL_TYPE_VAR_STRING; // IN var_vecchia_password VARCHAR(128)
    param[1].buffer = old_passwd;
    param[1].buffer_length = strlen(old_passwd);

    param[2].buffer_type = MYSQL_TYPE_VAR_STRING; // IN var_nuova_password VARCHAR(128)
    param[2].buffer = new_passwd;
    param[2].buffer_length = strlen(new_passwd);

    if (!exec_sp(&stmt, param, "call modifica_password(?, ?, ?)"))
        return false;

    mysql_stmt_close(stmt);
    return true;
}

static bool login_manager(void)
{
    char client_identifier[BUFFSIZE_XS];
    struct credentials cred;
    enum role role;
    char choice;

    memset(&client_identifier, 0, sizeof(client_identifier));
    memset(&cred, 0, sizeof(cred));
```

```
    init_screen(false);

    printf("Insert username: ");
    get_input(BUFFSIZE_L, cred.username, false, true);
    printf("Insert password: ");
    get_input(BUFFSIZE_L, cred.password, true, true);

    role = attempt_login(&cred, client_identifier);

    switch (role) {
    case PCS: run_as_customer(cred.username, client_identifier, true, false); break;
    case RCS: run_as_customer(cred.username, client_identifier, false, false); break;
    case MNG: run_as_manager(cred.username); break;
    case COS: run_as_chief_of_staff(cred.username); break;
    case WHC: run_as_warehouse_clerk(cred.username); break;
    case OPC: run_as_order_processor(cred.username); break;
    case ERR: printf("Login failed!\n"); break;
    default:
        fprintf(stderr, "Invalid condition at %s:%d\n", __FILE__, __LINE__);
        abort();
    }

    if (role == ERR) {
        choice = multi_choice("Do you wanna quit? ", "yn", 2);
        return (choice == 'y');
    }

    return true;
}

static char *strupper(char *str)
{
```

```
    for (unsigned int i = 0; i < strlen(str); ++i)
        str[i] = toupper(str[i]);

    return str;
}

static bool signup_manager(void)
{
    char password_check[BUFFSIZE_L];
    struct customer_signup_params cst;
    struct credentials creds;
    char modality;
    char choice;

    memset(&cst, 0, sizeof(cst));
    memset(&creds, 0, sizeof(creds));
    memset(password_check, 0, sizeof(password_check));

    init_screen(false);

    modality = multi_choice("Are you a [p]rivate or [r]etailer..?", "pr", 2);

    printf("Insert username.....: ");
    get_input(BUFFSIZE_L, creds.username, false, true);

retype_pass:
    printf("Insert password.....: ");
    get_input(BUFFSIZE_L, creds.password, true, true);

    printf("Confirm password.....: ");
    get_input(BUFFSIZE_L, password_check, true, true);
```

```
if (strcmp(creds.password, password_check) != 0) {
    printf("Mismatch password, please retry!\n");
    goto retype_pass;
}

if (modality == 'p') {
    printf("Insert fiscal code.....: ");
    get_input(16, cst.code, false, true);
} else {
    printf("Insert VAT code.....: ");
    get_input(11, cst.code, false, true);
}

printf("Insert your name.....: ");
get_input(BUFFSIZE_S, cst.name, false, true);

printf("Insert your residential address.....: ");
get_input(BUFFSIZE_M, cst.residential_address, false, true);

printf("Insert your billing address (default null): ");
get_input(BUFFSIZE_M, cst.billing_address, false, false);

if (modality == 'r') {
    printf("Insert referent first name.....: ");
    get_input(BUFFSIZE_S, cst.referent_first_name, false, true);

    printf("Insert referent last name:.....:");
    get_input(BUFFSIZE_S, cst.referent_last_name, false, true);
}

cst.creds = &creds;
```

```
    if (!attempt_signup(&cst, (modality == 'p'))) {
        printf("Signup failed!\n");
        choice = multi_choice("Do you wanna quit?", "yn", 2);
        return (choice == 'y');
    }

    run_as_customer(creds.username, strupper(cst.code), (modality == 'p'), true);
    return true;
}

void change_password(char *username)
{
    char old_passwd[BUFFSIZE_L];
    char new_passwd[BUFFSIZE_L];
    char passwd_check[BUFFSIZE_L];

    memset(old_passwd, 0, sizeof(old_passwd));
    memset(new_passwd, 0, sizeof(new_passwd));
    memset(passwd_check, 0, sizeof(passwd_check));

    init_screen(false);

    printf("*** Change password ***\n");
    printf("Customer username.....: %s\n", username);
    printf("Insert old password....: ");
    get_input(BUFFSIZE_L, old_passwd, true, true);

retype_pass:
    printf("Insert new password....: ");
    get_input(BUFFSIZE_L, new_passwd, true, true);

    printf("Retype new password....: ");
```

```
    get_input(BUFFSIZE_L, passwd_check, true, true);

    if (strcmp(new_passwd, passwd_check) != 0) {
        printf("Mismatch password, please retry!\n");
        goto retype_pass;
    }

    if (attempt_change_password(username, old_passwd, new_passwd))
        printf("Password has been changed!\n");
    else
        printf("Operation failed\n");

    printf("Press enter key to get back to menu ...\n");
    getchar();
}

int format_prompt(char *dest, size_t length, const char *src, unsigned int dots)
{
    int len = snprintf(dest, length, src);

    for (unsigned int i = 0; i < dots; ++i)
        dest[len + i] = '.';

    dest[len + dots] = '?';

    return len + dots;
}

bool ask_for_tips(const char *message, unsigned int dots)
{
    char prompt[BUFFSIZE_XL];
    char choice;
```

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

    putchar('\n');

    format_prompt(prompt, BUFFSIZE_XL, message, dots);

    choice = multi_choice(prompt, "yn", 2);
    return (choice == 'y');
}

bool attempt_search_species(bool only_flowery, char *name)
{
    MYSQL_STMT *stmt;
    MYSQL_BIND param[2];
    char prompt[BUFFSIZE_L];

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

    param[0].buffer_type = MYSQL_TYPE_VAR_STRING; // IN var_nome_comune VARCHAR(64)
    param[0].buffer = name;
    param[0].buffer_length = strlen(name);

    param[1].buffer_type = MYSQL_TYPE_TINY; // IN var_status TINYINT
    param[1].buffer = &(only_flowery);
    param[1].buffer_length = sizeof(only_flowery);

    if (!exec_sp(&stmt, param, "call visualizza_dettagli_specie(?, ?)"))
        return false;
```

```
    if (strlen(name) > 0)
        snprintf(prompt, BUFFSIZE_L, "\nSearch results for \'%s\':", name);
    else
        snprintf(prompt, BUFFSIZE_L, "\nSearch results:");

    if (!dump_result_set(stmt, prompt, LEADING_ZERO_BITMASK_IDX_0)) {
        CLOSE_AND_RETURN(false, stmt);
    }

    mysql_stmt_close(stmt);
    return true;
}

bool exec_sp(MYSQL_STMT **stmt_ptr, MYSQL_BIND *param, char *sp_name)
{
    if(!setup_prepared_stmt(stmt_ptr, sp_name, conn)) {
        print_stmt_error(*stmt_ptr, "Unable to initialize the statement\n");
        return false;
    }

    if (mysql_stmt_bind_param(*stmt_ptr, param) != 0) {
        print_stmt_error(*stmt_ptr, "Could not bind parameters for the statement");
        CLOSE_AND_RETURN(false, *stmt_ptr);
    }

    if (mysql_stmt_execute(*stmt_ptr) != 0) {
        print_stmt_error(*stmt_ptr, "Could not execute the statement");
        CLOSE_AND_RETURN(false, *stmt_ptr);
    }

    return true;
}
```

```
bool fetch_res_sp(MYSQL_STMT *stmt, MYSQL_BIND *param)
{
   if(mysql_stmt_bind_result(stmt, param)) {
      print_stmt_error(stmt, "Could not retrieve output parameter");
      CLOSE_AND_RETURN(false, stmt);
   }

   if(mysql_stmt_fetch(stmt)) {
      print_stmt_error(stmt, "Could not buffer results");
      CLOSE_AND_RETURN(false, stmt);
   }

   return true;
}

int main(void)
{
   char choice;
   struct configuration cnf;
   MYSQL *ret;

   memset(&cnf, 0, sizeof(cnf));

   if (parse_config("config/nrg.user", &cnf, "=")) {
      fprintf(stderr, "Invalid configuration file selected (NRG)\n");
      exit(EXIT_FAILURE);
   }

   conn = mysql_init(NULL);

   if (conn == NULL) {
```

```
    fprintf(stderr, "Out of memory, connection was not established.");
    exit(EXIT_FAILURE);
  }

  ret = mysql_real_connect(conn, cnf.host, cnf.username, cnf.password, cnf.database,
                           cnf.port, NULL, CLIENT_MULTI_STATEMENTS |
CLIENT_MULTI_RESULTS);
  if (ret == NULL) {
    print_error(conn, "Something went wrong, connection was not established.");
    mysql_close (conn);
    exit(EXIT_FAILURE);
  }

  while (true) {
    init_screen(false);
    choice = multi_choice("Do you wanna [l]ogin or [s]ignup?", "ls", 2);

    if (choice == 'l') {
      if (login_manager())
        break;
    }

    else if (choice == 's') {
      if (signup_manager())
        break;
    }

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

```
    mysql_close(conn);
    mysql_library_end();
    exit(EXIT_SUCCESS);
}
```

FILE manager.c

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

struct insert_species_sp_params {
    char common_name[BUFFSIZE_M];
    char latin_name[BUFFSIZE_M];
    signed char in_or_out;
    char coloring[BUFFSIZE_S];
    signed char exotic;
    unsigned int stock;
    char price[BUFFSIZE_XS];
 };



static char curr_user[BUFFSIZE_L];



static bool attempt_show_colors(unsigned int species_code)
{
```

```
    MYSQL_STMT *stmt;
    MYSQL_BIND param[1];

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

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

    if(!exec_sp(&stmt, param, "call visualizza_colorazioni(?)"))
        return false;

    if (!dump_result_set(stmt, "\nColors available for selected species:", 0)) {
        CLOSE_AND_RETURN(false, stmt);
    }

    mysql_stmt_close(stmt);
    return true;
}

static int check_price(char *inserted_price, char *strerror, size_t strerror_length)
{
    regex_t reg;
    char err_mess[BUFFSIZE_L];
    int ret;

    ret = regcomp(&reg, "^([0-9]{0,5}((.)[0-9]{0,2}))$", REG_EXTENDED);
    if (ret) {
        regerror(ret, &reg, err_mess, sizeof(err_mess));
        snprintf(strerror, strerror_length, "Unable to compile regexp to check inserted price\nThe error was: %s\n", err_mess);
        return -1;
```

```
    }

    ret = regexec(&reg, inserted_price, 0, NULL, 0);
    if (ret == 0) {
        regfree(&reg);
        return 1;
    } else if (ret == REG_NOMATCH) {
        regfree(&reg);
        return 0;
    } else {
        regerror(ret, &reg, err_mess, sizeof(err_mess));
        snprintf(strerror, strerror_length, "Unable to exec regexp to check inserted price\nThe error was: %s\n", err_mess);
        return -1;
    }
}

static unsigned int attempt_insert_species(struct insert_species_sp_params *input)
{
    MYSQL_STMT *stmt;
    MYSQL_BIND param[8];
    unsigned int code;

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

    param[0].buffer_type = MYSQL_TYPE_VAR_STRING; // IN var_nome_comune VARCHAR(64)
    param[0].buffer = input->common_name;
    param[0].buffer_length = strlen(input->common_name);

    param[1].buffer_type = MYSQL_TYPE_VAR_STRING; // IN var_nome_latino VARCHAR(64)
    param[1].buffer = input->latin_name;
    param[1].buffer_length = strlen(input->latin_name);
```

```
    param[2].buffer_type = MYSQL_TYPE_TINY; // IN var_int_est TINYINT
    param[2].buffer = &(input->in_or_out);
    param[2].buffer_length = sizeof(input->in_or_out);

    if (strlen(input->coloring) > 0) {
        param[3].buffer_type = MYSQL_TYPE_VAR_STRING; // IN var_colorazione VARCHAR(32)
        param[3].buffer = input->coloring;
        param[3].buffer_length = strlen(input->coloring);
    } else {
        param[3].buffer_type = MYSQL_TYPE_NULL; // IN var_colorazione VARCHAR(32)
        param[3].buffer = NULL;
        param[3].buffer_length = 0;
    }

    param[4].buffer_type = MYSQL_TYPE_TINY; // IN var_esotica TINYINT
    param[4].buffer = &(input->exotic);
    param[4].buffer_length = sizeof(input->exotic);

    param[5].buffer_type= MYSQL_TYPE_LONG; // IN var_giacenza INT
    param[5].buffer = &(input->stock);
    param[5].buffer_length = sizeof(input->stock);

    param[6].buffer_type = MYSQL_TYPE_NEWDECIMAL; // IN var_prezzo DECIMAL(7, 2)
    param[6].buffer = input->price;
    param[6].buffer_length = strlen(input->price);

    param[7].buffer_type = MYSQL_TYPE_LONG; // OUT var_codice INT
    param[7].buffer = &code;
    param[7].buffer_length = sizeof(code);
```

```
    if (!exec_sp(&stmt, param, "call inserisci_nuova_specie(?, ?, ?, ?, ?, ?, ?, ?)"))
        return 0;

    param[0].buffer_type = MYSQL_TYPE_LONG; // OUT var_codice INT
    param[0].buffer = &code;
    param[0].buffer_length = sizeof(code);

    if (!fetch_res_sp(stmt, param))
        return 0;

    mysql_stmt_close(stmt);
    return code;
}

static void insert_a_species(void)
{
    struct insert_species_sp_params params;
    char buffer_for_integer[BUFFSIZE_XS];
    char strerror[BUFFSIZE_XL];
    unsigned int species_code;
    char choice;
    int ret;

    memset(&params, 0, sizeof(params));
    memset(buffer_for_integer, 0, sizeof(buffer_for_integer));
    memset(strerror, 0, sizeof(strerror));

    init_screen(false);

    printf("*** Insert a new species ***\n");

    printf("Insert common name.....: ");
```

```
    get_input(BUFFSIZE_M, params.common_name, false, true);

    printf("Insert latin name.....: ");
    get_input(BUFFSIZE_M, params.latin_name, false, true);

    choice = multi_choice("Which kind of species is it? [o]utdoor or [i]ndoor", "oi", 2);
    params.in_or_out = (choice == 'i');

    choice = multi_choice("Which kind of species is it? [g]reen or [f]lowery.", "gf", 2);
    if (choice == 'f') {
        printf("Insert default coloring.....: ");
        get_input(BUFFSIZE_S, params.coloring, false, true);
    }

    choice = multi_choice("Is it exotic.....?", "yn", 2);
    params.exotic = (choice == 'y') ? 1 : 0;

    printf("Insert initial stock.....: ");
    get_input(BUFFSIZE_XS, buffer_for_integer, false, true);
    params.stock = strtol(buffer_for_integer, NULL, 10);

insert_price:
    printf("Insert price (#####.##).....: ");
    get_input(BUFFSIZE_XS, params.price, false, true);
    ret = check_price(params.price, strerror, BUFFSIZE_XL);

    if (ret == 0) {
        printf("Not compliant input please retry\n");
        goto insert_price;
    }

    if (ret == -1) {
```

```
        fprintf(stderr, strerror);
        goto exit;
    }

    putchar('\n');

    species_code = attempt_insert_species(&params);
    if (species_code > 0)
        printf("New species inserted (CODE: %010u)\n", species_code);
    else
        printf("Operation failed\n");

exit:
    printf("Press enter key to get back to menu ...\n");
    getchar();
}

static int attempt_remove_species(unsigned int species_code)
{
    MYSQL_STMT *stmt;
    MYSQL_BIND param[2];
    int affected_rows;

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

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

    param[1].buffer_type = MYSQL_TYPE_LONG; // OUT var_eliminazione_effettiva INT
    param[1].buffer = &affected_rows;
    param[1].buffer_length = sizeof(affected_rows);
```

```
    if (!exec_sp(&stmt, param, "call rimuovi_specie_di_pianta(?, ?)"))
        return -1;

    param[0].buffer_type = MYSQL_TYPE_LONG; // OUT var_eliminazione_effettiva INT
    param[0].buffer = &affected_rows;
    param[0].buffer_length = sizeof(affected_rows);

    if (!fetch_res_sp(stmt, param))
        return -1;

    mysql_stmt_close(stmt);
    return affected_rows;
}

static void remove_a_species(void)
{
    char buffer_for_integer[BUFFSIZE_XS];
    char spec_name[BUFFSIZE_M];
    unsigned int species_code;
    int ret;

    memset(buffer_for_integer, 0, sizeof(buffer_for_integer));
    memset(spec_name, 0, sizeof(spec_name));

    init_screen(false);

    printf("*** Remove a species ***\n");

    if (ask_for_tips("Do you wanna search species by name to find the right code", 0)) {
        printf("\nInsert the name to filter on (default all).....: ");
        get_input(BUFFSIZE_M, spec_name, false, false);
```

```
        if (!attempt_search_species(false, spec_name))
            printf("Operation failed\n");

        putchar('\n');
    }

    printf("Insert species code.....: ");
    get_input(BUFFSIZE_XS, buffer_for_integer, false, true);
    species_code = strtol(buffer_for_integer, NULL, 10);

    putchar('\n');

    ret = attempt_remove_species(species_code);

    if (ret > 0)
        printf("Species %u succesfully deleted\n", species_code);
    else if (ret == 0)
        printf("Nothing has changed (species %u not found)\n", species_code);
    else
        printf("Operation failed\n");

    printf("Press enter key to get back to menu ...\n");
    getchar();
}

static int attempt_add_coloring(unsigned int species_code, char *coloring)
{
    MYSQL_STMT *stmt;
    MYSQL_BIND param[2];

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

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

    param[1].buffer_type = MYSQL_TYPE_VAR_STRING; // IN var_colore VARCHAR(32)
    param[1].buffer = coloring;
    param[1].buffer_length = strlen(coloring);

    if(!exec_sp(&stmt, param, "call aggiungi_colorazione(?, ?)"))
        return false;

    if (!dump_result_set(stmt, "\nUpdated coloring list:", 0)) {
        CLOSE_AND_RETURN(false, stmt);
    }

    mysql_stmt_close(stmt);
    return true;
}

static void add_coloring(void)
{
    unsigned int species_code;
    char coloring[BUFFSIZE_S];
    char buffer_for_integer[BUFFSIZE_XS];
    char spec_name[BUFFSIZE_M];

    memset(coloring, 0, sizeof(coloring));
    memset(buffer_for_integer, 0, sizeof(buffer_for_integer));
    memset(spec_name, 0, sizeof(spec_name));

    init_screen(false);
```

```
printf("*** Add a coloring for a flowering species ***\n");

if (ask_for_tips("Do you wanna search flowery species by name to find the right code", 0)) {
    printf("\nInsert the name to filter on (default all).....: ");
    get_input(BUFFSIZE_M, spec_name, false, false);

    if (!attempt_search_species(true, spec_name))
        printf("Operation failed\n");

    putchar('\n');
}
printf("Insert species code.....: ");
get_input(BUFFSIZE_XS, buffer_for_integer, false, true);
species_code = strtol(buffer_for_integer, NULL, 10);

if (ask_for_tips("Do you wanna see a report of available colors", 21)) {
    if (!attempt_show_colors(species_code))
        printf("Operation failed\n");

    putchar('\n');;
}
printf("Insert coloring for this species.....: ");
get_input(BUFFSIZE_S, coloring, false, true);

putchar('\n');

if (attempt_add_coloring(species_code, coloring))
    printf("\nColoring \"%s\" for %010u succesfully added\n", coloring, species_code);
else
    printf("\nOperation failed\n");
```

```
    printf("Press enter key to get back to menu ...\n");
    getchar();
}

static int attempt_remove_coloring(unsigned int species_code, char *coloring)
{
    MYSQL_STMT *stmt;
    MYSQL_BIND param[3];
    int affected_rows;

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

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

    param[1].buffer_type = MYSQL_TYPE_VAR_STRING; // IN var_colore VARCHAR(32)
    param[1].buffer = coloring;
    param[1].buffer_length = strlen(coloring);

    param[2].buffer_type = MYSQL_TYPE_LONG; // OUT var_eliminazione_effettiva INT
    param[2].buffer = &affected_rows;
    param[2].buffer_length = sizeof(affected_rows);

    if (!exec_sp(&stmt, param, "call rimuovi_colorazione(?, ?, ?)"))
        return -1;

    param[0].buffer_type = MYSQL_TYPE_LONG; // OUT var_eliminazione_effettiva INT
    param[0].buffer = &affected_rows;
    param[0].buffer_length = sizeof(affected_rows);

    if (!fetch_res_sp(stmt, param))
```

```
        return -1;

    mysql_stmt_close(stmt);
    return affected_rows;
}

static void remove_coloring(void)
{
    unsigned int species_code;
    char coloring[BUFFSIZE_S];
    char buffer_for_integer[BUFFSIZE_XS];
    char spec_name[BUFFSIZE_M];
    int ret;

    memset(coloring, 0, sizeof(coloring));
    memset(buffer_for_integer, 0, sizeof(buffer_for_integer));
    memset(spec_name, 0, sizeof(spec_name));

    init_screen(false);

    printf("*** Remove a coloring from a flowering species ***\n");

    if (ask_for_tips("Do you wanna search flowery species by name to find the right code", 0)) {
        printf("\nInsert the name to filter on (default all).....: ");
        get_input(BUFFSIZE_M, spec_name, false, false);
        if (!attempt_search_species(true, spec_name))
            printf("Operation failed\n");

        putchar('\n');
    }

    printf("Insert species code.....: ");
```

```
    get_input(BUFFSIZE_XS, buffer_for_integer, false, true);
    species_code = strtol(buffer_for_integer, NULL, 10);

    if (ask_for_tips("Do you wanna see a report of available colors", 21)) {
        if (!attempt_show_colors(species_code))
            printf("Operation failed\n");

        putchar('\n');;
    }

    printf("Insert coloring to be removed.....: ");
    get_input(BUFFSIZE_S, coloring, false, true);

    putchar('\n');

    ret = attempt_remove_coloring(species_code, coloring);

    if (ret > 0)
        printf("Coloring \"%s\" succesfully removed for %010u\n", coloring, species_code);
    else if (ret == 0)
        printf("Nothing has changed (species %010u does not have this coloring)\n", species_code);
    else
        printf("Operation failed\n");

    printf("Press enter key to get back to menu ...\n");
    getchar();
}

static int attempt_change_price(unsigned int species_code, char *price)
{
    MYSQL_STMT *stmt;
    MYSQL_BIND param[3];
```

```
    int affected_rows;

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

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

    param[1].buffer_type = MYSQL_TYPE_NEWDECIMAL; // IN var_prezzo DECIMAL(7, 2)
    param[1].buffer = price;
    param[1].buffer_length = strlen(price);

    param[2].buffer_type = MYSQL_TYPE_LONG; // OUT var_aggiornamento_effettivo INT
    param[2].buffer = &affected_rows;
    param[2].buffer_length = sizeof(affected_rows);

    if (!exec_sp(&stmt, param, "call modifica_prezzo(?, ?, ?)"))
        return -1;

    param[0].buffer_type = MYSQL_TYPE_LONG; // OUT var_aggiornamento_effettivo INT
    param[0].buffer = &affected_rows;
    param[0].buffer_length = sizeof(affected_rows);

    if (!fetch_res_sp(stmt, param))
        return -1;

    mysql_stmt_close(stmt);
    return affected_rows;
}

static void change_price(void)
{
```

```
    char buffer_for_integer[BUFFSIZE_XS];
    char spec_name[BUFFSIZE_M];
    char price[BUFFSIZE_XS];
    char strerror[BUFFSIZE_XL];
    unsigned int species_code;
    int ret;

    memset(buffer_for_integer, 0, sizeof(buffer_for_integer));
    memset(strerror, 0, sizeof(strerror));
    memset(price, 0, sizeof(price));
    memset(spec_name, 0, sizeof(spec_name));

    init_screen(false);

    printf("*** Change the price of a species ***\n");

    if (ask_for_tips("Do you wanna search species by name to find the right code", 0)) {
        printf("\nInsert the name to filter on (default all).....: ");
        get_input(BUFFSIZE_M, spec_name, false, false);
        if (!attempt_search_species(false, spec_name))
            printf("Operation failed\n");

        putchar('\n');
    }
    printf("Insert species code.....: ");
    get_input(BUFFSIZE_XS, buffer_for_integer, false, true);
    species_code = strtol(buffer_for_integer, NULL, 10);

insert_price:
    printf("Insert price (#####.##).....: ");
    get_input(BUFFSIZE_XS, price, false, true);
    ret = check_price(price, strerror, BUFFSIZE_XL);
```

```
    if (ret == 0) {
        printf("Not compliant input please retry\n");
        goto insert_price;
    }

    if (ret == -1) {
        fprintf(stderr, strerror);
        goto exit;
    }

    putchar('\n');

    ret = attempt_change_price(species_code, price);

    if (ret > 0)
        printf("Price updated for %010u\n", species_code);
    else if (ret == 0)
        printf("Nothing has changed (the stored and entered prices coincide)\n");
    else
        printf("Operation failed\n");

exit:
    printf("Press enter key to get back to menu ...\n");
    getchar();
}

static bool attempt_report_species(unsigned int species_code)
{
    MYSQL_STMT *stmt;
    MYSQL_BIND param[1];
    char prompt[2 * BUFFSIZE_XL];
```

```
int i = 0;
int status;

time_t t = time(NULL);
struct tm *tm = localtime(&t);

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

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

if(!exec_sp(&stmt, param, "call report_specie(?)"))
   return false;

do {
   if(conn->server_status & SERVER_PS_OUT_PARAMS)
      goto next;

   if (i == 0) {
      if (!dump_result_set(stmt, "\nSpecies info:", LEADING_ZERO_BITMASK_IDX_0)) {
         CLOSE_AND_RETURN(false, stmt);
      }
   } else {
      snprintf(prompt, 2 * BUFFSIZE_XL, "\n\nSales details (year: %d):", (tm->tm_year + 1900));
      if (!dump_result_set(stmt, prompt, 0)) {
         CLOSE_AND_RETURN(false, stmt);
      }
   }

   ++i;
```

```
next:
        status = mysql_stmt_next_result(stmt);
        if (status > 0) {
            print_stmt_error(stmt, "Unexpected condition");
            CLOSE_AND_RETURN(false, stmt);
        }

    } while (status == 0);

    mysql_stmt_close(stmt);
    return true;
}

static void report_species(void)
{
    unsigned int species_code;
    char buffer_for_integer[BUFFSIZE_XS];
    char spec_name[BUFFSIZE_M];

    memset(buffer_for_integer, 0, sizeof(buffer_for_integer));
    memset(spec_name, 0, sizeof(spec_name));

    init_screen(false);

    printf("*** View sales trend for a chosen species ***\n");

    if (ask_for_tips("Do you wanna search species by name to find the right code", 0)) {
        printf("\nInsert the name to filter on (default all).....: ");
        get_input(BUFFSIZE_M, spec_name, false, false);

        if (!attempt_search_species(false, spec_name))
            printf("Operation failed\n");
```

```
        putchar('\n');
    }
    printf("Insert species code.....................................................: ");
    get_input(BUFFSIZE_XS, buffer_for_integer, false, true);
    species_code = strtol(buffer_for_integer, NULL, 10);

    putchar('\n');

    if (!attempt_report_species(species_code))
        printf("Operation failed\n");

    printf("\nPress enter key to get back to menu ...\n");
    getchar();
}

void run_as_manager(char *username)
{
    struct configuration cnf;
    char choice;

    memset(&cnf, 0, sizeof(cnf));
    memset(curr_user, 0, sizeof(curr_user));

    strncpy(curr_user, username, BUFFSIZE_L);

    if (parse_config("config/mng.user", &cnf, "=")) {
        fprintf(stderr, "Invalid configuration file selected (MNG)\n");
        exit(EXIT_FAILURE);
    }

    if (mysql_change_user(conn, cnf.username, cnf.password, cnf.database)) {
```

```
    fprintf(stderr, "Unable to switch privileges\n");
    exit(EXIT_FAILURE);
}

while (true) {
    init_screen(true);
    printf("Welcome %s\n\n", curr_user);
    printf("*** What do you wanna do? ***\n\n");
    printf("1) Insert a new species\n");
    printf("2) Remove a species\n");
    printf("3) Add a coloring for a flowering species\n");
    printf("4) Remove a coloring from a flowering species list\n");
    printf("5) Change the price of a species\n");
    printf("6) View sales trend for a chosen species\n");
    printf("p) Change password\n");
    printf("q) Quit\n");

    choice = multi_choice("Pick an option", "123456pq", 8);

    switch (choice) {
    case '1': insert_a_species(); break;
    case '2': remove_a_species(); break;
    case '3': add_coloring(); break;
    case '4': remove_coloring(); break;
    case '5': change_price(); break;
    case '6': report_species(); break;
    case 'p': change_password(curr_user); break;
    case 'q': printf("Bye bye!\n\n\n"); return;
    default:
        fprintf(stderr, "Invalid condition at %s:%d\n", __FILE__, __LINE__);
        abort();
    }
```

```
    }
}
```

FILE orderprocessor.c

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

struct create_pack_sp_param {
    unsigned int order_id;
    unsigned int pack_number;
    unsigned int species_code;
    unsigned int quantity;
};



static char curr_user[BUFFSIZE_L];



static bool attempt_search_species_belonging_to_order(unsigned int order_id)
{
    MYSQL_STMT *stmt;
    MYSQL_BIND param[1];

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

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

```
    if(!exec_sp(&stmt, param, "call visualizza_piante_rimanenti_da_impacchettare(?)"))
        return false;

    if (!dump_result_set(stmt, "Species belonging to selected order:",
LEADING_ZERO_BITMASK_IDX_0)) {
        CLOSE_AND_RETURN(false, stmt);
    }

    mysql_stmt_close(stmt);
    return true;
}

static bool attempt_show_status(unsigned int order_id)
{
    MYSQL_STMT *stmt;
    MYSQL_BIND param[1];

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

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

    if(!exec_sp(&stmt, param, "call visualizza_stato_ordine(?)"))
        return false;

    if (!dump_result_set(stmt, "Order info:", LEADING_ZERO_BITMASK_IDX_0)) {
        CLOSE_AND_RETURN(false, stmt);
    }

    mysql_stmt_close(stmt);
```

```
    return true;
}

static bool attempt_exec_op_on_pack(struct create_pack_sp_param *args, bool is_create)
{
    MYSQL_STMT *stmt;
    MYSQL_BIND param[4];
    char sp_str[BUFFSIZE_L];

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

    snprintf(sp_str, BUFFSIZE_L, "call %s_pacco(?, ?, ?, ?)", (is_create) ? "crea" : "aggiungi_specie_a");

    param[0].buffer_type = MYSQL_TYPE_LONG; // IN var_ordine INT
    param[0].buffer = &(args->order_id);
    param[0].buffer_length = sizeof(args->order_id);

    param[1].buffer_type = MYSQL_TYPE_LONG; // IN var_specie INT
    param[1].buffer = &(args->species_code);
    param[1].buffer_length = sizeof(args->species_code);

    param[2].buffer_type = MYSQL_TYPE_LONG; // IN var_quantita INT
    param[2].buffer = &(args->quantity);
    param[2].buffer_length = sizeof(args->quantity);

    param[3].buffer_type = MYSQL_TYPE_LONG; // OUT var_numero INT
    param[3].buffer = &(args->pack_number);
    param[3].buffer_length = sizeof(args->pack_number);

    if(!exec_sp(&stmt, param, sp_str))
```

```
        return false;

    if (is_create) {
        param[0].buffer_type = MYSQL_TYPE_LONG; // OUT var_numero INT
        param[0].buffer = &(args->pack_number);
        param[0].buffer_length = sizeof(args->pack_number);

        if (!fetch_res_sp(stmt, param))
            return false;
    }

    mysql_stmt_close(stmt);
    return true;
}

static bool attempt_report_packs(unsigned int order_id)
{
    MYSQL_STMT *stmt;
    MYSQL_BIND param[1];

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

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

    if(!exec_sp(&stmt, param, "call report_pacchi(?)"))
        return false;

    if (!dump_result_set(stmt, "Processing details", LEADING_ZERO_BITMASK_IDX_0)) {
        CLOSE_AND_RETURN(false, stmt);
    }
```

```
    mysql_stmt_close(stmt);
    return true;
}

static void report_packs(void)
{
    char buffer_for_integer[BUFFSIZE_XS];
    unsigned int order_id;

    memset(buffer_for_integer, 0, sizeof(buffer_for_integer));

    init_screen(false);

    printf("*** View details about order processing ***\n");

    printf("Insert order ID.....: ");
    get_input(BUFFSIZE_XS, buffer_for_integer, false, true);
    order_id = strtol(buffer_for_integer, NULL, 10);

    putchar('\n');

    if (!attempt_show_status(order_id))
        printf("Operation failed\n");

    putchar('\n');

    if (!attempt_search_species_belonging_to_order(order_id))
        printf("Operation failed\n");

    putchar('\n');
```

```
    if (!attempt_report_packs(order_id))
        printf("Operation failed\n");

    putchar('\n');

    printf("Press enter key to get back to menu ...\n");
    getchar();
}

static void create_pack(void)
{
    char buffer_for_integer[BUFFSIZE_XS];
    char spec_name[BUFFSIZE_M];
    struct create_pack_sp_param args;

    memset(buffer_for_integer, 0, sizeof(buffer_for_integer));
    memset(&args, 0, sizeof(args));
    memset(spec_name, 0, sizeof(spec_name));

    init_screen(false);

    printf("*** Create new pack ***\n");

    printf("Insert order ID.....: ");
    get_input(BUFFSIZE_XS, buffer_for_integer, false, true);
    args.order_id = strtol(buffer_for_integer, NULL, 10);

    if (!attempt_show_status(args.order_id))
        printf("Operation failed\n");

    if (ask_for_tips("Do you wanna see a list of species belonging to selected order", 0)) {
        if (!attempt_search_species_belonging_to_order(args.order_id))
```

```
        printf("Operation failed\n");


    putchar('\n');
  }
  printf("For each species insert code and relative quantity.\nTo exit the loop enter 0 as a species code\n\n");


  while (true) {
    printf("\nInsert species code.....: ");
    get_input(BUFFSIZE_XS, buffer_for_integer, false, true);
    args.species_code = strtol(buffer_for_integer, NULL, 10);


    if (args.species_code == 0) {
      if (args.pack_number == 0) {
        printf("A pack cannot be empty!\nOperation aborted\n");
        goto exit_with_a_failure;
      } else {
        break;
      }
    }


    printf("Insert quantity.....: ");
    get_input(BUFFSIZE_XS, buffer_for_integer, false, true);
    args.quantity = strtol(buffer_for_integer, NULL, 10);


    if (args.pack_number == 0) {
      if (!attempt_exec_op_on_pack(&args, true)) {
        printf("Operation failed\n");
        goto exit_with_a_failure;
      }
    } else {
      if (!attempt_exec_op_on_pack(&args, false)) {
```

```
            printf("Operation failed\n");
            goto exit_with_a_failure;
          }
        }
    }

    printf("New pack (no. %u) has been created for order %010u\n", args.pack_number, args.order_id);

exit_with_a_failure:
    putchar('\n');

    printf("Press enter key to get back to menu ...\n");
    getchar();
}

void run_as_order_processor(char *username)
{
    struct configuration cnf;
    char choice;

    memset(&cnf, 0, sizeof(cnf));
    memset(curr_user, 0, sizeof(curr_user));

    strncpy(curr_user, username, BUFFSIZE_L);

    if (parse_config("config/opc.user", &cnf, "=")) {
        fprintf(stderr, "Invalid configuration file selected (OPP)\n");
        exit(EXIT_FAILURE);
    }

    if (mysql_change_user(conn, cnf.username, cnf.password, cnf.database)) {
```

```
        fprintf(stderr, "Unable to switch privileges\n");
        exit(EXIT_FAILURE);
    }

    while (true) {
        init_screen(true);
        printf("Welcome %s\n\n", curr_user);
        printf("*** What do you wanna do? ***\n\n");
        printf("1) Create new pack\n");
        printf("2) View details about order processing\n");
        printf("p) Change password\n");
        printf("q) Quit\n");

        choice = multi_choice("Pick an option", "12pq", 4);

        switch (choice) {
        case '1': create_pack(); break;
        case '2': report_packs(); break;
        case 'p': change_password(curr_user); break;
        case 'q': printf("Bye bye!\n\n\n"); return;
        default:
            fprintf(stderr, "Invalid condition at %s:%d\n", __FILE__, __LINE__);
            abort();
        }
    }
}
```

FILE: parser.c

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

```
#include "defines.h"

#define LINE 1024

bool parse_config(const char *path, struct configuration *conf, const char * delimiter)
{
   char line[LINE];
   char *key;
   char *value;
   int no_lines = 0;

   FILE *file = fopen(path, "r");
   if (file == NULL)
      return 1;

   while (fgets(line, LINE, file) != NULL && no_lines < 5) {
      ++ no_lines;
      key = strtok(line, delimiter);
      value = strtok(NULL, delimiter);
      value[strcspn(value, "\n")] = '\0'; // necessario per rimuovere il '\n' finale

      if (strcmp(key, "host") == 0)
         strncpy(conf->host, value, BUFFSIZE_L);
      else if (strcmp(key, "username") == 0)
         strncpy(conf->username, value, BUFFSIZE_L);
      else if (strcmp(key, "password") == 0)
         strncpy(conf->password, value, BUFFSIZE_L);
      else if (strcmp(key, "database") == 0)
         strncpy(conf->database, value, BUFFSIZE_L);
      else if (strcmp(key, "port") == 0)
         conf->port = strtol(value, NULL, 10);
      else
```

```
            return 1;
    }

    fclose(file);

    return (no_lines != 5);
}
```

FILE: utils.c

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



void init_screen(bool title)
{
    (title) ? system("clear && cat head") : system("clear");
}

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

```
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));
   }
}

bool setup_prepared_stmt(MYSQL_STMT **stmt, char *statement, MYSQL *conn)
{
   my_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;
}

static void print_dashes(MYSQL_RES *res_set)
```

```
{
   MYSQL_FIELD *field;
   unsigned int i, j;

   mysql_field_seek(res_set, 0);
   putchar('+');
   for (i = 0; i < mysql_num_fields(res_set); i++) {
      field = mysql_fetch_field(res_set);
      for (j = 0; j < field->max_length + 2; j++)
         putchar('-');
      putchar('+');
   }
   putchar('\n');
}

static void dump_result_set_header(MYSQL_RES *res_set)
{
   MYSQL_FIELD *field;
   unsigned long col_len;
   unsigned int i;

   /* determine column display widths -- requires result set to be */
   /* generated with mysql_store_result(), not mysql_use_result() */

   mysql_field_seek (res_set, 0);

   for (i = 0; i < mysql_num_fields (res_set); ++i) {
      field = mysql_fetch_field (res_set);
      col_len = strlen(field->name);

      if (col_len < field->max_length)
         col_len = field->max_length;
```

```
    if (col_len < 4 && !IS_NOT_NULL(field->flags))
      col_len = 4; /* 4 = length of the word "NULL" */
    field->max_length = col_len; /* reset column info */
  }

  print_dashes(res_set);
  putchar('|');
  mysql_field_seek (res_set, 0);

  for (i = 0; i < mysql_num_fields(res_set); ++i) {
    field = mysql_fetch_field(res_set);
    printf(" %-*s |", (int)field->max_length, field->name);
  }
  putchar('\n');

  print_dashes(res_set);
}

bool dump_result_set(MYSQL_STMT *stmt, char *title, int leading_zeros_bitmask)
{
  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_MAX_LENGTH set in `setup_prepared_stmt`
    * updates the result set metadata which are fetched in this
```

```
    * function, to allow to compute the actual max length of
    * the columns.
    */
if (mysql_stmt_store_result(stmt)) {
    fprintf(stderr, " mysql_stmt_execute(), 1 failed\n");
    fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
    return false;
}

/* 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) {
        print_stmt_error(stmt, "Unable to retrieve result metadata");
        return false;
    }

    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) {
        print_stmt_error(stmt, "Cannot allocate output buffers");
        return false;
    }
```

```
memset(rs_bind, 0, sizeof (MYSQL_BIND) * num_fields);

/* set up and bind result set output buffers */
for (i = 0; i < num_fields; ++i) {
   // Properly size the parameter buffer
   switch(fields[i].type) {
   case MYSQL_TYPE_DATE:
   case MYSQL_TYPE_TIMESTAMP:
   case MYSQL_TYPE_DATETIME:
   case MYSQL_TYPE_TIME:
      attr_size = sizeof(MYSQL_TIME);
      break;
   case MYSQL_TYPE_FLOAT:
      attr_size = sizeof(float);
      break;
   case MYSQL_TYPE_DOUBLE:
      attr_size = sizeof(double);
      break;
   case MYSQL_TYPE_TINY:
      attr_size = sizeof(signed char);
      break;
   case MYSQL_TYPE_SHORT:
   case MYSQL_TYPE_YEAR:
      attr_size = sizeof(short int);
      break;
   case MYSQL_TYPE_LONG:
   case MYSQL_TYPE_INT24:
      attr_size = sizeof(int);
      break;
   case MYSQL_TYPE_LONGLONG:
      attr_size = sizeof(int);
      break;
```

```
    default:
        attr_size = fields[i].max_length;
        break;
    }

    // Setup the binding for the current parameter
    rs_bind[i].buffer_type = fields[i].type;
    rs_bind[i].buffer = malloc(attr_size + 1);
    rs_bind[i].buffer_length = attr_size + 1;

    if(rs_bind[i].buffer == NULL) {
        print_stmt_error(stmt, "Cannot allocate output buffers");
        return false;
    }
}

if(mysql_stmt_bind_result(stmt, rs_bind)) {
    print_stmt_error(stmt, "Unable to bind output parameters");
    return false;
}

/* fetch and display result set rows */
while (true) {
    status = mysql_stmt_fetch(stmt);

    if (status == 1 || status == MYSQL_NO_DATA)
        break;

    putchar('|');

    for (i = 0; i < num_fields; ++i) {
        if (rs_bind[i].is_null_value) {
```

```
    printf (" %-*s |", (int)fields[i].max_length, "NULL");
    continue;
}

switch (rs_bind[i].buffer_type) {
case MYSQL_TYPE_VAR_STRING:
case MYSQL_TYPE_TIMESTAMP:
case MYSQL_TYPE_NEWDECIMAL:
    printf(" %-*s |", (int)fields[i].max_length, (char*)rs_bind[i].buffer);
    break;

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

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

case MYSQL_TYPE_FLOAT:
case MYSQL_TYPE_DOUBLE:
    printf(" %.02f |", *(float *)rs_bind[i].buffer);
    break;

case MYSQL_TYPE_LONG:
case MYSQL_TYPE_SHORT:
case MYSQL_TYPE_TINY:
    /* ad ogni iterazione verifico se l'i-esimo
     * bit della maschera e' settato ad uno.
     * In tal caso la stampa prevede un padding di 0
     * iniziali altrimenti una semplice spaziatura
```

```
                * come tutte le altre colonne
                */
               if ((leading_zeros_bitmask & (1 << i)) == (1 << i))
                  printf(" %0*d |", (int)fields[i].max_length, *(int *)rs_bind[i].buffer);
               else
                  printf(" %-*d |", (int)fields[i].max_length, *(int *)rs_bind[i].buffer);
               break;

            case MYSQL_TYPE_LONGLONG:
               printf(" %-*lld |", (int)fields[i].max_length, *(long long int *)rs_bind[i].buffer);
               break;

            default:
               printf("ERROR: Unhandled type (%d)\n", rs_bind[i].buffer_type);
               abort();
            }
         }
         putchar('\n');
         print_dashes(rs_metadata);
      }

      mysql_free_result(rs_metadata); /* free metadata */

      /* free output buffers */
      for (i = 0; i < num_fields; ++i)
         free(rs_bind[i].buffer);

      free(rs_bind);
   }

   return true;
}
```

FILE: warehouseclerk.c

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

struct insert_supplier_sp_params {
    char fiscal_code[BUFFSIZE_XS];
    char name[BUFFSIZE_S];
    unsigned int species_code;
    char address[BUFFSIZE_M];
};

struct spec_info {
    unsigned int species_code;
    char species_name[132]; // |NOME COMUNE| + |NOME LATINO| + 3 + 1
    unsigned int stock;
};



static char curr_user[BUFFSIZE_L];



static bool attempt_search_suppliers(char *name)
{
    MYSQL_STMT *stmt;
    MYSQL_BIND param[1];
    char prompt[BUFFSIZE_L];
```

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

    param[0].buffer_type = MYSQL_TYPE_VAR_STRING; // IN var_nome VARCHAR(32)
    param[0].buffer = name;
    param[0].buffer_length = strlen(name);

    if(!exec_sp(&stmt, param, "call visualizza_fornitori(?)"))
        return false;

    if (strlen(name) > 0)
        snprintf(prompt, BUFFSIZE_L, "\nSearch results for \'%s\':", name);
    else
        snprintf(prompt, BUFFSIZE_L, "\nSearch results:");

    if (!dump_result_set(stmt, prompt, LEADING_ZERO_BITMASK_IDX_0)) {
        CLOSE_AND_RETURN(false, stmt);
    }

    mysql_stmt_close(stmt);
    return true;
}

static bool attempt_select_available_species(unsigned int sup_code)
{
    MYSQL_STMT *stmt;
    MYSQL_BIND param[1];

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

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

```
    param[0].buffer = &sup_code;
    param[0].buffer_length = sizeof(sup_code);

    if(!exec_sp(&stmt, param, "call visualizza_specie_disponibili(?)"))
        return false;

    if (!dump_result_set(stmt, "\nSearch results:", LEADING_ZERO_BITMASK_IDX_0)) {
        CLOSE_AND_RETURN(false, stmt);
    }

    mysql_stmt_close(stmt);
    return true;
}

static unsigned int attempt_insert_new_supplier(struct insert_supplier_sp_params *input)
{
    MYSQL_STMT *stmt;
    MYSQL_BIND param[5];
    unsigned int code;

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

    param[0].buffer_type = MYSQL_TYPE_STRING; // IN var_codice_fiscale CHAR(16)
    param[0].buffer = input->fiscal_code;
    param[0].buffer_length = strlen(input->fiscal_code);

    param[1].buffer_type = MYSQL_TYPE_VAR_STRING; // IN var_nome VARCHAR(32)
    param[1].buffer = input->name;
    param[1].buffer_length = strlen(input->name);

    param[2].buffer_type = MYSQL_TYPE_LONG; // IN var_specie INT
    param[2].buffer = &(input->species_code);
```

```
    param[2].buffer_length = sizeof(input->species_code);

    param[3].buffer_type = MYSQL_TYPE_VAR_STRING; // IN var_indirizzo VARCHAR(64)
    param[3].buffer = input->address;
    param[3].buffer_length = strlen(input->address);

    param[4].buffer_type = MYSQL_TYPE_LONG; // OUT var_codice_fornitore INT
    param[4].buffer = &code;
    param[4].buffer_length = sizeof(code);

    if(!exec_sp(&stmt, param, "call inserisci_fornitore(?, ?, ?, ?, ?)"))
        return 0;

    param[0].buffer_type = MYSQL_TYPE_LONG; // OUT var_codice INT
    param[0].buffer = &code;
    param[0].buffer_length = sizeof(code);

    if (!fetch_res_sp(stmt, param))
        return 0;

    mysql_stmt_close(stmt);
    return code;
}

static void insert_new_supplier(void)
{
    struct insert_supplier_sp_params params;
    unsigned int supplier_code;
    char spec_name[BUFFSIZE_M];
    char buffer_for_integer[BUFFSIZE_XS];

    memset(&params, 0, sizeof(params));
```

```
memset(spec_name, 0, sizeof(spec_name));
memset(buffer_for_integer, 0, sizeof(buffer_for_integer));

init_screen(false);

printf("*** Insert a new supplier ***\n");

printf("Insert fiscal code.....: ");
get_input(BUFFSIZE_S, params.fiscal_code, false, true);

printf("Insert first name.....: ");
get_input(BUFFSIZE_S, params.name, false, true);

if (ask_for_tips("Do you wanna search species by name to find the right code", 0)) {
    printf("\nInsert the name to filter on (default all).....: ");
    get_input(BUFFSIZE_M, spec_name, false, false);
    if (!attempt_search_species(false, spec_name))
        printf("Operation failed\n");

    putchar('\n');
}
printf("Insert species code.....: ");
get_input(BUFFSIZE_XS, buffer_for_integer, false, true);
params.species_code = strtol(buffer_for_integer, NULL, 10);

printf("Insert address.....: ");
get_input(BUFFSIZE_M, params.address, false, true);

putchar('\n');

supplier_code = attempt_insert_new_supplier(&params);
if (supplier_code > 0)
```

```
        printf("New supplier inserted (SCODE: %010u)\n", supplier_code);
    else
        printf("Operation failed\n");

    printf("Press enter key to get back to menu ...\n");
    getchar();
}

static bool attempt_add_supply_availability(unsigned int supplier_code, unsigned int species_code)
{
    MYSQL_STMT *stmt;
    MYSQL_BIND param[2];

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

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

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

    if(!exec_sp(&stmt, param, "call aggiungi_disponibilita_fornitura(?, ?)"))
        return false;

    if (!dump_result_set(stmt, "\nUpdated list:", 0)) {
        CLOSE_AND_RETURN(false, stmt);
    }

    mysql_stmt_close(stmt);
    return true;
```

```
}

static void add_supply_availability(void)
{
   char buffer_for_integer[BUFFSIZE_XS];
   char spec_name[BUFFSIZE_M];
   char sup_name[BUFFSIZE_S];
   unsigned int sup_code;
   unsigned int spec_code;

   memset(buffer_for_integer, 0, sizeof(buffer_for_integer));
   memset(spec_name, 0, sizeof(spec_name));
   memset(sup_name, 0, sizeof(sup_name));

   init_screen(false);

   printf("*** Add supply availability ***\n");

   if (ask_for_tips("Do you wanna see a list of available suppliers", 12)) {
      printf("\nInsert the name to filter on (default all).....: ");
      get_input(BUFFSIZE_S, sup_name, false, false);

      if (!attempt_search_suppliers(sup_name))
         printf("Operation failed\n");

      putchar('\n');
   }

   printf("Insert supplier code.....: ");
   get_input(BUFFSIZE_XS, buffer_for_integer, false, true);
   sup_code = strtol(buffer_for_integer, NULL, 10);
```

```
    if (ask_for_tips("Do you wanna search species by name to find the right code", 0)) {
        printf("\nInsert the name to filter on (default all).....: ");
        get_input(BUFFSIZE_M, spec_name, false, false);

        if (!attempt_search_species(false, spec_name))
            printf("Operation failed\n");

        putchar('\n');
    }

    printf("Insert species code.....: ");
    get_input(BUFFSIZE_XS, buffer_for_integer, false, true);
    spec_code = strtol(buffer_for_integer, NULL, 10);

    putchar('\n');

    if (attempt_add_supply_availability(sup_code, spec_code))
        printf("Availability succesfully added\n");
    else
        printf("Operation failed\n");

    printf("Press enter key to get back to menu ...\n");
    getchar();
}


static bool attempt_add_supply_request(unsigned int supplier_code, unsigned int species_code, unsigned int quantity)
{
    MYSQL_STMT *stmt;
    MYSQL_BIND param[3];
    int flags = LEADING_ZERO_BITMASK_IDX_0 | LEADING_ZERO_BITMASK_IDX_1;
```

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

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

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

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

    if (!exec_sp(&stmt, param, "call inserisci_richiesta_fornitura(?, ?, ?)"))
        return false;

    if (!dump_result_set(stmt, "\nPendant request for selected species:", flags)) {
        CLOSE_AND_RETURN(false, stmt);
    }

    mysql_stmt_close(stmt);
    return true;
}

static void add_supply_request(void)
{
    char buffer_for_integer[BUFFSIZE_XS];
    char sup_name[BUFFSIZE_S];
    unsigned int sup_code;
    unsigned int spec_code;
    unsigned int quantity;
```

```
memset(buffer_for_integer, 0, sizeof(buffer_for_integer));
memset(sup_name, 0, sizeof(sup_name));

init_screen(false);

printf("*** Add supply request ***\n");

if (ask_for_tips("Do you wanna see a list of available suppliers", 10)) {
    printf("\nInsert the name to filter on (default all).....: ");
    get_input(BUFFSIZE_S, sup_name, false, false);

    if (!attempt_search_suppliers(sup_name))
        printf("Operation failed\n");

    putchar('\n');
}

printf("Insert supplier code.....: ");
get_input(BUFFSIZE_XS, buffer_for_integer, false, true);
sup_code = strtol(buffer_for_integer, NULL, 10);

if (ask_for_tips("Do you wanna see avaliable species for selected supplier", 0)) {
    if (!attempt_select_available_species(sup_code))
        printf("Operation failed\n");

    putchar('\n');
}
printf("Insert species code.....: ");
get_input(BUFFSIZE_XS, buffer_for_integer, false, true);
spec_code = strtol(buffer_for_integer, NULL, 10);
```

```
    printf("Insert relative quantity.....: ");
    get_input(BUFFSIZE_XS, buffer_for_integer, false, true);
    quantity = strtol(buffer_for_integer, NULL, 10);

    putchar('\n');

    if (!attempt_add_supply_request(sup_code, spec_code, quantity))
        printf("Operation failed\n");

    putchar('\n');

    printf("Press enter key to get back to menu ...\n");
    getchar();
}

static bool attempt_report_stock(unsigned int range)
{
    MYSQL_STMT *stmt;
    MYSQL_BIND param[1];
    int counter = 0;
    char prompt[BUFFSIZE_XL];
    struct spec_info info;
    int status;

    memset(param, 0, sizeof(param));
    memset(&info, 0, sizeof(info));
    memset(prompt, 0, sizeof(prompt));

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

```
  if(!exec_sp(&stmt, param, "call report_giacenza(?)"))
    return false;

  do {
    if(conn->server_status & SERVER_PS_OUT_PARAMS)
      goto next;

    if (counter == 0)
      snprintf(prompt, BUFFSIZE_XL, "\n*** Species details ***");
    else if (counter % 2 == 0 && counter != 0)
      snprintf(prompt, BUFFSIZE_XL, "\n\n\n*** Species details ***");
    else
      snprintf(prompt, BUFFSIZE_XL, "\nList of the 5 most frequently chosen suppliers:");

    if (!dump_result_set(stmt, prompt, LEADING_ZERO_BITMASK_IDX_0)) {
      CLOSE_AND_RETURN(false, stmt);
    }

    ++counter;
next:
    status = mysql_stmt_next_result(stmt);
    if (status > 0) {
      print_stmt_error(stmt, "Unexpected condition");
      CLOSE_AND_RETURN(false, stmt);
    }

  } while (status == 0);

  mysql_stmt_close(stmt);
  return true;
}
```

```
static void report_stock(void)
{
    char buffer_for_integer[BUFFSIZE_XS];
    unsigned int range;

    memset(buffer_for_integer, 0, sizeof(buffer_for_integer));

    init_screen(false);

    printf("*** View details of the species to be supplied ***\n");

    printf("How many species do you wanna see into the report: ");
    get_input(BUFFSIZE_XS, buffer_for_integer, false, true);
    range = strtol(buffer_for_integer, NULL, 10);

    putchar('\n');

    if (!attempt_report_stock(range))
        printf("Operation failed\n");

    printf("\n\nPress enter key to get back to menu ...\n");
    getchar();
}

static int attempt_add_address(unsigned int supplier_code, char *address)
{
    MYSQL_STMT *stmt;
    MYSQL_BIND param[2];

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

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

```
    param[0].buffer = &supplier_code;
    param[0].buffer_length = sizeof(supplier_code);

    param[1].buffer_type = MYSQL_TYPE_VAR_STRING; // IN var_indirizzo VARCHAR(64)
    param[1].buffer = address;
    param[1].buffer_length = strlen(address);

    if(!exec_sp(&stmt, param, "call aggiungi_indirizzo_fornitore(?, ?)"))
        return false;

    if (!dump_result_set(stmt, "\nUpdated addresses list:", 0)) {
        CLOSE_AND_RETURN(false, stmt);
    }

    mysql_stmt_close(stmt);
    return true;
}

static void add_address(void)
{
    unsigned int sup_code;
    char address[BUFFSIZE_M];
    char buffer_for_integer[BUFFSIZE_XS];
    char sup_name[BUFFSIZE_M];

    memset(address, 0, sizeof(address));
    memset(buffer_for_integer, 0, sizeof(buffer_for_integer));
    memset(sup_name, 0, sizeof(sup_name));

    init_screen(false);

    printf("*** Add a address for a supplier ***\n");
```

```
    if (ask_for_tips("Do you wanna see a list of available suppliers", 0)) {
        printf("\nInsert the name to filter on (default all).....: ");
        get_input(BUFFSIZE_S, sup_name, false, false);
        if (!attempt_search_suppliers(sup_name))
            printf("Operation failed\n");

        putchar('\n');
    }

    printf("Insert supplier code.....: ");
    get_input(BUFFSIZE_XS, buffer_for_integer, false, true);
    sup_code = strtol(buffer_for_integer, NULL, 10);



    printf("Insert address.....: ");
    get_input(BUFFSIZE_S, address, false, true);

    putchar('\n');

    if (attempt_add_address(sup_code, address))
        printf("Address \"%s\" for %010u succesfully added\n", address, sup_code);
    else
        printf("\nOperation failed\n");

    printf("Press enter key to get back to menu ...\n");
    getchar();
}

void run_as_warehouse_clerk(char *username)
{
    struct configuration cnf;
```

```
char choice;

memset(&cnf, 0, sizeof(cnf));
memset(curr_user, 0, sizeof(curr_user));

strncpy(curr_user, username, BUFFSIZE_L);

if (parse_config("config/whc.user", &cnf, "=")) {
   fprintf(stderr, "Invalid configuration file selected (ADM)\n");
   exit(EXIT_FAILURE);
}

if(mysql_change_user(conn, cnf.username, cnf.password, cnf.database)) {
   fprintf(stderr, "Unable to switch privileges\n");
   exit(EXIT_FAILURE);
}

while (true) {
   init_screen(true);
   printf("Welcome %s\n\n", curr_user);
   printf("*** What do you wanna do? ***\n\n");
   printf("1) Insert a new supplier\n");
   printf("2) Add address for a supplier\n");
   printf("3) Add supply availability\n");
   printf("4) Add supply request\n");
   printf("5) View details of the species to be supplied\n");
   printf("p) Change password\n");
   printf("q) Quit\n");

   choice = multi_choice("Pick an option", "12345pq", 8);

   switch (choice)
```

```
        {
        case '1': insert_new_supplier(); break;
        case '2': add_address(); break;
        case '3': add_supply_availability(); break;
        case '4': add_supply_request(); break;
        case '5': report_stock(); break;
        case 'p': change_password(curr_user); break;
        case 'q': printf("Bye bye!\n\n\n"); return;
        default:
            fprintf(stderr, "Invalid condition at %s:%d\n", __FILE__, __LINE__);
            abort();
        }
    }
}
```