|  |  |
| --- | --- |
|  |  |

Basi di Dati e Conoscenza

Progetto A.A. 2019/2020

SISTEMA GESTIONE PIANTE

0253467

Massimo Mazzetti

**Indice**

[1. Descrizione del Minimondo 3](#_Toc606296459)

[2. Analisi dei Requisiti 4](#_Toc1289394997)

[3. Progettazione concettuale 5](#_Toc2081466291)

[4. Progettazione logica 6](#_Toc2147004904)

[5. Progettazione fisica 8](#_Toc518560220)

[Appendice: Implementazione 9](#_Toc403811585)

# Descrizione del Minimondo

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  4243444546 | Si vuole realizzare un Sistema per la gestione della vendita all’ingrosso di piante.  L’azienda Verde S.r.l. gestisce la vendita all’ingrosso di piante da interni ed esterni. L’azienda tratta diverse specie di piante, ciascuna caratterizzata sia dal nome latino che dal nome comune, e da un codice univoco alfanumerico attraverso cui la specie viene identificata. Per ciascuna specie è inoltre noto se sia tipicamente da giardino o da appartamento e se sia una specie esotica o meno. Le piante possono essere verdi oppure fiorite. Nel caso di specie di piante fiorite, sono note tutte le colorazioni in cui una specie è disponibile.  L’azienda gestisce ordini massivi ed ha un parco clienti sia di rivendite che di privati. Per ciascun privato sono noti il codice fiscale, il nome e l’indirizzo della persona, mentre per ogni rivendita sono noti la partita iva, il nome e l’indirizzo della rivendita. In entrambi i casi, è possibile mantenere un numero arbitrario di contatti, ad esempio numeri di telefono, di cellulare, di indirizzi email. Per ciascun cliente è possibile indicare qual è il mezzo di comunicazione preferito per essere contattati. Nel caso di una rivendita, è necessario mantenere anche il nome/cognome di un referente, eventualmente associato ad altri contatti (con la possibilità, sempre, di indicarne uno preferito). Sia i clienti privati che le rivendite devono avere un indirizzo di fatturazione, che può essere differente dall’indirizzo di residenza o dall’indirizzo di spedizione.  I fornitori di Verde S.r.l. sono identificati attraverso un codice fornitore; per ciascun fornitore sono inoltre noti il nome, il codice fiscale ed un numero arbitrario di indirizzi. Il fornitore può fornire diverse specie di piante.  Verde S.r.l. ha un dipartimento di gestione di magazzino che tiene traccia delle giacenze ed effettua, periodicamente, ordini ai fornitori per mantenere una giacenza di tutte le specie di piante trattate.  Le specie di piante trattate sono gestite dai manager di Verde S.r.l.  Si vuole tener traccia di tutti gli acquisti eseguiti da ciascun cliente. Un acquisto, effettuato in una data specifica, è relativo a una certa quantità di piante appartenenti ad un certo numero di specie. Nell’ambito di un ordine è di interesse sapere a quale indirizzo questo deve essere inviato, e quale referente (se presente) e quale contatto fornire al corriere per mettersi in contatto con il destinatario in caso di problemi nella consegna. Non è possibile aprire un ordine se non vi è disponibilità in magazzino.  Il listino prezzi, in cui si vuole tener traccia dei prezzi assunti nel tempo da ciascuna specie di piante.  Una variazione di prezzo non deve avere effetto su un ordine già aperto ma non ancora finalizzato. I prezzi sono gestiti dai manager di Verde S.r.l.  Gli ordini vengono evasi in pacchi. Un ordine è associato ad un numero arbitrario di pacchi ed è di interesse di Verde S.r.l. tenere traccia di quali piante sono contenute all’interno di un pacco. Per motivi di ottimizzazione degli spazi, un pacco può contenere un insieme differente di specie di piante. Quando si prepara un pacco, è di interesse per l’operatore sapere quali piante devono essere ancora inserite nei pacchi, al fine di evadere correttamente l’ordine. |

# Analisi dei Requisiti

## Identificazione dei termini ambigui e correzioni possibili

|  |  |  |  |
| --- | --- | --- | --- |
| **Linea** | **Termine** | **Nuovo termine** | **Motivo correzione** |
| 6 | Giardino | Esterni | Sinonimo |
| 7 | Appartamento | Interni | Sinonimo |
| 7, 31, 43, 46 | Piante | Specie di Piante | Sinonimo |
| 12 | Indirizzo di persona | Indirizzo di residenza | Sinonimo |
| 13 | Indirizzo di rivendita | Indirizzo di residenza | Sinonimo |
| 30 | acquisto | ordine | Sinonimo |

### Specifica disambiguata

|  |
| --- |
| Si vuole realizzare un Sistema per la gestione della vendita all’ingrosso di piante.  L’azienda tratta diverse specie di piante, ciascuna caratterizzata sia dal nome latino che dal nome comune, e da un codice univoco alfanumerico attraverso cui la specie viene identificata.  Per ciascuna specie è inoltre noto se sia tipicamente da INTERNI o da ESTERNI, se sia una specie esotica o meno e se sono verdi oppure fiorite.  Nel caso di specie di piante fiorite, sono note tutte le colorazioni in cui una specie è disponibile.  L’azienda gestisce ordini massivi ed ha un parco clienti sia di rivendite che di privati.  Per ciascun privato sono noti il codice fiscale, il nome e l’indirizzo di residenza.  Per ogni rivendita sono noti la partita iva, il nome e l’indirizzo di residenza.  In entrambi i casi, è possibile mantenere un numero arbitrario di contatti, ad esempio numeri di telefono, di cellulare, di indirizzi email.  Per ciascun cliente è possibile indicare qual è il mezzo di comunicazione preferito per essere contattati.  Nel caso di una rivendita, è necessario mantenere anche il nome/cognome di un referente, eventualmente associato ad altri contatti (con la possibilità, sempre, di indicarne uno preferito).  Sia i clienti privati che le rivendite devono avere un indirizzo di fatturazione, che può essere differente dall’indirizzo di residenza o dall’indirizzo di spedizione.  I fornitori di Verde S.r.l. sono identificati attraverso un codice fornitore; per ciascun fornitore sono inoltre noti il nome, il codice fiscale ed un numero arbitrario di indirizzi.  Il fornitore può fornire diverse specie di piante.  Verde S.r.l. ha un dipartimento di gestione di magazzino che tiene traccia delle giacenze ed effettua, periodicamente, ordini ai fornitori per mantenere una giacenza di tutte le specie di piante trattate.  Le specie di piante trattate sono gestite dai manager di Verde S.r.l.  Si vuole tener traccia di tutti gli acquisti eseguiti da ciascun cliente.  Un ordine, effettuato in una data specifica, è relativo a una certa quantità di piante appartenenti ad un certo numero di specie.  Nell’ambito di un ordine è di interesse sapere a quale indirizzo questo deve essere inviato, e quale referente (se presente) e quale contatto fornire al corriere per mettersi in contatto con il destinatario in caso di problemi nella consegna.  Non è possibile inserire una determinata specie in un ordine se non vi è disponibilità in magazzino.  Il listino prezzi, in cui si vuole tener traccia dei prezzi assunti nel tempo da ciascuna specie di piante.  Una variazione di prezzo non deve avere effetto su un ordine già aperto ma non ancora finalizzato. I prezzi sono gestiti dai manager di Verde S.r.l.  Gli ordini vengono evasi in pacchi.  Un ordine è associato ad un numero arbitrario di pacchi ed è di interesse di Verde S.r.l. tenere traccia di quali piante sono contenute all’interno di un pacco.  Per motivi di ottimizzazione degli spazi, un pacco può contenere un insieme differente di specie di piante.  Quando si prepara un pacco, è di interesse per l’operatore sapere quali piante devono essere ancora inserite nei pacchi, al fine di evadere correttamente l’ordine. |

## Glossario dei Termini

|  |  |  |  |
| --- | --- | --- | --- |
| **Termine** | **Descrizione** | **Sinonimi** | **Collegamenti** |
| Specie di piante | Per specie di piante indichiamo tutti i diversi tipi di piante trattate dall’azienda. | Piante | Listino prezzi,  Magazzino, Ordini, Fornitori, Pacchi |
| Clienti | Possono essere privati o rivendite, sono coloro che acquistano le specie di piante. |  | Ordine |
| Fornitori | Aziende da cui si rifornisce il magazzino dell’azienda. |  | Specie di Piante |
| Magazzino | Posto dove si ha la giacenza delle piante. |  | Specie di piante |
| Ordini | Specie di Piante acquistate da cliente. | Acquisti | Clienti, Specie di Piante, Pacchi |
| Listino Prezzi | Costo di vendita delle piante. |  | Specie di piante |
| Pacchi | Mezzo attraverso cui vengono evasi gli ordini. |  | Ordini, Specie di Piante |

## Raggruppamento dei requisiti in insiemi omogenei

|  |
| --- |
| **Frasi relative a specie di Piante** |
| L’azienda tratta diverse specie di piante, ciascuna caratterizzata sia dal nome latino che dal nome comune, e da un codice univoco alfanumerico attraverso cui la specie viene identificata.  Per ciascuna specie è inoltre noto se sia tipicamente da INTERNI o da ESTERNI, se sia una specie esotica o meno e se sono verdi oppure fiorite.  Nel caso di specie di piante fiorite, sono note tutte le colorazioni in cui una specie è disponibile. |

|  |
| --- |
| **Frasi relative a Clienti** |
| L’azienda gestisce ordini massivi ed ha un parco clienti sia di rivendite che di privati.  Per ciascun privato sono noti il codice fiscale, il nome e l’indirizzo di residenza.  Per ogni rivendita sono noti la partita iva, il nome e l’indirizzo di residenza.  In entrambi i casi, è possibile mantenere un numero arbitrario di contatti, ad esempio numeri di telefono, di cellulare, di indirizzi email.  Per ciascun cliente è possibile indicare qual è il mezzo di comunicazione preferito per essere contattati.  Nel caso di una rivendita, è necessario mantenere anche il nome/cognome di un referente, eventualmente associato ad altri contatti (con la possibilità, sempre, di indicarne uno preferito).  Sia i clienti privati che le rivendite devono avere un indirizzo di fatturazione, che può essere differente dall’indirizzo di residenza o dall’indirizzo di spedizione. |

|  |
| --- |
| **Frasi relative a Fornitori** |
| I fornitori di Verde S.r.l. sono identificati attraverso un codice fornitore; per ciascun fornitore sono inoltre noti il nome, il codice fiscale ed un numero arbitrario di indirizzi.  Il fornitore può fornire diverse specie di piante. |

|  |
| --- |
| **Frasi relative a Magazzino** |
| Verde S.r.l. ha un dipartimento di gestione di magazzino che tiene traccia delle giacenze ed effettua, periodicamente, ordini ai fornitori per mantenere una giacenza di tutte le specie di piante trattate.  Le specie di piante trattate sono gestite dai manager di Verde S.r.l. |

|  |
| --- |
| **Frasi relative a Ordine** |
| Si vuole tener traccia di tutti gli ordini eseguiti da ciascun cliente.  Un ordine, effettuato in una data specifica, è relativo a una certa quantità di piante appartenenti ad un certo numero di specie.  Nell’ambito di un ordine è di interesse sapere a quale indirizzo questo deve essere inviato, e quale referente (se presente) e quale contatto fornire al corriere per mettersi in contatto con il destinatario in caso di problemi nella consegna.  Non è possibile inserire una determinata specie in un ordine se non vi è disponibilità in magazzino. |

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

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

# Progettazione concettuale

## Costruzione dello schema E-R

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

**----------------------------------PIANTE----------------------------------**

L’entità SPECIE DI PIANTA tiene traccia di tutte le specie gestite.

COD è un codice di 6 cifre alfanumerico che rappresenta la chiave primaria per differenziare e ricercare ogni specie.

Ci sono due generalizzazioni totali perché le Specie possono essere verdi o fiorite(in questo caso potrà presentare diversi tipi di colorazione), e allo stesso tempo, da interno o da esterno. C’è anche una generalizzazione parziale in quanto le piante possono essere esotiche.

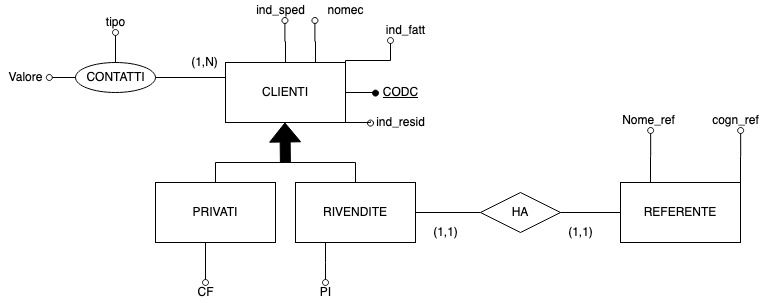
### 

**----------------------------------CLIENTI----------------------------------**

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

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

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



**----------------------------------FORNITORI----------------------------------**

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

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

Immagine che contiene testo

Descrizione generata automaticamente

**----------------------------------ORDINI----------------------------------**

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

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

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

Immagine che contiene testo, mappa

Descrizione generata automaticamente

INDIRIZZOF

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

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

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

Immagine che contiene testo, mappa

Descrizione generata automaticamente

**----------------------------------PACCHI----------------------------------**

Per evadere l’ordine bisognerà creare un numero arbitrario di pacchi e inserirci le Specie di piante.

Ogni pacco sarà identificato da un proprio codice univoco e conterrà solo piante di un determinato ordine.

Immagine che contiene testo, mappa

Descrizione generata automaticamente

### Integrazione finale

Immagine che contiene testo, mappa

Descrizione generata automaticamente

## Regole aziendali

1. Non è possibile aprire un ordine se non vi è disponibilità in magazzino.
2. Una variazione di prezzo non deve avere effetto su un ordine già aperto ma non ancora finalizzato.
3. La giacenza nel magazzino si ottiene sottraendo la quantità di piante vendute in un ordine dal totale.
4. Un pacco può contenere solo piante di un determinato ordine.

## Dizionario dei dati

|  |  |  |  |
| --- | --- | --- | --- |
| **Entità** | **Descrizione** | **Attributi** | **Identificatori** |
| Specie di Piante | Tabella che tiene memorizzate tutte le specie gestite dall’azienda | Nom\_comune,  Nom\_latino,  COD | COD |
| Listino | Tiene traccia dei prezzi assunti nel tempo da ciascuna specie di piante | Prezzo, Dataini, Datafin, CODP | CODP, Dataini |
| Magazzino | Indica il numero di pezzi disponibili della specifica pianta. | Pianta, giacenz | Pianta |
| Fornitori | Memorizza tutti dati relativi ai fornitori. | CODFOR, codfis, nomef, indirizzof, piantaforn | CODFOR |
| Ordini | Tiene traccia di tutti gli ordini eseguiti da ciascun cliente. | ID\_ordine, data, cliente | ID\_ordine |
| Clienti | Memorizza tutti i dati relativi ai clienti | CODC, nomec, ind\_fatt, ind\_resid, contatti(tipo, valore) , ind\_sped | CODC |
| Pacchi | Tiene traccia dei pacchi dove vengono evasi gli ordini. | Id\_pacco, NUM\_ORD | Id\_pacco |
| Referente | Referente di una rivendita | NOME\_REF, COGN\_REF, COD\_CLI | COD\_CLI |
| Verdi | Piante verdi | COD | COD |
| Fiorite | Piante fiorite | COD, colorazioni | COD |
| Esotica | Piante esotiche | COD | COD |
| Interni | Piante da interni | COD | COD |
| Esterni | Piante da esterni | COD | COD |
| Privati | Clienti Privati | CODC, cf | CODC |
| Rivendite | Clienti rivendite | CODC, pi | CODC |
| Acquist | Relazione di piante acquistate in un ordine | Piante, quant, Ordine | Ordine, Piante |

# Progettazione logica

## Volume dei dati

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

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

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

Assumo che ogni fornitore in media fornisce 30 specie.

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

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

## Tavola delle operazioni

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

## Costo delle operazioni

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Operazione 01🡪 costo 15 \* 1/MESE= 15/MESE** | | | | |
| **Concetto** | **Costrutto** | **Accessi** | **Tipo** |
| SPECIE DI PIANTA | E | 1 | S |
| VERDI/FIORITE | E | 1 | S |
| INTERNI/ESTERNI | E | 1 | S |
| ESOTICA | E | 1 | S |
| LISITNO | E | 1 | S |
| MAGAZZINO | E | 1 | S |
| FORNIT | R | 1 | S |
| FORNITORI | E | 1 | L |
| **Operazione 04🡪 costo 3 \* 15/MESE = 45/MESE** | | | | |
| **Concetto** | **Costrutto** | **Accessi** | **Tipo** |
| LISTINO | E | 1 | S |
| SPECIE DI PIANTA | E | 1 | L |
| **Operazione 06🡪 costo 3 \* 200/MESE = 600/MESE** | | | | |
| **Concetto** | **Costrutto** | **Accessi** | **Tipo** |
| ORDINI | E | 1 | S |
| CLIENTI | E | 1 | L |
| **Operazione 07🡪 costo 6 \* 8/MESE = 48/MESE** | | | | |
| **Concetto** | **Costrutto** | **Accessi** | **Tipo** |
| CLIENTE | E | 1 | S |
| PRIVATI | E | 1 | S |
| CONTATTI | E | 1 | S |
| **Operazione 07bis🡪 costo 8 \* 12/MESE = 96/MESE** | | | | |
| **Concetto** | **Costrutto** | **Accessi** | **Tipo** |
| REFERENTE | E | 1 | S |
| RIVENDITE | E | 1 | S |
| CLIENTE | E | 1 | S |
| CONTATTI | E | 1 | S |
| **Operazione 08🡪 costo 12 \* 1.200/MESE = 14.400/MESE** | | | | |
| **Concetto** | **Costrutto** | **Accessi** | **Tipo** |
| PACCHI | E | 1 | L |
| SPECIE DI PIANTA | E | 1 | L |
| CONTIENE | R | 3 | S |
| ORDINI | E | 1 | L |
| ACQUIST | R | 3 | L |
| **Operazione 12🡪 costo 3 \* 100/MESE = 300/MESE** | | | | |
| **Concetto** | **Costrutto** | **Accessi** | **Tipo** |
| SPECIE DI PIANTE | E | 1 | L |
| MAGAZZINO | E | 1 | S |
| **Operazione 13🡪 costo 3 \* 2/MESE = 6/MESE** | | | | |
| **Concetto** | **Costrutto** | **Accessi** | **Tipo** |
| SPECIE DI PIANTE | E | 1 | L |
| FIORITE | R | 1 | S |
| **Operazione 14🡪 costo 3\* 15/MESE = 45/MESE** | | | | |
| **Concetto** | **Costrutto** | **Accessi** | **Tipo** |
| CLIENTI | E | 1 | L |
| CONTATTI | E | 1 | S |
| **Operazione 15🡪 costo 3 \* 400/MESE = 1.200/MESE** | | | | |
| **Concetto** | **Costrutto** | **Accessi** | **Tipo** |
| ORDINI | E | 1 | L |
| PACCHI | E | 1 | S |
| **Operazione 16🡪 costo 19 \* 1.200 = 22.800/MESE** | | | | |
| **Concetto** | **Costrutto** | **Accessi** | **Tipo** |
| SPECIE DI PIANTE | E | 6 | L |
| ORDINI | E | 1 | L |
| ACQUIST | R | 6 | S |
| **Operazione 17🡪 costo 3 \* 10/MESE = 30/MESE** | | | | |
| **Concetto** | **Costrutto** | **Accessi** | **Tipo** |
| CLIENTI | E | 1 | L |
| CONTATTI | E | 1 | S |
| **Operazione 21🡪 costo 4 \* 1/MESE = 4/MESE** | | | | |
| **Concetto** | **Costrutto** | **Accessi** | **Tipo** |
| FORNIT | R | 1 | S |
| FORNITORI | E | 1 | L |
| SPECIE DI PIANTE | E | 1 | L |

## Ristrutturazione dello schema E-R

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

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

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

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

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

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

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

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

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

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

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

## Trasformazione di attributi e identificatori

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

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

LISTINO avrà come chiave la tupla Pianta\_listino e Data\_inizio.

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

## Traduzione di entità e associazioni

Specie di Piante (COD\_Pianta, Nom\_latino, Nom\_comune, Tipo, esotica)

Colore (Colore, IdFiorite)

Listino (Prezzo, Data\_inizio, Data\_fine, Pianta\_listino)

Magazzino (Giacenza, Pianta)

Fornit (Pianta\_Fornita, Fornitore)

Fornitori (idFornitori, cod\_fis, Nome\_fornitore)

Indirizzi (indirizzo, forn)

Acquist (Specie, ordine, quantita)

Ordini (idOrdine, dataAcqu, cliente)

Pacchi (idPacco, ordine)

Contiene (pacco, pianta\_pacco)

Clienti (COD\_C, Nome, ind\_fatt, ind\_resid, ind\_sped )

Referente (Nome, Cognome, rivendita)

Contatti (preferito ,tipo, valore, cliente)

Immagine che contiene testo

Descrizione generata automaticamente

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

## Normalizzazione del modello relazionale

Le tabelle sono tutte 3NF.

# Progettazione fisica

## Utenti e privilegi

**----------------------------------UTENTI----------------------------------**

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

**----------------------------------PRIVILEGI----------------------------------**

1. **Manager:**

* Aggiungere nuove specie di piante e aggiungere le varie colorazioni;
* Modificare i prezzi delle piante e visualizzarne lo storico;
* Visualizzare tutti i Fornitori di una specie;
* Creare nuovi utenti per l’utilizzo del DB;
* Visualizzare le Specie di Piante ed eventualmente le loro colorazioni tramite il loro codice;
* Aggiornare le giacenze delle Specie;
* Assegna un fornitore alla Specie di Pianta;
* Visualizzare il costo totale dell’ordine.

1. **Operatore:**

* Creare un nuovo ordine;
* Aggiungere specie ad ordine;
* Registrare nuovi clienti;
* Aggiungere contatti e selezionare un preferito;
* Creare nuovi pacchi, visualizzare le piante inserite e quelle ancora da inserire;
* Visualizzare un ordine e tutti gli ordini di un cliente;
* Visualizzare Specie, le loro colorazioni, il loro storico dei prezzi e il costo totale di un ordine.

## Strutture di memorizzazione

|  |  |  |
| --- | --- | --- |
| **Tabella Cliente** | | |
| **Attributo** | **Tipo di dato** | **Attributi[[2]](#footnote-2)** |
| **COD\_C** | VARCHAR(16) | PK, NN |
| **Nome** | VARCHAR(45) | NN |
| **Ind\_fatt** | VARCHAR(45) | NN |
| **Ind\_sped** | VARCHAR(45) | NN |
| **Ind\_resid** | VARCHAR(45) | NN |

|  |  |  |
| --- | --- | --- |
| **Tabella Colore** | | |
| **Attributo** | **Tipo di dato** | **Attributi[[3]](#footnote-3)** |
| **ID\_colore** | INT | PK, NN, AI |
| **Colore** | VARCHAR(20) | NN |

|  |  |  |
| --- | --- | --- |
| **Tabella Colore\_has\_Specie\_di\_Piante** | | |
| **Attributo** | **Tipo di dato** | **Attributi[[4]](#footnote-4)** |
| **Colore\_ID\_colore** | INT | PK, NN |
| **ID\_Fiorita** | VARCHAR(6) | PK, NN |

|  |  |  |
| --- | --- | --- |
| **Tabella Contatti** | | |
| **Attributo** | **Tipo di dato** | **Attributi[[5]](#footnote-5)** |
| **Cliente\_COD\_C** | VARCHAR(16) | PK, NN |
| **Valore** | VARCHAR(45) | PK,NN |
| **tipo** | ENUM(“email”, “cel”, “tel”) | NN |
| **preferito** | TINYINT | NN |

|  |  |  |
| --- | --- | --- |
| **Tabella Fornitori** | | |
| **Attributo** | **Tipo di dato** | **Attributi[[6]](#footnote-6)** |
| **idFornitori** | INT | PK, NN |
| **cod\_fis** | VARCHAR(16) | NN |
| **Nome\_fornitore** | VARCHAR(45) | NN |

|  |  |  |
| --- | --- | --- |
| **Tabella INDIRIZZI** | | |
| **Attributo** | **Tipo di dato** | **Attributi[[7]](#footnote-7)** |
| **indirizzo** | VARCHAR(64) | PK, NN |
| **Fornitori\_idFornitori** | INT | PK,NN |

|  |  |  |
| --- | --- | --- |
| **Tabella LISTINO** | | |
| **Attributo** | **Tipo di dato** | **Attributi[[8]](#footnote-8)** |
| **Pianta\_listino** | VARCHAR(6) | PK, NN |
| **Data\_inizio** | DATETIME | PK,NN |
| **Data\_fine** | DATETIME |  |
| **Prezzo** | DOUBLE | NN, UN |
| **Tabella MAGAZZINO** | | |
| **Attributo** | **Tipo di dato** | **Attributi[[9]](#footnote-9)** |
| **Pianta** | VARCHAR(6) | PK, NN |
| **Giacenza** | INT | NN, UN |

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

|  |  |  |
| --- | --- | --- |
| **Tabella ORDINI\_has\_Specie\_di\_Piante** | | |
| **Attributo** | **Tipo di dato** | **Attributi[[11]](#footnote-11)** |
| **ORDINI\_idORDINI** | INT | PK, NN |
| **Specie** | VARCHAR(6) | PK,NN |
| **Quantita** | INT | NN |

|  |  |  |
| --- | --- | --- |
| **Tabella PACCHI** | | |
| **Attributo** | **Tipo di dato** | **Attributi[[12]](#footnote-12)** |
| **idPACCHI** | INT | PK, NN |
| **Ordine** | INT | NN |

|  |  |  |
| --- | --- | --- |
| **Tabella PACCHI\_has\_Specie\_di\_Piante** | | |
| **Attributo** | **Tipo di dato** | **Attributi[[13]](#footnote-13)** |
| **PACCHI\_idPACCHI** | INT | PK, NN |
| **Pianta\_Pacco** | VARCHAR(6) | PK,NN |

|  |  |  |
| --- | --- | --- |
| **Tabella Referente** | | |
| **Attributo** | **Tipo di dato** | **Attributi[[14]](#footnote-14)** |
| **Rivendita** | VARCHAR(11) | PK, NN |
| **Cognome\_referente** | VARCHAR(45) | PK,NN |
| **Nome\_referente** | VARCHAR(45) | NN |

|  |  |  |
| --- | --- | --- |
| **Tabella Specie\_di\_Piante** | | |
| **Attributo** | **Tipo di dato** | **Attributi[[15]](#footnote-15)** |
| **COD\_Pianta** | VARCHAR(6) | PK, NN |
| **Nom\_latino** | VARCHAR(45) | NN |
| **Nome\_comune** | VARCHAR(45) | NN |
| **Tipo** | ENUM(“interno”,”esterno”) | NN |
| **esotica** | TINYINT | NN |

|  |  |  |
| --- | --- | --- |
| **Tabella Specie\_di\_Piante\_has\_Fornitori** | | |
| **Attributo** | **Tipo di dato** | **Attributi[[16]](#footnote-16)** |
| **Pianta\_Fornita** | VARCHAR(6) | PK, NN |
| **Fornitore** | INT | PK, NN |

|  |  |  |
| --- | --- | --- |
| **Tabella utenti** | | |
| **Attributo** | **Tipo di dato** | **Attributi[[17]](#footnote-17)** |
| **username** | VARCHAR(45) | PK, NN |
| **password** | VARCHAR(32) | NN |
| **ruolo** | ENUM(“Manager”, “operatore”) | NN |

## Indici

|  |  |
| --- | --- |
| **Tabella Cliente** | |
| **Indice PRIMARY** | **Tipo[[18]](#footnote-18):** |
| COD\_C | PR |
| **Tabella Colore** | |
| **Indice PRIMARY** | **Tipo[[19]](#footnote-19):** |
| ID\_colore | PR |
| **Tabella Colore\_has\_Specie\_di\_Piante** | |
| **Indice PRIMARY** | **Tipo[[20]](#footnote-20):** |
| Colore\_ID\_colore  ID\_Fiorita | PR |
| **Indice fk\_Colore\_has\_Specie\_di\_Piante\_Specie\_di\_Piante1\_idx** | **Tipo:** |
| ID\_Fiorita | IDX |
| **Indice fk\_Colore\_has\_Specie\_di\_Piante\_Colore1\_idx** | **Tipo:** |
| Colore\_ID\_colore | IDX |
| **Tabella Contatti** | |
| **Indice PRIMARY** | **Tipo[[21]](#footnote-21):** |
| Valore  Cliente\_COD\_C | PR |
| **Indice** fk\_Contatti\_Cliente1\_idx | **Tipo** : |
| Client\_COD\_C | IDX |
| **Tabella Fornitori** | |
| **Indice PRIMARY** | **Tipo[[22]](#footnote-22):** |
| idFornitori | PR |
| **Tabella INDIRIZZI** | |
| **Indice PRIMARY** | **Tipo[[23]](#footnote-23):** |
| indirizzo  Fornitori\_idFornitori | PR |
| **Indice fk\_INDIRIZZI\_Fornitori1\_idx** | **Tipo[[24]](#footnote-24):** |
| Fornitori\_idfornitori | IDX |
|  | |
| **Tabella LISTINO** | |
| **Indice PRIMARY** | **Tipo[[25]](#footnote-25):** |
| Data\_inizio  Pianta\_listino | PR |
| **Indice fk\_LISTINO\_Specie\_di\_Piante\_idx** | **Tipo[[26]](#footnote-26):** |
| Pianta\_listino | IDX |
|  | |
| **Tabella MAGAZZINO** | |
| **Indice PRIMARY** | **Tipo[[27]](#footnote-27):** |
| Pianta | PR |
|  | |
| **Tabella ORDINI** | |
| **Indice PRIMARY** | **Tipo[[28]](#footnote-28):** |
| idORDINI | PR |
| **Indice fk\_ORDINI\_Cliente1\_idx** | **Tipo[[29]](#footnote-29):** |
| Cliente | IDX |
|  | |
| **Tabella ORDINI\_has\_Specie\_di\_Piante** | |
| **Indice PRIMARY** | **Tipo[[30]](#footnote-30):** |
| ORDINI\_idORDINI  Specie | PR |
| **Indice fk\_ORDINI\_has\_Specie\_di\_Piante\_Specie\_di\_Piante1\_idx** | **Tipo[[31]](#footnote-31):** |
| Specie | IDX |
| **Indice fk\_ORDINI\_has\_Specie\_di\_Piante\_ORDINI1\_idx** | **Tipo[[32]](#footnote-32):** |
| ORDINI\_idORDINI | IDX |
|  | |
| **Tabella PACCHI** | |
| **Indice PRIMARY** | **Tipo[[33]](#footnote-33):** |
| idPACCHI | PR |
| **Indice fk\_PACCHI\_ORDINI1\_idx** | **Tipo[[34]](#footnote-34):** |
| Ordine | IDX |
|  | |
| **Tabella PACCHI\_has\_Specie\_di\_Piante** | |
| **Indice PRIMARY** | **Tipo[[35]](#footnote-35):** |
| PACCHI\_idPACCHI  Pianta\_Pacco | PR |
| **Indice fk\_PACCHI\_has\_Specie\_di\_Piante\_Specie\_di\_Piante1\_idx** | **Tipo[[36]](#footnote-36):** |
| Pianta\_Pacco | IDX |
| **Indice fk\_PACCHI\_has\_Specie\_di\_Piante\_PACCHI1\_idx** | **Tipo[[37]](#footnote-37):** |
| PACCHI\_idPACCHI | IDX |
|  | |
| **Tabella Referente** | |
| **Indice PRIMARY** | **Tipo[[38]](#footnote-38):** |
| Rivendita  Cognome\_referente | PR |
| **Indice fk\_Referente\_Cliente1\_idx** | **Tipo[[39]](#footnote-39):** |
| Rivendita | IDX |
|  | |
| **Tabella Specie\_di\_Piante** | |
| **Indice PRIMARY** | **Tipo[[40]](#footnote-40):** |
| COD\_Pianta | PR |
|  | |
| **Tabella Specie\_di\_Piante\_has\_Fornitori** | |
| **Indice PRIMARY** | **Tipo:** |
| Pianta\_Fortnita  Fornitore | PR |
| **Indice fk\_Specie\_di\_Piante\_has\_Fornitori\_Fornitori1\_idx** | **Tipo:** |
| Fornitore | IDX |
| **Indice fk\_Specie\_di\_Piante\_has\_Fornitori\_Specie\_di\_Piante1\_idx** | **Tipo:** |
| Pianta\_Fornita | IDX |
|  |  |
| **Tabella utenti** | |
| **Indice PRIMARY** | **Tipo:** |
| username | PR |

## Trigger

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

CREATE DEFINER = CURRENT\_USER TRIGGER `mydb`.`ORDINI\_has\_Specie\_di\_Piante\_BEFORE\_INSERT` BEFORE INSERT ON `ORDINI\_has\_Specie\_di\_Piante` FOR EACH ROW

BEGIN

if((NEW.Quantita)>(select Giacenza

from mydb.MAGAZZINO, Specie\_di\_Piante

where mydb.MAGAZZINO.Pianta = Specie\_di\_Piante.COD\_Pianta and NEW.Specie = Specie\_di\_Piante.COD\_Pianta))

then

signal sqlstate '45000' set message\_text=' Non ci sono abbastanza Piante in magazzino';

else update MAGAZZINO

set Giacenza = Giacenza - NEW.Quantita

where MAGAZZINO.Pianta = NEW.Specie;

end if;

END

## Eventi

Non ci sono eventi specifici.

## Stored Procedures e transazioni

1. **Aggiornare Giacenze:**

CREATE PROCEDURE `Aggiornare\_Giacenze`(in Specie\_pianta varchar(6),in giac int)

BEGIN

declare exit handler for sqlexception

begin

rollback;

resignal;

end;

set transaction isolation level repeatable read ;

start transaction;

if(Specie\_pianta in (select COD\_Pianta

from Specie\_di\_Piante))

then

update MAGAZZINO

SET Giacenza = Giacenza + giac

where Pianta = Specie\_pianta ;

else

signal sqlstate '45002' set message\_text=' Pianta non trovata';

end if;

commit;

END

1. **Aggiungere Specie di Piante**

CREATE PROCEDURE `aggiungere\_specie\_pianta`(in codice varchar(6), in n\_com varchar(45), in n\_lat varchar(45), in posizione enum('interno', 'esterno'), in eso TINYINT, in quantita int, in Costo double, in cod\_for int)

BEGIN

set transaction isolation level repeatable read;

start transaction;

insert into Specie\_di\_Piante(COD\_Pianta, Nom\_latino, Nom\_comune, Tipo, esotica)values (codice, n\_lat, n\_com, posizione, eso) ;

insert into LISTINO(Prezzo, Data\_inizio, Pianta\_listino) values (Costo, NOW() ,codice);

insert into MAGAZZINO(Giacenza, Pianta) values (quantita, codice);

insert into Specie\_di\_Piante\_has\_Fornitori(Pianta\_Fornita, Fornitore) values (codice, cod\_for);

commit;

END

1. **Aggiungere Cliente Privato**

CREATE PROCEDURE `Aggiungi\_cliente\_privato`(in cod varchar(16), in nom varchar(45), in fat varchar(45),in sped varchar(45), in res varchar(45),in tipo\_contatto ENUM('email', 'cel', 'tel'), in valore\_contatto varchar(45))

BEGIN

insert into Cliente(COD\_C, Nome, ind\_fatt, ind\_sped, ind\_resid) values(cod, nom, fat, sped, res);

insert into Contatti(tipo, Valore, preferito, Cliente\_COD\_C) values(tipo\_contatto, valore\_contatto, 1, cod);

END

1. **Aggiungere cliente Rivendita**

CREATE PROCEDURE `Aggiungi\_cliente\_rivendita`(in cod varchar(11), in nom varchar(45), in fat varchar(45),in sped varchar(45), in res varchar(45),in tipo\_contatto ENUM('email', 'cel', 'tel'), in valore\_contatto varchar(45), in n\_ref varchar(45), in c\_ref varchar(45))

BEGIN

insert into Cliente(COD\_C, Nome, ind\_fatt, ind\_sped, ind\_resid) values(cod, nom, fat, sped, res);

insert into Contatti(tipo, Valore, preferito, Cliente\_COD\_C) values(tipo\_contatto, valore\_contatto,1, cod);

insert into Referente(Nome\_referente, Cognome\_referente, Rivendita) values (n\_ref, c\_ref, cod);

END

1. **Aggiungere colore a Pianta**

CREATE PROCEDURE `Aggiungi\_colore\_pianta`(in colorazione varchar(20), in pianta varchar(6) )

BEGIN

if((select count(\*)

from Colore

where Colore =colorazione) = 0)

then insert into Colore (Colore) values (colorazione) ;

end if;

insert into Colore\_has\_Specie\_di\_Piante(id\_Fiorita, Colore\_ID\_colore) values

(pianta, (select ID\_colore

from Colore

where Colore =colorazione));

END

1. **Aggiungere Contatto**

CREATE PROCEDURE `aggiungi\_contatto`(in cod\_Cliente varchar(16), in tipo\_contatto ENUM('email', 'cel', 'tel'), in valore\_contatto varchar(45))

BEGIN

insert into Contatti(tipo, Valore, preferito, Cliente\_COD\_C) values(tipo\_contatto, valore\_contatto, 0, cod\_Cliente);

END

1. **Aggiungere ordine**

CREATE PROCEDURE `Aggiungi\_ordine`(in id int, in cod\_cliente varchar(16))

BEGIN

insert into ORDINI(idORDINI, dataAcq, Cliente) values (id, NOW(), cod\_cliente);

END

1. **Aggiungere Pacco**

CREATE PROCEDURE `Aggiungi\_pacco`(in ordine int, in pacco int)

BEGIN

insert into PACCHI(idPACCHI, Ordine) values (pacco, ordine);

END

1. **Aggiungere Pianta a Pacco**

CREATE PROCEDURE `Aggiungi\_piante\_a\_pacco`(in idpacco int,in pianta varchar(6))

BEGIN

declare exit handler for sqlexception

begin

rollback;

resignal;

end;

set transaction isolation level repeatable read;

start transaction;

if (pianta in (select Specie

from ORDINI\_has\_Specie\_di\_Piante, ORDINI, PACCHI

where ORDINI\_has\_Specie\_di\_Piante.ORDINI\_idORDINI = ORDINI.idORDINI and ORDINI.idORDINI = PACCHI.ordine and PACCHI.idPACCHI = idpacco

and ORDINI\_has\_Specie\_di\_Piante.Specie NOT IN (select Pianta\_Pacco as Specie

from PACCHI\_has\_Specie\_di\_Piante

where idpacco = PACCHI\_idPACCHI)))

THEN

insert into PACCHI\_has\_Specie\_di\_Piante(PACCHI\_idPACCHI, Pianta\_Pacco) values (idpacco, pianta);

else

signal sqlstate '45001' set message\_text=' Pianta non valida';

END IF;

commit;

END

1. **Aggiungere Pianta a ordine**

CREATE PROCEDURE `agg\_piante\_ordine`(in ordine int, in pianta VARCHAR(6), in quant int)

BEGIN

insert into ORDINI\_has\_Specie\_di\_Piante(ORDINI\_idORDINI, Specie, Quantita) values (ordine, pianta, quant);

END

1. **Creare nuovo utente**

CREATE PROCEDURE `crea\_utente`(IN username VARCHAR(45), IN pass VARCHAR(32), IN ruolo enum('Manager','operatore'))

BEGIN

insert into utenti(username,password, ruolo) VALUES(username, MD5(pass), ruolo);

END

1. **Login**

CREATE PROCEDURE `login`(in var\_username varchar(45), in var\_pass varchar(45), out var\_role INT)

BEGIN

declare var\_user\_role ENUM('operatore', 'Manager');

select `ruolo` from `utenti`

where `username` = var\_username and `password` = md5(var\_pass) into var\_user\_role;

-- See the corresponding enum in the client

if var\_user\_role = 'operatore' then

set var\_role = 1;

elseif var\_user\_role = 'Manager' then

set var\_role = 2;

else

set var\_role = 3;

end if;

END

1. **Modificare Prezzo Pianta**

CREATE PROCEDURE `Modificare\_prezzo\_pianta`(in Specie\_pianta varchar(6), in Nuovo\_prezzo double)

BEGIN

declare exit handler for sqlexception

begin

rollback;

resignal;

end;

set transaction isolation level serializable;

start transaction;

if((select count(\*)

from Specie\_di\_Piante

where Specie\_pianta = COD\_Pianta)>0)

then

update LISTINO

set Data\_fine = NOW()

where Pianta\_listino = Specie\_pianta and Data\_fine is null;

insert into LISTINO(Prezzo, Data\_inizio, Pianta\_listino) values (Nuovo\_prezzo,NOW() ,Specie\_pianta);

else

signal sqlstate '45002'set message\_text="Pianta non trovata";

end if;

commit;

END

1. **Scegliere Contatto Preferito**

CREATE PROCEDURE `scegli\_contatto\_preferito`(in cliente varchar(16), in valore\_contatto varchar(45))

BEGIN

declare exit handler for sqlexception

begin

rollback;

resignal;

end;

if(valore\_contatto in (select Valore from Contatti where Cliente\_COD\_C = cliente))

then

if ((select count(\*)

from Contatti

where cliente = Cliente\_COD\_C and preferito = 1)>0)

then update Contatti set preferito = 0 where cliente = Cliente\_COD\_C;

end if;

update Contatti set preferito = 1 where valore\_contatto = Valore and cliente = Cliente\_COD\_C;

else

signal sqlstate '45004' set message\_text='Contatto non trovato';

end if;

END

1. **Visualizzare Fornitori di una Specie**

CREATE PROCEDURE `visualizza\_fornitori\_pianta`(in pianta varchar(6) )

BEGIN

declare exit handler for sqlexception

begin

rollback;

resignal;

end;

if((select count(\*)

from Specie\_di\_Piante

where pianta = COD\_Pianta)>0)

then

select idFornitori, cod\_fis as 'codice fiscale', Nome\_fornitore as 'Nome', Pianta\_Fornita as 'Pianta', indirizzo

from Fornitori, Specie\_di\_Piante\_has\_Fornitori, INDIRIZZI

where pianta = Specie\_di\_Piante\_has\_Fornitori.Pianta\_Fornita and Specie\_di\_Piante\_has\_Fornitori.Fornitore = Fornitori.idFornitori and Fornitori.idFornitori = INDIRIZZI.Fornitori\_idFornitori;

else

signal sqlstate '45002'set message\_text="Pianta non trovata";

end if;

END

1. **Visualizzare ordine**

CREATE PROCEDURE `Visualizza\_ordine`(IN ordine int)

BEGIN

declare exit handler for sqlexception

begin

rollback;

resignal;

end;

set transaction isolation level repeatable read;

start transaction;

if((select count(\*) from ORDINI where idORDINI = ordine)>0)

then

if((select count(\*)

from ORDINI, ORDINI\_has\_Specie\_di\_Piante, Cliente, Contatti, Referente

where ordine = ORDINI.idORDINI and Cliente.COD\_C = ORDINI.Cliente and Cliente.COD\_C = Contatti.Cliente\_COD\_C and ORDINI\_has\_Specie\_di\_Piante.ORDINI\_idORDINI = ordine and Cliente.COD\_C = Referente.Rivendita) > 0)

then

select dataAcq, ORDINI\_has\_Specie\_di\_Piante.Specie, Quantita,Cliente.COD\_C, Cliente.Nome, Cliente.ind\_sped, Contatti.tipo, Contatti.Valore, Referente.Nome\_referente, Referente.Cognome\_referente

from ORDINI, ORDINI\_has\_Specie\_di\_Piante, Cliente, Contatti, Referente

where ordine = ORDINI.idORDINI and Cliente.COD\_C = ORDINI.Cliente and Cliente.COD\_C = Contatti.Cliente\_COD\_C and ORDINI\_has\_Specie\_di\_Piante.ORDINI\_idORDINI = ordine and Cliente.COD\_C = Referente.Rivendita and Contatti.preferito =1;

else

select distinct dataAcq, ORDINI\_has\_Specie\_di\_Piante.Specie, Quantita,Cliente.COD\_C, Cliente.Nome, Cliente.ind\_sped, Contatti.tipo, Contatti.Valore

from ORDINI, ORDINI\_has\_Specie\_di\_Piante, Cliente, Contatti

where (ordine = ORDINI.idORDINI and Cliente.COD\_C = ORDINI.Cliente and Cliente.COD\_C = Contatti.Cliente\_COD\_C and ORDINI\_has\_Specie\_di\_Piante.ORDINI\_idORDINI = ordine and Contatti.preferito=1);

end if;

else

signal sqlstate '45003' set message\_text='Ordine non trovato';

end if;

commit;

END

1. **Visualizzare ordini effettuati da un cliente**

CREATE PROCEDURE `Visualizza\_ordini\_cliente`(IN COD\_CLI VARCHAR(16))

BEGIN

declare exit handler for sqlexception

begin

rollback;

resignal;

end;

set transaction isolation level repeatable read;

start transaction;

if((select count(\*) from Cliente where COD\_CLI = COD\_C)>0)

then

select distinct idORDINI, dataAcq, ORDINI\_has\_Specie\_di\_Piante.Specie, Quantita

from ORDINI, ORDINI\_has\_Specie\_di\_Piante

where COD\_CLI = Cliente and idORDINI = ORDINI\_idORDINI;

else

signal sqlstate '45005' set message\_text="Cliente non trovato";

end if;

commit;

END

1. **Visualizzare Specie di Pianta**

CREATE DEFINER=`root`@`localhost` PROCEDURE `Visualizza\_Pianta`(IN var\_Pianta VARCHAR(6))

BEGIN

declare exit handler for sqlexception

begin

rollback;

resignal;

end;

if((select count(\*) from Specie\_di\_Piante where COD\_Pianta = var\_Pianta)>0)

then

select distinct Nom\_latino, Nom\_comune, Tipo, esotica, Giacenza

from Specie\_di\_Piante, MAGAZZINO

where var\_Pianta = COD\_Pianta and Pianta=var\_Pianta;

else

signal sqlstate '45002' set message\_text ="Pianta non trovata";

end if;

END

1. **Visualizzare Colorazioni Specie**

CREATE PROCEDURE `Visualizzare\_Colorazioni`(IN var\_Pianta VARCHAR(6))

BEGIN

declare exit handler for sqlexception

begin

rollback;

resignal;

end;

set transaction isolation level repeatable read;

start transaction;

if((select count(\*) from Specie\_di\_Piante where var\_Pianta = COD\_Pianta)>0)

then

select distinct Nom\_latino, Nom\_comune, Tipo, esotica, Colore

from Specie\_di\_Piante, Colore\_has\_Specie\_di\_Piante, Colore

where var\_Pianta = COD\_Pianta and var\_Pianta = ID\_Fiorita and Colore\_ID\_colore = ID\_colore;

else

signal sqlstate '45002' set message\_text='Pianta non trovata';

end if;

commit;

END

1. **Visualizzare Piante da inserire nei pacchi**

CREATE PROCEDURE `visualizzare\_piante\_inserire\_pacco`(in ordine int)

BEGIN

declare exit handler for sqlexception

begin

rollback;

resignal;

end;

set transaction isolation level repeatable read;

start transaction;

if((select count(\*) from ORDINI where ordine = idORDINI)>0)

then

select Specie

from ORDINI\_has\_Specie\_di\_Piante

where ORDINI\_idORDINI = ordine and Specie NOT IN (select Pianta\_Pacco as Specie

from PACCHI\_has\_Specie\_di\_Piante, PACCHI

where PACCHI.ordine = ordine and idPACCHI = PACCHI\_idPACCHI);

else

signal sqlstate '45003' set message\_text='Ordine non trovato';

end if;

commit;

END

1. **Visualizzare Piante contenute in un pacco**

CREATE PROCEDURE `Visualizzare\_piante\_Pacco`(in id\_pacco int)

BEGIN

declare exit handler for sqlexception

begin

rollback;

resignal;

end;

set transaction isolation level repeatable read;

start transaction;

if((select count(\*) from PACCHI where id\_pacco= idPACCHI)>0)

then

select Pianta\_Pacco, Nom\_comune, Nom\_latino, Ordine, Quantita

from PACCHI\_has\_Specie\_di\_Piante, Specie\_di\_Piante, PACCHI, ORDINI, ORDINI\_has\_Specie\_di\_Piante

where PACCHI\_idPACCHI = id\_pacco and Pianta\_Pacco = COD\_Pianta and id\_pacco=idPACCHI and Ordine = idORDINI and idORDINI=ORDINI\_idORDINI;

else

signal sqlstate '45006' set message\_text="Pacco non trovato";

end if;

commit;

END

1. **Visualizzare Storico de Prezzi**

CREATE PROCEDURE `Visualizzare\_Storico\_Prezzo`(IN var\_Pianta VARCHAR(6))

BEGIN

declare exit handler for sqlexception

begin

rollback;

resignal;

end;

set transaction isolation level repeatable read;

start transaction;

if((select count(\*) from Specie\_di\_Piante where var\_Pianta = COD\_Pianta)>0)

then

select \*

from LISTINO

where Pianta\_listino = var\_Pianta;

else signal sqlstate '45002' set message\_text="Pianta non trovata";

end if;

commit;

END

1. **Visualizzare il costo totale di un ordine**

CREATE PROCEDURE `visualizza\_totale\_ordine`(in ordine int)

BEGIN

declare exit handler for sqlexception

begin

rollback;

resignal;

end;

set transaction isolation level read committed;

start transaction;

if((select count(\*) from ORDINI where idORDINI = ordine)>0)

then

select sum(Prezzo \* quantita) as 'Totale'

from LISTINO, Specie\_di\_Piante, ORDINI\_has\_Specie\_di\_Piante, ORDINI

where ordine = ORDINI\_idORDINI and ordine=idORDINI and dataAcq>=Data\_inizio and (dataAcq<=Data\_fine or Data\_fine is null) and Pianta\_listino = COD\_Pianta and COD\_Pianta = Specie;

else

signal sqlstate '45003' set message\_text= 'Ordine non trovato';

end if;

commit;

END

1. **Aggiungere Fornitore ad una specie**

CREATE PROCEDURE `aggiungi\_fornitore\_specie`(in cod int, in pianta varchar(6))

BEGIN

insert into Specie\_di\_Piante\_has\_Fornitori(Pianta\_Fornita, Fornitore) values(pianta, cod);

END

# Appendice: Implementazione

## Codice SQL per instanziare il database

-- MySQL Workbench Forward Engineering

SET @OLD\_UNIQUE\_CHECKS=@@UNIQUE\_CHECKS, UNIQUE\_CHECKS=0;

SET @OLD\_FOREIGN\_KEY\_CHECKS=@@FOREIGN\_KEY\_CHECKS, FOREIGN\_KEY\_CHECKS=0;

SET @OLD\_SQL\_MODE=@@SQL\_MODE, SQL\_MODE='ONLY\_FULL\_GROUP\_BY,STRICT\_TRANS\_TABLES,NO\_ZERO\_IN\_DATE,NO\_ZERO\_DATE,ERROR\_FOR\_DIVISION\_BY\_ZERO,NO\_ENGINE\_SUBSTITUTION';

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

-- Schema mydb

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

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

-- Schema mydb

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

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

USE `mydb` ;

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

-- Table `mydb`.`Specie\_di\_Piante`

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

CREATE TABLE IF NOT EXISTS `mydb`.`Specie\_di\_Piante` (

`COD\_Pianta` VARCHAR(6) NOT NULL,

`Nom\_latino` VARCHAR(45) NOT NULL,

`Nom\_comune` VARCHAR(45) NOT NULL,

`Tipo` ENUM("interno", "esterno") NOT NULL,

`esotica` TINYINT NOT NULL,

PRIMARY KEY (`COD\_Pianta`))

ENGINE = InnoDB;

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

-- Table `mydb`.`LISTINO`

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

CREATE TABLE IF NOT EXISTS `mydb`.`LISTINO` (

`Prezzo` DOUBLE UNSIGNED NOT NULL,

`Data\_inizio` DATETIME NOT NULL,

`Data\_fine` DATETIME NULL,

`Pianta\_listino` VARCHAR(6) NOT NULL,

INDEX `fk\_LISTINO\_Specie\_di\_Piante\_idx` (`Pianta\_listino` ASC) VISIBLE,

PRIMARY KEY (`Pianta\_listino`, `Data\_inizio`),

CONSTRAINT `fk\_LISTINO\_Specie\_di\_Piante`

FOREIGN KEY (`Pianta\_listino`)

REFERENCES `mydb`.`Specie\_di\_Piante` (`COD\_Pianta`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

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

-- Table `mydb`.`Colore`

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

CREATE TABLE IF NOT EXISTS `mydb`.`Colore` (

`Colore` VARCHAR(20) NOT NULL,

`ID\_colore` INT NOT NULL AUTO\_INCREMENT,

PRIMARY KEY (`ID\_colore`))

ENGINE = InnoDB;

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

-- Table `mydb`.`Colore\_has\_Specie\_di\_Piante`

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

CREATE TABLE IF NOT EXISTS `mydb`.`Colore\_has\_Specie\_di\_Piante` (

`Colore\_ID\_colore` INT NOT NULL,

`ID\_Fiorita` VARCHAR(6) NOT NULL,

PRIMARY KEY (`Colore\_ID\_colore`, `ID\_Fiorita`),

INDEX `fk\_Colore\_has\_Specie\_di\_Piante\_Specie\_di\_Piante1\_idx` (`ID\_Fiorita` ASC) VISIBLE,

INDEX `fk\_Colore\_has\_Specie\_di\_Piante\_Colore1\_idx` (`Colore\_ID\_colore` ASC) VISIBLE,

CONSTRAINT `fk\_Colore\_has\_Specie\_di\_Piante\_Colore1`

FOREIGN KEY (`Colore\_ID\_colore`)

REFERENCES `mydb`.`Colore` (`ID\_colore`)

ON DELETE NO ACTION

ON UPDATE NO ACTION,

CONSTRAINT `fk\_Colore\_has\_Specie\_di\_Piante\_Specie\_di\_Piante1`

FOREIGN KEY (`ID\_Fiorita`)

REFERENCES `mydb`.`Specie\_di\_Piante` (`COD\_Pianta`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

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

-- Table `mydb`.`Fornitori`

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

CREATE TABLE IF NOT EXISTS `mydb`.`Fornitori` (

`idFornitori` INT NOT NULL,

`cod\_fis` VARCHAR(16) NOT NULL,

`Nome\_fornitore` VARCHAR(45) NOT NULL,

PRIMARY KEY (`idFornitori`))

ENGINE = InnoDB;

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

-- Table `mydb`.`Specie\_di\_Piante\_has\_Fornitori`

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

CREATE TABLE IF NOT EXISTS `mydb`.`Specie\_di\_Piante\_has\_Fornitori` (

`Pianta\_Fornita` VARCHAR(6) NOT NULL,

`Fornitore` INT NOT NULL,

PRIMARY KEY (`Pianta\_Fornita`, `Fornitore`),

INDEX `fk\_Specie\_di\_Piante\_has\_Fornitori\_Fornitori1\_idx` (`Fornitore` ASC) VISIBLE,

INDEX `fk\_Specie\_di\_Piante\_has\_Fornitori\_Specie\_di\_Piante1\_idx` (`Pianta\_Fornita` ASC) VISIBLE,

CONSTRAINT `fk\_Specie\_di\_Piante\_has\_Fornitori\_Specie\_di\_Piante1`

FOREIGN KEY (`Pianta\_Fornita`)

REFERENCES `mydb`.`Specie\_di\_Piante` (`COD\_Pianta`)

ON DELETE NO ACTION

ON UPDATE NO ACTION,

CONSTRAINT `fk\_Specie\_di\_Piante\_has\_Fornitori\_Fornitori1`

FOREIGN KEY (`Fornitore`)

REFERENCES `mydb`.`Fornitori` (`idFornitori`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

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

-- Table `mydb`.`INDIRIZZI`

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

CREATE TABLE IF NOT EXISTS `mydb`.`INDIRIZZI` (

`indirizzo` VARCHAR(64) NOT NULL,

`Fornitori\_idFornitori` INT NOT NULL,

INDEX `fk\_INDIRIZZI\_Fornitori1\_idx` (`Fornitori\_idFornitori` ASC) VISIBLE,

PRIMARY KEY (`indirizzo`, `Fornitori\_idFornitori`),

CONSTRAINT `fk\_INDIRIZZI\_Fornitori1`

FOREIGN KEY (`Fornitori\_idFornitori`)

REFERENCES `mydb`.`Fornitori` (`idFornitori`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

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

-- Table `mydb`.`MAGAZZINO`

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

CREATE TABLE IF NOT EXISTS `mydb`.`MAGAZZINO` (

`Giacenza` INT UNSIGNED NOT NULL,

`Pianta` VARCHAR(6) NOT NULL,

PRIMARY KEY (`Pianta`),

CONSTRAINT `fk\_MAGAZZINO\_Specie\_di\_Piante1`

FOREIGN KEY (`Pianta`)

REFERENCES `mydb`.`Specie\_di\_Piante` (`COD\_Pianta`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

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

-- Table `mydb`.`Cliente`

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

CREATE TABLE IF NOT EXISTS `mydb`.`Cliente` (

`COD\_C` VARCHAR(16) NOT NULL,

`Nome` VARCHAR(45) NOT NULL,

`ind\_fatt` VARCHAR(45) NOT NULL,

`ind\_sped` VARCHAR(45) NOT NULL,

`ind\_resid` VARCHAR(45) NOT NULL,

PRIMARY KEY (`COD\_C`))

ENGINE = InnoDB;

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

-- Table `mydb`.`ORDINI`

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

CREATE TABLE IF NOT EXISTS `mydb`.`ORDINI` (

`idORDINI` INT NOT NULL,

`dataAcq` DATETIME NOT NULL,

`Cliente` VARCHAR(16) NOT NULL,

PRIMARY KEY (`idORDINI`),

INDEX `fk\_ORDINI\_Cliente1\_idx` (`Cliente` ASC) VISIBLE,

CONSTRAINT `fk\_ORDINI\_Cliente1`

FOREIGN KEY (`Cliente`)

REFERENCES `mydb`.`Cliente` (`COD\_C`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

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

-- Table `mydb`.`ORDINI\_has\_Specie\_di\_Piante`

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

CREATE TABLE IF NOT EXISTS `mydb`.`ORDINI\_has\_Specie\_di\_Piante` (

`ORDINI\_idORDINI` INT NOT NULL,

`Specie` VARCHAR(6) NOT NULL,

`Quantita` INT NOT NULL,

PRIMARY KEY (`ORDINI\_idORDINI`, `Specie`),

INDEX `fk\_ORDINI\_has\_Specie\_di\_Piante\_Specie\_di\_Piante1\_idx` (`Specie` ASC) VISIBLE,

INDEX `fk\_ORDINI\_has\_Specie\_di\_Piante\_ORDINI1\_idx` (`ORDINI\_idORDINI` ASC) VISIBLE,

CONSTRAINT `fk\_ORDINI\_has\_Specie\_di\_Piante\_ORDINI1`

FOREIGN KEY (`ORDINI\_idORDINI`)

REFERENCES `mydb`.`ORDINI` (`idORDINI`)

ON DELETE NO ACTION

ON UPDATE NO ACTION,

CONSTRAINT `fk\_ORDINI\_has\_Specie\_di\_Piante\_Specie\_di\_Piante1`

FOREIGN KEY (`Specie`)

REFERENCES `mydb`.`Specie\_di\_Piante` (`COD\_Pianta`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

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

-- Table `mydb`.`PACCHI`

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

CREATE TABLE IF NOT EXISTS `mydb`.`PACCHI` (

`idPACCHI` INT NOT NULL,

`Ordine` INT NOT NULL,

PRIMARY KEY (`idPACCHI`),

INDEX `fk\_PACCHI\_ORDINI1\_idx` (`Ordine` ASC) VISIBLE,

CONSTRAINT `fk\_PACCHI\_ORDINI1`

FOREIGN KEY (`Ordine`)

REFERENCES `mydb`.`ORDINI` (`idORDINI`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

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

-- Table `mydb`.`PACCHI\_has\_Specie\_di\_Piante`

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

CREATE TABLE IF NOT EXISTS `mydb`.`PACCHI\_has\_Specie\_di\_Piante` (

`PACCHI\_idPACCHI` INT NOT NULL,

`Pianta\_Pacco` VARCHAR(6) NOT NULL,

PRIMARY KEY (`PACCHI\_idPACCHI`, `Pianta\_Pacco`),

INDEX `fk\_PACCHI\_has\_Specie\_di\_Piante\_Specie\_di\_Piante1\_idx` (`Pianta\_Pacco` ASC) VISIBLE,

INDEX `fk\_PACCHI\_has\_Specie\_di\_Piante\_PACCHI1\_idx` (`PACCHI\_idPACCHI` ASC) VISIBLE,

CONSTRAINT `fk\_PACCHI\_has\_Specie\_di\_Piante\_PACCHI1`

FOREIGN KEY (`PACCHI\_idPACCHI`)

REFERENCES `mydb`.`PACCHI` (`idPACCHI`)

ON DELETE NO ACTION

ON UPDATE NO ACTION,

CONSTRAINT `fk\_PACCHI\_has\_Specie\_di\_Piante\_Specie\_di\_Piante1`

FOREIGN KEY (`Pianta\_Pacco`)

REFERENCES `mydb`.`Specie\_di\_Piante` (`COD\_Pianta`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

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

-- Table `mydb`.`Referente`

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

CREATE TABLE IF NOT EXISTS `mydb`.`Referente` (

`Nome\_referente` VARCHAR(45) NOT NULL,

`Cognome\_referente` VARCHAR(45) NOT NULL,

`Rivendita` VARCHAR(11) NOT NULL,

INDEX `fk\_Referente\_Cliente1\_idx` (`Rivendita` ASC) VISIBLE,

PRIMARY KEY (`Rivendita`, `Cognome\_referente`),

CONSTRAINT `fk\_Referente\_Cliente1`

FOREIGN KEY (`Rivendita`)

REFERENCES `mydb`.`Cliente` (`COD\_C`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

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

-- Table `mydb`.`Contatti`

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

CREATE TABLE IF NOT EXISTS `mydb`.`Contatti` (

`tipo` ENUM("email", "cel", "tel") NOT NULL,

`Valore` VARCHAR(45) NOT NULL,

`preferito` TINYINT NOT NULL,

`Cliente\_COD\_C` VARCHAR(16) NOT NULL,

INDEX `fk\_Contatti\_Cliente1\_idx` (`Cliente\_COD\_C` ASC) VISIBLE,

PRIMARY KEY (`Cliente\_COD\_C`, `Valore`),

CONSTRAINT `fk\_Contatti\_Cliente1`

FOREIGN KEY (`Cliente\_COD\_C`)

REFERENCES `mydb`.`Cliente` (`COD\_C`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

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

-- Table `mydb`.`utenti`

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

CREATE TABLE IF NOT EXISTS `mydb`.`utenti` (

`username` VARCHAR(45) NOT NULL,

`password` VARCHAR(32) NOT NULL,

`ruolo` ENUM("operatore", "Manager") NOT NULL,

PRIMARY KEY (`username`));

USE `mydb` ;

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

-- procedure Visualizza\_Pianta

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

DELIMITER $$

USE `mydb`$$

CREATE DEFINER=`root`@`localhost` PROCEDURE `Visualizza\_Pianta`(IN var\_Pianta VARCHAR(6))

BEGIN

declare exit handler for sqlexception

begin

rollback;

resignal;

end;

if((select count(\*)

from Specie\_di\_Piante

where COD\_Pianta = var\_Pianta)>0)

then

select distinct Nom\_latino, Nom\_comune, Tipo, esotica, Giacenza

from Specie\_di\_Piante, MAGAZZINO

where var\_Pianta = COD\_Pianta and Pianta=var\_Pianta;

else

signal sqlstate '45002' set message\_text ="Pianta non trovata";

end if;

END$$

DELIMITER ;

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

-- procedure Visualizzare\_Storico\_Prezzo

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

DELIMITER $$

USE `mydb`$$

CREATE PROCEDURE `Visualizzare\_Storico\_Prezzo`(IN var\_Pianta VARCHAR(6))

BEGIN

declare exit handler for sqlexception

begin

rollback;

resignal;

end;

set transaction isolation level repeatable read;

start transaction;

if((select count(\*)

from Specie\_di\_Piante

where var\_Pianta = COD\_Pianta)>0)

then

select \*

from LISTINO

where Pianta\_listino = var\_Pianta;

else signal sqlstate '45002' set message\_text="Pianta non trovata";

end if;

commit;

END$$

DELIMITER ;

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

-- procedure Visualizzare\_Colorazioni

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

DELIMITER $$

USE `mydb`$$

CREATE PROCEDURE `Visualizzare\_Colorazioni`(IN var\_Pianta VARCHAR(6))

BEGIN

declare exit handler for sqlexception

begin

rollback;

resignal;

end;

set transaction isolation level repeatable read;

start transaction;

if((select count(\*)

from Specie\_di\_Piante

where var\_Pianta = COD\_Pianta)>0)

then

select distinct Nom\_latino, Nom\_comune, Tipo, esotica, Colore

from Specie\_di\_Piante, Colore\_has\_Specie\_di\_Piante, Colore

where var\_Pianta = COD\_Pianta and var\_Pianta = ID\_Fiorita and Colore\_ID\_colore = ID\_colore;

else

signal sqlstate '45002' set message\_text='Pianta non trovata';

end if;

commit;

END$$

DELIMITER ;

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

-- procedure Visualizza\_ordine

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

DELIMITER $$

USE `mydb`$$

CREATE PROCEDURE `Visualizza\_ordine`(IN ordine int)

BEGIN

declare exit handler for sqlexception

begin

rollback;

resignal;

end;

set transaction isolation level repeatable read;

start transaction;

if((select count(\*)

from ORDINI

where idORDINI = ordine)>0)

then

if((select count(\*)

from ORDINI, ORDINI\_has\_Specie\_di\_Piante, Cliente, Contatti, Referente

where ordine = ORDINI.idORDINI and Cliente.COD\_C = ORDINI.Cliente and Cliente.COD\_C = Contatti.Cliente\_COD\_C and ORDINI\_has\_Specie\_di\_Piante.ORDINI\_idORDINI = ordine and Cliente.COD\_C = Referente.Rivendita) > 0)

then

select dataAcq, ORDINI\_has\_Specie\_di\_Piante.Specie, Quantita,Cliente.COD\_C, Cliente.Nome, Cliente.ind\_sped, Contatti.tipo, Contatti.Valore, Referente.Nome\_referente, Referente.Cognome\_referente

from ORDINI, ORDINI\_has\_Specie\_di\_Piante, Cliente, Contatti, Referente

where ordine = ORDINI.idORDINI and Cliente.COD\_C = ORDINI.Cliente and Cliente.COD\_C = Contatti.Cliente\_COD\_C and ORDINI\_has\_Specie\_di\_Piante.ORDINI\_idORDINI = ordine and Cliente.COD\_C = Referente.Rivendita and Contatti.preferito =1;

else

select distinct dataAcq, ORDINI\_has\_Specie\_di\_Piante.Specie, Quantita,Cliente.COD\_C, Cliente.Nome, Cliente.ind\_sped, Contatti.tipo, Contatti.Valore

from ORDINI, ORDINI\_has\_Specie\_di\_Piante, Cliente, Contatti

where (ordine = ORDINI.idORDINI and Cliente.COD\_C = ORDINI.Cliente and Cliente.COD\_C = Contatti.Cliente\_COD\_C and ORDINI\_has\_Specie\_di\_Piante.ORDINI\_idORDINI = ordine and Contatti.preferito=1);

end if;

else

signal sqlstate '45003' set message\_text='Ordine non trovato';

end if;

commit;

END$$

DELIMITER ;

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

-- procedure Visualizza\_ordini\_cliente

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

DELIMITER $$

USE `mydb`$$

CREATE PROCEDURE `Visualizza\_ordini\_cliente`(IN COD\_CLI VARCHAR(16))

BEGIN

declare exit handler for sqlexception

begin

rollback;

resignal;

end;

set transaction isolation level repeatable read;

start transaction;

if((select count(\*)

from Cliente

where COD\_CLI = COD\_C)>0)

then

select distinct idORDINI, dataAcq, ORDINI\_has\_Specie\_di\_Piante.Specie, Quantita

from ORDINI, ORDINI\_has\_Specie\_di\_Piante

where COD\_CLI = Cliente and idORDINI = ORDINI\_idORDINI;

else

signal sqlstate '45005' set message\_text="Cliente non trovato";

end if;

commit;

END$$

DELIMITER ;

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

-- procedure Visualizzare\_piante\_Pacco

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

DELIMITER $$

USE `mydb`$$

CREATE PROCEDURE `Visualizzare\_piante\_Pacco`(in id\_pacco int)

BEGIN

declare exit handler for sqlexception

begin

rollback;

resignal;

end;

set transaction isolation level repeatable read;

start transaction;

if((select count(\*)

from PACCHI

where id\_pacco= idPACCHI)>0)

then

select Pianta\_Pacco, Nom\_comune, Nom\_latino, Ordine, Quantita

from PACCHI\_has\_Specie\_di\_Piante, Specie\_di\_Piante, PACCHI, ORDINI, ORDINI\_has\_Specie\_di\_Piante

where PACCHI\_idPACCHI = id\_pacco and Pianta\_Pacco = COD\_Pianta and id\_pacco=idPACCHI and Ordine = idORDINI and idORDINI=ORDINI\_idORDINI;

else

signal sqlstate '45006' set message\_text="Pacco non trovato";

end if;

commit;

END$$

DELIMITER ;

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

-- procedure visualizzare\_piante\_inserire\_pacco

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

DELIMITER $$

USE `mydb`$$

CREATE PROCEDURE `visualizzare\_piante\_inserire\_pacco`(in ordine int)

BEGIN

declare exit handler for sqlexception

begin

rollback;

resignal;

end;

set transaction isolation level repeatable read;

start transaction;

if((select count(\*)

from ORDINI

where ordine = idORDINI)>0)

then

select Specie

from ORDINI\_has\_Specie\_di\_Piante

where ORDINI\_idORDINI = ordine and Specie NOT IN (select Pianta\_Pacco as Specie

from PACCHI\_has\_Specie\_di\_Piante, PACCHI

where PACCHI.ordine = ordine and idPACCHI = PACCHI\_idPACCHI);

else

signal sqlstate '45003' set message\_text='Ordine non trovato';

end if;

commit;

END$$

DELIMITER ;

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

-- procedure Aggiornare\_Giacenze

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

DELIMITER $$

USE `mydb`$$

CREATE PROCEDURE `Aggiornare\_Giacenze`(in Specie\_pianta varchar(6),in giac int)

BEGIN

declare exit handler for sqlexception

begin

rollback;

resignal;

end;

set transaction isolation level repeatable read ;

start transaction;

if(Specie\_pianta in (select COD\_Pianta

from Specie\_di\_Piante))

then

update MAGAZZINO

SET Giacenza = Giacenza + giac

where Pianta = Specie\_pianta ;

else

signal sqlstate '45002' set message\_text=' Pianta non trovata';

end if;

commit;

END$$

DELIMITER ;

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

-- procedure Modificare\_prezzo\_pianta

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

DELIMITER $$

USE `mydb`$$

CREATE PROCEDURE `Modificare\_prezzo\_pianta`(in Specie\_pianta varchar(6), in Nuovo\_prezzo double)

BEGIN

declare exit handler for sqlexception

begin

rollback;

resignal;

end;

set transaction isolation level serializable;

start transaction;

if((select count(\*)

from Specie\_di\_Piante

where Specie\_pianta = COD\_Pianta)>0)

then

update LISTINO

set Data\_fine = NOW()

where Pianta\_listino = Specie\_pianta and Data\_fine is null;

insert into LISTINO(Prezzo, Data\_inizio, Pianta\_listino) values (Nuovo\_prezzo,NOW() ,Specie\_pianta);

else

signal sqlstate '45002'set message\_text="Pianta non trovata";

end if;

commit;

END$$

DELIMITER ;

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

-- procedure aggiungere\_specie\_pianta

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

DELIMITER $$

USE `mydb`$$

CREATE PROCEDURE `aggiungere\_specie\_pianta`(in codice varchar(6), in n\_com varchar(45), in n\_lat varchar(45), in posizione enum('interno', 'esterno'), in eso TINYINT, in quantita int, in Costo double, in cod\_for int)

BEGIN

set transaction isolation level repeatable read;

start transaction;

insert into Specie\_di\_Piante(COD\_Pianta, Nom\_latino, Nom\_comune, Tipo, esotica) values (codice, n\_lat, n\_com, posizione, eso) ;

insert into LISTINO(Prezzo, Data\_inizio, Pianta\_listino) values (Costo, NOW() ,codice);

insert into MAGAZZINO(Giacenza, Pianta) values (quantita, codice);

insert into Specie\_di\_Piante\_has\_Fornitori(Pianta\_Fornita, Fornitore) values (codice, cod\_for);

commit;

END$$

DELIMITER ;

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

-- procedure Aggiungi\_cliente\_privato

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

DELIMITER $$

USE `mydb`$$

CREATE PROCEDURE `Aggiungi\_cliente\_privato`(in cod varchar(16), in nom varchar(45), in fat varchar(45),in sped varchar(45), in res varchar(45),in tipo\_contatto ENUM('email', 'cel', 'tel'), in valore\_contatto varchar(45))

BEGIN

insert into Cliente(COD\_C, Nome, ind\_fatt, ind\_sped, ind\_resid) values(cod, nom, fat, sped, res);

insert into Contatti(tipo, Valore, preferito, Cliente\_COD\_C) values(tipo\_contatto, valore\_contatto, 1, cod);

END$$

DELIMITER ;

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

-- procedure Aggiungi\_cliente\_rivendita

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

DELIMITER $$

USE `mydb`$$

CREATE PROCEDURE `Aggiungi\_cliente\_rivendita`(in cod varchar(11), in nom varchar(45), in fat varchar(45),in sped varchar(45), in res varchar(45),in tipo\_contatto ENUM('email', 'cel', 'tel'), in valore\_contatto varchar(45), in n\_ref varchar(45), in c\_ref varchar(45))

BEGIN

insert into Cliente(COD\_C, Nome, ind\_fatt, ind\_sped, ind\_resid) values(cod, nom, fat, sped, res);

insert into Contatti(tipo, Valore, preferito, Cliente\_COD\_C) values(tipo\_contatto, valore\_contatto,1, cod);

insert into Referente(Nome\_referente, Cognome\_referente, Rivendita) values (n\_ref, c\_ref, cod);

END$$

DELIMITER ;

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

-- procedure Aggiungi\_ordine

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

DELIMITER $$

USE `mydb`$$

CREATE PROCEDURE `Aggiungi\_ordine`(in id int, in cod\_cliente varchar(16))

BEGIN

insert into ORDINI(idORDINI, dataAcq, Cliente) values (id, NOW(), cod\_cliente);

END$$

DELIMITER ;

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

-- procedure agg\_piante\_ordine

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

DELIMITER $$

USE `mydb`$$

CREATE PROCEDURE `agg\_piante\_ordine`(in ordine int, in pianta VARCHAR(6), in quant int)

BEGIN

insert into ORDINI\_has\_Specie\_di\_Piante(ORDINI\_idORDINI, Specie, Quantita) values (ordine, pianta, quant);

END$$

DELIMITER ;

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

-- procedure aggiungi\_contatto

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

DELIMITER $$

USE `mydb`$$

CREATE PROCEDURE `aggiungi\_contatto`(in cod\_Cliente varchar(16), in tipo\_contatto ENUM('email', 'cel', 'tel'), in valore\_contatto varchar(45))

BEGIN

insert into Contatti(tipo, Valore, preferito, Cliente\_COD\_C) values(tipo\_contatto, valore\_contatto, 0, cod\_Cliente);

END$$

DELIMITER ;

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

-- procedure scegli\_contatto\_preferito

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

DELIMITER $$

USE `mydb`$$

CREATE PROCEDURE `scegli\_contatto\_preferito`(in cliente varchar(16), in valore\_contatto varchar(45))

BEGIN

declare exit handler for sqlexception

begin

rollback;

resignal;

end;

if(valore\_contatto in (select Valore from Contatti where Cliente\_COD\_C = cliente))

then

if ((select count(\*)

from Contatti

where cliente = Cliente\_COD\_C and preferito = 1)>0)

then update Contatti set preferito = 0 where cliente = Cliente\_COD\_C;

end if;

update Contatti set preferito = 1 where valore\_contatto = Valore and cliente = Cliente\_COD\_C;

else

signal sqlstate '45004' set message\_text='Contatto non trovato';

end if;

END$$

DELIMITER ;

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

-- procedure Aggiungi\_piante\_a\_pacco

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

DELIMITER $$

USE `mydb`$$

CREATE PROCEDURE `Aggiungi\_piante\_a\_pacco`(in idpacco int,in pianta varchar(6))

BEGIN

declare exit handler for sqlexception

begin

rollback;

resignal;

end;

set transaction isolation level repeatable read;

start transaction;

if (pianta in (select Specie

from ORDINI\_has\_Specie\_di\_Piante, ORDINI, PACCHI

where ORDINI\_has\_Specie\_di\_Piante.ORDINI\_idORDINI = ORDINI.idORDINI and ORDINI.idORDINI = PACCHI.ordine and PACCHI.idPACCHI = idpacco

and ORDINI\_has\_Specie\_di\_Piante.Specie NOT IN (select Pianta\_Pacco as Specie

from PACCHI\_has\_Specie\_di\_Piante

where idpacco = PACCHI\_idPACCHI)))

THEN

insert into PACCHI\_has\_Specie\_di\_Piante(PACCHI\_idPACCHI, Pianta\_Pacco) values (idpacco, pianta);

else

signal sqlstate '45001' set message\_text=' Pianta non valida';

END IF;

commit;

END$$

DELIMITER ;

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

-- procedure Aggiungi\_colore\_pianta

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

DELIMITER $$

USE `mydb`$$

CREATE PROCEDURE `Aggiungi\_colore\_pianta`(in colorazione varchar(20), in pianta varchar(6) )

BEGIN

if((select count(\*)

from Colore

where Colore =colorazione) = 0)

then insert into Colore (Colore) values (colorazione) ;

end if;

insert into Colore\_has\_Specie\_di\_Piante(id\_Fiorita, Colore\_ID\_colore) values (pianta, (select ID\_colore

from Colore

where Colore =colorazione));

END$$

DELIMITER ;

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

-- procedure visualizza\_fornitori\_pianta

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

DELIMITER $$

USE `mydb`$$

CREATE PROCEDURE `visualizza\_fornitori\_pianta`(in pianta varchar(6) )

BEGIN

declare exit handler for sqlexception

begin

rollback;

resignal;

end;

if((select count(\*)

from Specie\_di\_Piante

where pianta = COD\_Pianta)>0)

then

select idFornitori, cod\_fis as 'codice fiscale', Nome\_fornitore as 'Nome', Pianta\_Fornita as 'Pianta', indirizzo

from Fornitori, Specie\_di\_Piante\_has\_Fornitori, INDIRIZZI

where pianta = Specie\_di\_Piante\_has\_Fornitori.Pianta\_Fornita and Specie\_di\_Piante\_has\_Fornitori.Fornitore = Fornitori.idFornitori and Fornitori.idFornitori = INDIRIZZI.Fornitori\_idFornitori;

else

signal sqlstate '45002'set message\_text="Pianta non trovata";

end if;

END$$

DELIMITER ;

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

-- procedure login

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

DELIMITER $$

USE `mydb`$$

CREATE PROCEDURE `login`(in var\_username varchar(45), in var\_pass varchar(45), out var\_role INT)

BEGIN

declare var\_user\_role ENUM('operatore', 'Manager');

select `ruolo` from `utenti`

where `username` = var\_username

and `password` = md5(var\_pass)

into var\_user\_role;

-- See the corresponding enum in the client

if var\_user\_role = 'operatore' then

set var\_role = 1;

elseif var\_user\_role = 'Manager' then

set var\_role = 2;

else

set var\_role = 3;

end if;

END$$

DELIMITER ;

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

-- procedure Aggiungi\_pacco

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

DELIMITER $$

USE `mydb`$$

CREATE PROCEDURE `Aggiungi\_pacco`(in ordine int, in pacco int)

BEGIN

insert into PACCHI(idPACCHI, Ordine) values (pacco, ordine);

END$$

DELIMITER ;

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

-- procedure visualizza\_totale\_ordine

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

DELIMITER $$

USE `mydb`$$

CREATE PROCEDURE `visualizza\_totale\_ordine`(in ordine int)

BEGIN

declare exit handler for sqlexception

begin

rollback;

resignal;

end;

set transaction isolation level read committed;

start transaction;

if((select count(\*)

from ORDINI

where idORDINI = ordine)>0)

then

select sum(Prezzo \* quantita) as 'Totale'

from LISTINO, Specie\_di\_Piante, ORDINI\_has\_Specie\_di\_Piante, ORDINI

where ordine = ORDINI\_idORDINI and ordine=idORDINI and dataAcq>=Data\_inizio and (dataAcq<=Data\_fine or Data\_fine is null) and Pianta\_listino = COD\_Pianta and COD\_Pianta = Specie;

else

signal sqlstate '45003' set message\_text= 'Ordine non trovato';

end if;

commit;

END$$

DELIMITER ;

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

-- procedure crea\_utente

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

DELIMITER $$

USE `mydb`$$

CREATE PROCEDURE `crea\_utente`(IN username VARCHAR(45), IN pass VARCHAR(32), IN ruolo enum('Manager','operatore'))

BEGIN

insert into utenti(username,password, ruolo) VALUES(username, MD5(pass), ruolo);

END$$

DELIMITER ;

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

-- procedure aggiungi\_fornitore\_specie

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

DELIMITER $$

USE `mydb`$$

CREATE PROCEDURE `aggiungi\_fornitore\_specie`(in cod int, in pianta varchar(6))

BEGIN

insert into Specie\_di\_Piante\_has\_Fornitori(Pianta\_Fornita, Fornitore) values(pianta, cod);

END$$

DELIMITER ;

USE `mydb`;

DELIMITER $$

USE `mydb`$$

CREATE DEFINER = CURRENT\_USER TRIGGER `mydb`.`ORDINI\_has\_Specie\_di\_Piante\_BEFORE\_INSERT` BEFORE INSERT ON `ORDINI\_has\_Specie\_di\_Piante` FOR EACH ROW

BEGIN

if((NEW.Quantita)>(select Giacenza

from mydb.MAGAZZINO, Specie\_di\_Piante

where mydb.MAGAZZINO.Pianta = Specie\_di\_Piante.COD\_Pianta and NEW.Specie = Specie\_di\_Piante.COD\_Pianta))

then

signal sqlstate '45000' set message\_text=' Non ci sono abbastanza Piante in magazzino';

else update MAGAZZINO

set Giacenza = Giacenza - NEW.Quantita

where MAGAZZINO.Pianta = NEW.Specie;

end if;

END$$

DELIMITER ;

CREATE USER 'operatore' IDENTIFIED BY 'operatore';

GRANT EXECUTE ON procedure `mydb`.`agg\_piante\_ordine` TO 'operatore';

GRANT EXECUTE ON procedure `mydb`.`Aggiungi\_cliente\_privato` TO 'operatore';

GRANT EXECUTE ON procedure `mydb`.`Aggiungi\_cliente\_rivendita` TO 'operatore';

GRANT EXECUTE ON procedure `mydb`.`aggiungi\_contatto` TO 'operatore';

GRANT EXECUTE ON procedure `mydb`.`Aggiungi\_ordine` TO 'operatore';

GRANT EXECUTE ON procedure `mydb`.`Aggiungi\_piante\_a\_pacco` TO 'operatore';

GRANT EXECUTE ON procedure `mydb`.`scegli\_contatto\_preferito` TO 'operatore';

GRANT EXECUTE ON procedure `mydb`.`Visualizza\_ordine` TO 'operatore';

GRANT EXECUTE ON procedure `mydb`.`Visualizza\_ordini\_cliente` TO 'operatore';

GRANT EXECUTE ON procedure `mydb`.`visualizzare\_piante\_inserire\_pacco` TO 'operatore';

GRANT EXECUTE ON procedure `mydb`.`Visualizzare\_piante\_Pacco` TO 'operatore';

GRANT EXECUTE ON procedure `mydb`.`Visualizzare\_Storico\_Prezzo` TO 'operatore';

GRANT EXECUTE ON procedure `mydb`.`Aggiungi\_pacco` TO 'operatore';

GRANT EXECUTE ON procedure `mydb`.`Visualizza\_Pianta` TO 'operatore';

GRANT EXECUTE ON procedure `mydb`.`visualizza\_totale\_ordine` TO 'operatore';

GRANT EXECUTE ON procedure `mydb`.`Visualizzare\_Colorazioni` TO 'operatore';

CREATE USER 'Manager' IDENTIFIED BY 'Manager';

GRANT EXECUTE ON procedure `mydb`.`aggiungere\_specie\_pianta` TO 'Manager';

GRANT EXECUTE ON procedure `mydb`.`Aggiungi\_colore\_pianta` TO 'Manager';

GRANT EXECUTE ON procedure `mydb`.`Modificare\_prezzo\_pianta` TO 'Manager';

GRANT EXECUTE ON procedure `mydb`.`Visualizzare\_Storico\_Prezzo` TO 'Manager';

GRANT EXECUTE ON procedure `mydb`.`visualizza\_fornitori\_pianta` TO 'Manager';

GRANT EXECUTE ON procedure `mydb`.`crea\_utente` TO 'Manager';

GRANT EXECUTE ON procedure `mydb`.`Visualizza\_Pianta` TO 'Manager';

GRANT EXECUTE ON procedure `mydb`.`Visualizzare\_Colorazioni` TO 'Manager';

GRANT EXECUTE ON procedure `mydb`.`Aggiornare\_Giacenze` TO 'Manager';

GRANT EXECUTE ON procedure `mydb`.`visualizza\_totale\_ordine` TO 'Manager';

CREATE USER 'login' IDENTIFIED BY 'login';

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

SET SQL\_MODE=@OLD\_SQL\_MODE;

SET FOREIGN\_KEY\_CHECKS=@OLD\_FOREIGN\_KEY\_CHECKS;

SET UNIQUE\_CHECKS=@OLD\_UNIQUE\_CHECKS;

call mydb.crea\_utente('admin', 'admin', 'Manager');

INSERT INTO `mydb`.`Fornitori` (`idFornitori`, `cod\_fis`, `Nome\_fornitore`) VALUES ('1', 'aaabbb12c34d567e', 'Mario Rossi');

INSERT INTO `mydb`.`Fornitori` (`idFornitori`, `cod\_fis`, `Nome\_fornitore`) VALUES ('2', 'xxxyyy12z34t567e', 'Carlo Flora');

INSERT INTO `mydb`.`Fornitori` (`idFornitori`, `cod\_fis`, `Nome\_fornitore`) VALUES ('3', '01786610894', 'Piante co.');

INSERT INTO `mydb`.`Fornitori` (`idFornitori`, `cod\_fis`, `Nome\_fornitore`) VALUES ('4', '12345678910', 'EXPOLANT');

INSERT INTO `mydb`.`Fornitori` (`idFornitori`, `cod\_fis`, `Nome\_fornitore`) VALUES ('5', '10987654321', 'naturarbor');

INSERT INTO `mydb`.`INDIRIZZI` (`indirizzo`, `Fornitori\_idFornitori`) VALUES ('Via del corso, 1', '1');

INSERT INTO `mydb`.`INDIRIZZI` (`indirizzo`, `Fornitori\_idFornitori`) VALUES ('Via Brombeis, 8', '2');

INSERT INTO `mydb`.`INDIRIZZI` (`indirizzo`, `Fornitori\_idFornitori`) VALUES ('Via dei Tribunali, 5', '2');

INSERT INTO `mydb`.`INDIRIZZI` (`indirizzo`, `Fornitori\_idFornitori`) VALUES ('Via Po, 101', '3');

INSERT INTO `mydb`.`INDIRIZZI` (`indirizzo`, `Fornitori\_idFornitori`) VALUES ('Viale Crispi, 74', '3');

INSERT INTO `mydb`.`INDIRIZZI` (`indirizzo`, `Fornitori\_idFornitori`) VALUES ('Via Tuscolana, 404', '4');

INSERT INTO `mydb`.`INDIRIZZI` (`indirizzo`, `Fornitori\_idFornitori`) VALUES ('Via Casilina, 102', '4');

INSERT INTO `mydb`.`INDIRIZZI` (`indirizzo`, `Fornitori\_idFornitori`) VALUES ('Via Cassia, 104', '4');

INSERT INTO `mydb`.`INDIRIZZI` (`indirizzo`, `Fornitori\_idFornitori`) VALUES ('Via Prenestina, 500', '4');

INSERT INTO `mydb`.`INDIRIZZI` (`indirizzo`, `Fornitori\_idFornitori`) VALUES ('Viale Bovio, 15', '5');

INSERT INTO `mydb`.`INDIRIZZI` (`indirizzo`, `Fornitori\_idFornitori`) VALUES ('Via Cona, 19', '5');

## Codice del Front-End

**----------------------------------Defines.h----------------------------------**

#pragma once

#include <stdbool.h>

#include "mysql.h"

**struct** configuration {

**char** \*host;

**char** \*db\_username;

**char** \*db\_password;

**unsigned** **int** port;

**char** \*database;

**char** username[128];

**char** password[128];

};

**extern** **struct** configuration conf;

**extern** **int** parse\_config(**char** \*path, **struct** configuration \*conf);

**extern** **char** \*getInput(**unsigned** **int** lung, **char** \*stringa, **bool** hide);

**extern** **bool** yesOrNo(**char** \*domanda, **char** yes, **char** no, **bool** predef, **bool** insensitive);

**extern** **char** multiChoice(**char** \*domanda, **char** choices[], **int** num);

**extern** **void** print\_error (MYSQL \*conn, **char** \*message);

**extern** **void** print\_stmt\_error (MYSQL\_STMT \*stmt, **char** \*message);

**extern** **void** finish\_with\_error(MYSQL \*conn, **char** \*message);

**extern** **void** finish\_with\_stmt\_error(MYSQL \*conn, MYSQL\_STMT \*stmt, **char** \*message, **bool** close\_stmt);

**extern** **bool** setup\_prepared\_stmt(MYSQL\_STMT \*\*stmt, **char** \*statement, MYSQL \*conn);

**extern** **void** dump\_result\_set(MYSQL \*conn, MYSQL\_STMT \*stmt, **char** \*title);

**extern** **void** run\_as\_manager(MYSQL \*conn);

**extern** **void** run\_as\_operatore(MYSQL \*conn);

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

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <ctype.h>

#include "defines.h"

**void** flush\_stdin(){

**int** c;

**while** ((c = getchar()) != '\n' && c != EOF) { }

}

**void** print\_stmt\_error (MYSQL\_STMT \*stmt, **char** \*message)

{

fprintf (stderr, "%s\n", message);

**if** (stmt != **NULL**) {

fprintf (stderr, "Error %u (%s): %s\n",

mysql\_stmt\_errno (stmt),

mysql\_stmt\_sqlstate(stmt),

mysql\_stmt\_error (stmt));

}

}

**void** print\_error(MYSQL \*conn, **char** \*message)

{

fprintf (stderr, "%s\n", message);

**if** (conn != **NULL**) {

#if MYSQL\_VERSION\_ID >= 40101

fprintf (stderr, "Error %u (%s): %s\n",

mysql\_errno (conn), mysql\_sqlstate(conn), mysql\_error (conn));

#else

fprintf (stderr, "Error %u: %s\n",

mysql\_errno (conn), mysql\_error (conn));

#endif

}

}

**bool** setup\_prepared\_stmt(MYSQL\_STMT \*\*stmt, **char** \*statement, MYSQL \*conn)

{

**bool** update\_length = **true**;

\*stmt = mysql\_stmt\_init(conn);

**if** (\*stmt == **NULL**)

{

print\_error(conn, "Could not initialize statement handler");

**return** **false**;

}

**if** (mysql\_stmt\_prepare (\*stmt, statement, strlen(statement)) != 0) {

print\_stmt\_error(\*stmt, "Could not prepare statement");

**return** **false**;

}

mysql\_stmt\_attr\_set(\*stmt, STMT\_ATTR\_UPDATE\_MAX\_LENGTH, &update\_length);

**return** **true**;

}

**void** finish\_with\_error(MYSQL \*conn, **char** \*message)

{

print\_error(conn, message);

mysql\_close(conn);

exit(EXIT\_FAILURE);

}

**void** finish\_with\_stmt\_error(MYSQL \*conn, MYSQL\_STMT \*stmt, **char** \*message, **bool** close\_stmt)

{

print\_stmt\_error(stmt, message);

**if**(close\_stmt) mysql\_stmt\_close(stmt);

mysql\_close(conn);

exit(EXIT\_FAILURE);

}

**static** **void** print\_dashes(MYSQL\_RES \*res\_set)

{

MYSQL\_FIELD \*field;

**unsigned** **int** i, j;

mysql\_field\_seek(res\_set, 0);

putchar('+');

**for** (i = 0; i < mysql\_num\_fields(res\_set); i++) {

field = mysql\_fetch\_field(res\_set);

**for** (j = 0; j < field->length + 2; j++)

putchar('-');

putchar('+');

}

putchar('\n');

}

**static** **void** dump\_result\_set\_header(MYSQL\_RES \*res\_set)

{

MYSQL\_FIELD \*field;

**unsigned** **long** col\_len;

**unsigned** **int** i;

*/\* determine column display widths -- requires result set to be \*/*

*/\* generated with mysql\_store\_result(), not mysql\_use\_result() \*/*

mysql\_field\_seek (res\_set, 0);

**for** (i = 0; i < mysql\_num\_fields (res\_set); i++) {

field = mysql\_fetch\_field (res\_set);

col\_len = strlen(field->name);

**if** (col\_len < field->length)

col\_len = field->length;

**if** (col\_len < 4 && !IS\_NOT\_NULL(field->flags))

col\_len = 4; */\* 4 = length of the word "NULL" \*/*

field->length = col\_len; */\* reset column info \*/*

}

print\_dashes(res\_set);

putchar('|');

mysql\_field\_seek (res\_set, 0);

**for** (i = 0; i < mysql\_num\_fields(res\_set); i++) {

field = mysql\_fetch\_field(res\_set);

printf(" %-\*s |", (**int**)field->length, field->name);

}

putchar('\n');

print\_dashes(res\_set);

}

**void** dump\_result\_set(MYSQL \*conn, MYSQL\_STMT \*stmt, **char** \*title)

{

**int** i;

**int** status;

**int** num\_fields; */\* number of columns in result \*/*

MYSQL\_FIELD \*fields; */\* for result set metadata \*/*

MYSQL\_BIND \*rs\_bind; */\* for output buffers \*/*

MYSQL\_RES \*rs\_metadata;

MYSQL\_TIME \*date;

size\_t attr\_size;

*/\* Prefetch the whole result set. This in conjunction with*

*\* STMT\_ATTR\_UPDATE\_length set in `setup\_prepared\_stmt`*

*\* updates the result set metadata which are fetched in this*

*\* function, to allow to compute the actual max length of*

*\* the columns.*

*\*/*

**if** (mysql\_stmt\_store\_result(stmt)) {

fprintf(stderr, " mysql\_stmt\_execute(), 1 failed\n");

fprintf(stderr, " %s\n", mysql\_stmt\_error(stmt));

exit(0);

}

*/\* the column count is > 0 if there is a result set \*/*

*/\* 0 if the result is only the final status packet \*/*

num\_fields = mysql\_stmt\_field\_count(stmt);

**if** (num\_fields > 0) {

*/\* there is a result set to fetch \*/*

printf("%s\n", title);

**if**((rs\_metadata = mysql\_stmt\_result\_metadata(stmt)) == **NULL**) {

finish\_with\_stmt\_error(conn, stmt, "Unable to retrieve result metadata\n", **true**);

}

dump\_result\_set\_header(rs\_metadata);

fields = mysql\_fetch\_fields(rs\_metadata);

rs\_bind = (MYSQL\_BIND \*)malloc(**sizeof** (MYSQL\_BIND) \* num\_fields);

**if** (!rs\_bind) {

finish\_with\_stmt\_error(conn, stmt, "Cannot allocate output buffers\n", **true**);

}

memset(rs\_bind, 0, **sizeof** (MYSQL\_BIND) \* num\_fields);

*/\* set up and bind result set output buffers \*/*

**for** (i = 0; i < num\_fields; ++i) {

*// Properly size the parameter buffer*

**switch**(fields[i].type) {

**case** MYSQL\_TYPE\_DATE:

**case** MYSQL\_TYPE\_TIMESTAMP:

**case** MYSQL\_TYPE\_DATETIME:

**case** MYSQL\_TYPE\_TIME:

attr\_size = **sizeof**(MYSQL\_TIME);

**break**;

**case** MYSQL\_TYPE\_FLOAT:

attr\_size = **sizeof**(**float**);

**break**;

**case** MYSQL\_TYPE\_DOUBLE:

attr\_size = **sizeof**(**double**);

**break**;

**case** MYSQL\_TYPE\_TINY:

attr\_size = **sizeof**(**signed** **char**);

**break**;

**case** MYSQL\_TYPE\_SHORT:

**case** MYSQL\_TYPE\_YEAR:

attr\_size = **sizeof**(**short** **int**);

**break**;

**case** MYSQL\_TYPE\_LONG:

**case** MYSQL\_TYPE\_INT24:

attr\_size = **sizeof**(**int**);

**break**;

**case** MYSQL\_TYPE\_LONGLONG:

attr\_size = **sizeof**(**long** **long** **int**);

**break**;

**default**:

attr\_size = fields[i].length;

**break**;

}

*// Setup the binding for the current parameter*

rs\_bind[i].buffer\_type = fields[i].type;

rs\_bind[i].buffer = malloc(attr\_size + 1);

rs\_bind[i].buffer\_length = attr\_size + 1;

**if**(rs\_bind[i].buffer == **NULL**) {

finish\_with\_stmt\_error(conn, stmt, "Cannot allocate output buffers\n", **true**);

}

}

**if**(mysql\_stmt\_bind\_result(stmt, rs\_bind)) {

finish\_with\_stmt\_error(conn, stmt, "Unable to bind output parameters\n", **true**);

}

*/\* fetch and display result set rows \*/*

**while** (**true**) {

status = mysql\_stmt\_fetch(stmt);

**if** (status == 1 || status == MYSQL\_NO\_DATA)

**break**;

putchar('|');

**for** (i = 0; i < num\_fields; i++) {

**if** (rs\_bind[i].buffer\_type == MYSQL\_TYPE\_NULL) {

printf (" %-\*s |", (**int**)fields[i].length, "NULL");

**continue**;

}

**switch** (rs\_bind[i].buffer\_type) {

**case** MYSQL\_TYPE\_VAR\_STRING:

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

**break**;

**case** MYSQL\_TYPE\_DATETIME:

**case** MYSQL\_TYPE\_TIMESTAMP:

date = (MYSQL\_TIME \*)rs\_bind[i].buffer;

printf("%02d/%02d/%4d %02d:%02d:%02d |", date->day, date->month, date->year,date->hour,date->minute,date->second);

**break**;

**case** MYSQL\_TYPE\_TIME:

date = (MYSQL\_TIME \*)rs\_bind[i].buffer;

printf(" %02d-%02d-%02d | ", date->hour,date->minute,date->second);

**break**;

**case** MYSQL\_TYPE\_DATE:

date = (MYSQL\_TIME \*)rs\_bind[i].buffer;

printf(" %4d-%02d-%02d |", date->year, date->month, date->day);

**break**;

**case** MYSQL\_TYPE\_STRING:

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

**break**;

**case** MYSQL\_TYPE\_FLOAT:

**case** MYSQL\_TYPE\_DOUBLE:

printf(" %-\*.02f |", (**int**)fields[i].length,\*(**double** \*)rs\_bind[i].buffer);

**break**;

**case** MYSQL\_TYPE\_LONG:

printf(" %-\*d |", (**int**)fields[i].length, \*(**int** \*)rs\_bind[i].buffer);

**break**;

**case** MYSQL\_TYPE\_SHORT:

**case** MYSQL\_TYPE\_TINY:

**if**(\*(**signed** **char**\*)rs\_bind[i].buffer == 0){

printf("no |");

}

**else**{printf("si |");}

*// printf(" %-\*u |", (int)fields[i].max\_length, \*(signed char \*)rs\_bind[i].buffer);*

**break**;

**case** MYSQL\_TYPE\_LONGLONG:

printf(" %-\*lld |", (**int**)fields[i].length, \*(**long** **long** **int** \*)rs\_bind[i].buffer);

**break**;

**case** MYSQL\_TYPE\_NEWDECIMAL:

printf(" %-\*.02lf |", (**int**)fields[i].length, \*(**float**\*) rs\_bind[i].buffer);

**break**;

**case** MYSQL\_TYPE\_NULL:

printf("null");

**break**;

**default**:

printf("ERROR: Unhandled type (%d)\n", rs\_bind[i].buffer\_type);

abort();

}

}

putchar('\n');

print\_dashes(rs\_metadata);

}

mysql\_free\_result(rs\_metadata); */\* free metadata \*/*

*/\* free output buffers \*/*

**for** (i = 0; i < num\_fields; i++) {

free(rs\_bind[i].buffer);

}

free(rs\_bind);

}

}

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

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <mysql.h>

#include "defines.h"

**typedef** **enum** {

operatore = 1,

Manager = 2,

FAILED\_LOGIN = 3

} role\_t;

**struct** configuration conf;

**static** MYSQL \*conn;

**static** role\_t attempt\_login(MYSQL \*conn, **char** \*username, **char** \*password) {

MYSQL\_STMT \*login\_procedure;

MYSQL\_BIND param[3]; *// Used both for input and output*

**int** role = 0;

**if**(!setup\_prepared\_stmt(&login\_procedure, "call login(?, ?, ?)", conn)) {

print\_stmt\_error(login\_procedure, "Unable to initialize login statement\n");

**goto** err2;

}

*// Prepare parameters*

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

param[0].buffer\_type = MYSQL\_TYPE\_VAR\_STRING; *// IN*

param[0].buffer = username;

param[0].buffer\_length = strlen(username);

param[1].buffer\_type = MYSQL\_TYPE\_VAR\_STRING; *// IN*

param[1].buffer = password;

param[1].buffer\_length = strlen(password);

param[2].buffer\_type = MYSQL\_TYPE\_LONG; *// OUT*

param[2].buffer = &role;

param[2].buffer\_length = **sizeof**(role);

**if** (mysql\_stmt\_bind\_param(login\_procedure, param) != 0) { *// Note \_param*

print\_stmt\_error(login\_procedure, "Could not bind parameters for login");

**goto** err;

}

*// Run procedure*

**if** (mysql\_stmt\_execute(login\_procedure) != 0) {

print\_stmt\_error(login\_procedure, "Could not execute login procedure");

**goto** err;

}

*// Prepare output parameters*

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

param[0].buffer\_type = MYSQL\_TYPE\_LONG; *// OUT*

param[0].buffer = &role;

param[0].buffer\_length = **sizeof**(role);

**if**(mysql\_stmt\_bind\_result(login\_procedure, param)) {

print\_stmt\_error(login\_procedure, "Could not retrieve output parameter");

**goto** err;

}

*// Retrieve output parameter*

**if**(mysql\_stmt\_fetch(login\_procedure)) {

print\_stmt\_error(login\_procedure, "Could not buffer results");

**goto** err;

}

mysql\_stmt\_close(login\_procedure);

**return** role;

err:

mysql\_stmt\_close(login\_procedure);

err2:

**return** FAILED\_LOGIN;

}

**int** main(**void**) {

role\_t role;

**char** username[45];

**char** pswd[45];

**if**(!parse\_config("users/login.json", &conf)) {

fprintf(stderr, "Unable to load login configuration\n");

exit(EXIT\_FAILURE);

}

conn = mysql\_init (**NULL**);

**if** (conn == **NULL**) {

fprintf (stderr, "mysql\_init() failed (probably out of memory)\n");

exit(EXIT\_FAILURE);

}

**if** (mysql\_real\_connect(conn, "localhost", "root", "root1234", "mydb", 3306, **NULL**, CLIENT\_MULTI\_STATEMENTS | CLIENT\_MULTI\_RESULTS) == **NULL**) {

fprintf (stderr, "mysql\_real\_connect() failed\n");

mysql\_close (conn);

exit(EXIT\_FAILURE);

}

printf("--------Benvenuto nel database di Verde S.r.l--------\n \t --------Inserisci le tue credenziali--------\n");

printf("Username: ");

getInput(45, username, **false**);

printf("Password: ");

getInput(45, pswd, **true**);

role = attempt\_login(conn, username, pswd);

**switch**(role) {

**case** operatore:

run\_as\_operatore(conn);

**break**;

**case** Manager:

run\_as\_manager(conn);

**break**;

**case** FAILED\_LOGIN:

fprintf(stderr, "Invalid credentials\n");

exit(EXIT\_FAILURE);

**break**;

**default**:

fprintf(stderr, "Invalid condition at %s:%d\n", \_\_FILE\_\_, \_\_LINE\_\_);

abort();

}

printf("----------------------------- CIAO ---------------------------------------\n");

mysql\_close (conn);

}

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

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <mysql.h>

#include "defines.h"

**void** clear\_stdin(){

**int** a;

**while**((a = getchar()) != '\n' && a != EOF){}

}

**static** **void** Aggiungi\_Specie\_di\_Pianta(MYSQL \*conn){

MYSQL\_STMT \*prepared\_stmt;

MYSQL\_BIND param[8];

**char** Pianta[6];

**char** Nome\_comune[45];

**char** Nome\_latino[45];

**char** posizione[8];

**char** options[4] ={'1','2'};

**char** p;

**signed** **char** esotica;

**int** quantita;

**double** Costo;

**char** Fornitore[16];

printf("\nCodice Pianta: ");

getInput(6, Pianta, **false**);

printf("Nome Comune: ");

getInput(45, Nome\_comune, **false**);

printf("Nome Latino: ");

getInput(45, Nome\_latino, **false**);

printf("Posizione Pianta\n");

printf("\t1) interno\n");

printf("\t2) esterno\n");

p = multiChoice("Seleziona Posizione", options, 2);

**switch**(p) {

**case** '1':

strcpy(posizione, "interno");

**break**;

**case** '2':

strcpy(posizione, "esterno");

**break**;

**default**:

fprintf(stderr, "Invalid condition at %s:%d\n", \_\_FILE\_\_, \_\_LINE\_\_);

abort();

}

printf("La Pianta è esotica?\n");

p = multiChoice("1)No \t2)Si", options, 2);

**switch**(p) {

**case** '1':

esotica=0;

**break**;

**case** '2':

esotica=1;

**break**;

**default**:

fprintf(stderr, "Invalid condition at %s:%d\n", \_\_FILE\_\_, \_\_LINE\_\_);

abort();

}

printf("Quantità disponibile: ");

scanf("%d", &quantita);

clear\_stdin();

printf("Costo Pianta: ");

scanf("%lf", &Costo);

clear\_stdin();

printf("Indicare il codice del Fornitore: ");

getInput(17, Fornitore, **false**);

**if**(!setup\_prepared\_stmt(&prepared\_stmt, "call aggiungere\_specie\_pianta(?, ?, ?, ?, ?, ?, ?, ?)", conn)) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Impossibile inzializzare lo statement\n", **false**);

}

*// Prepare parameters*

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

param[0].buffer\_type = MYSQL\_TYPE\_VAR\_STRING; *//IN*

param[0].buffer = Pianta;

param[0].buffer\_length = strlen(Pianta);

param[1].buffer\_type = MYSQL\_TYPE\_VAR\_STRING; *//IN*

param[1].buffer = Nome\_comune;

param[1].buffer\_length = strlen(Nome\_comune);

param[2].buffer\_type = MYSQL\_TYPE\_VAR\_STRING; *//IN*

param[2].buffer = Nome\_latino;

param[2].buffer\_length = strlen(Nome\_latino);

param[3].buffer\_type =MYSQL\_TYPE\_VAR\_STRING ; *//IN*

param[3].buffer = posizione;

param[3].buffer\_length = strlen(posizione);

param[4].buffer\_type = MYSQL\_TYPE\_TINY; *//IN*

param[4].buffer = &esotica;

param[4].buffer\_length = **sizeof**(esotica);

param[5].buffer\_type = MYSQL\_TYPE\_LONG; *//IN*

param[5].buffer = &quantita;

param[5].buffer\_length = **sizeof**(quantita);

param[6].buffer\_type = MYSQL\_TYPE\_DOUBLE; *//IN*

param[6].buffer = &Costo;

param[6].buffer\_length = **sizeof**(Costo);

param[7].buffer\_type = MYSQL\_TYPE\_VAR\_STRING; *//IN*

param[7].buffer = Fornitore;

param[7].buffer\_length = strlen(Fornitore);

**if** (mysql\_stmt\_bind\_param(prepared\_stmt, param) != 0) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Could not bind parameters for user insertion\n", **true**);

}

*// Run procedure*

**if** (mysql\_stmt\_execute(prepared\_stmt) != 0) {

print\_stmt\_error (prepared\_stmt, "errore durante l'inseriemento");

} **else** {

printf("Pianta inserita correttamente\n");

}

mysql\_stmt\_close(prepared\_stmt);

}

**static** **void** Aggiungi\_Colore (MYSQL \*conn){

MYSQL\_STMT \*prepared\_stmt;

MYSQL\_BIND param[2];

**char** colorazione[20];

**char** pianta[7];

printf("\nInserisci Pianta: ");

getInput(6, pianta, **false**);

printf("Inserisci colorazione: ");

getInput(20, colorazione, **false**);

**if**(!setup\_prepared\_stmt(&prepared\_stmt, "call Aggiungi\_colore\_pianta(?, ?)", conn)) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Unable to initialize student insertion statement\n", **false**);

}

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

param[0].buffer\_type = MYSQL\_TYPE\_VAR\_STRING; *//IN*

param[0].buffer = colorazione;

param[0].buffer\_length = strlen(colorazione);

param[1].buffer\_type = MYSQL\_TYPE\_VAR\_STRING; *//IN*

param[1].buffer = pianta;

param[1].buffer\_length = strlen(pianta);

**if** (mysql\_stmt\_bind\_param(prepared\_stmt, param) != 0) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Impossibile associare i parametri.\n", **true**);

}

*// Run procedure*

**if** (mysql\_stmt\_execute(prepared\_stmt) != 0) {

print\_stmt\_error (prepared\_stmt, "Errore durante l'inserimento colorazione.");

} **else** {

printf("Colorazione aggiunta con successo.\n");

}

mysql\_stmt\_close(prepared\_stmt);

}

**static** **void** Modifica\_Prezzo (MYSQL \*conn){

MYSQL\_STMT \*prepared\_stmt;

MYSQL\_BIND param[2];

**char** pianta[6];

**double** Prezzo;

printf("\nInserisci Pianta: ");

getInput(6, pianta, **false**);

printf("Inserisci Prezzo nuovo: ");

scanf("%lf", &Prezzo);

clear\_stdin();

**if**(!setup\_prepared\_stmt(&prepared\_stmt, "call Modificare\_prezzo\_pianta(?, ?)", conn)) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Unable to initialize student insertion statement\n", **false**);

}

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

param[0].buffer\_type = MYSQL\_TYPE\_VAR\_STRING; *//IN*

param[0].buffer = pianta;

param[0].buffer\_length = strlen(pianta);

param[1].buffer\_type = MYSQL\_TYPE\_DOUBLE; *//IN*

param[1].buffer = &Prezzo;

param[1].buffer\_length = **sizeof**(Prezzo);

**if** (mysql\_stmt\_bind\_param(prepared\_stmt, param) != 0) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Impossibile associare i parametri.\n", **true**);

}

*// Run procedure*

**if** (mysql\_stmt\_execute(prepared\_stmt) != 0) {

print\_stmt\_error (prepared\_stmt, "Errore durante l'aggiornamento del prezzo.");

} **else** {

printf("Prezzo modificato con successo.\n");

}

mysql\_stmt\_close(prepared\_stmt);

}

**static** **void** crea\_utente (MYSQL \*conn){

MYSQL\_STMT \*prepared\_stmt;

MYSQL\_BIND param[3];

**char** username[45];

**char** password[32];

**char** tipo[10];

**char** options[4] ={'1','2'};

**char** p;

printf("\nInserisci Username: ");

getInput(45, username, **false**);

printf("Inserisci Password: ");

getInput(32, password, **false**);

printf("Seleziona Ruolo: \n");

printf("\t1) Manager\n");

printf("\t2) Operatore\n");

p = multiChoice("Seleziona Ruolo:", options, 2);

**switch**(p) {

**case** '1':

strcpy(tipo, "Manager");

**break**;

**case** '2':

strcpy(tipo, "operatore");

**break**;

**default**:

fprintf(stderr, "Invalid condition at %s:%d\n", \_\_FILE\_\_, \_\_LINE\_\_);

abort();

}

**if**(!setup\_prepared\_stmt(&prepared\_stmt, "call crea\_utente(?, ?, ?)", conn)) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Unable to initialize student insertion statement\n", **false**);

}

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

param[0].buffer\_type = MYSQL\_TYPE\_VAR\_STRING; *//IN*

param[0].buffer = username;

param[0].buffer\_length = strlen(username);

param[1].buffer\_type = MYSQL\_TYPE\_VAR\_STRING; *//IN*

param[1].buffer = password;

param[1].buffer\_length = strlen(password);

param[2].buffer\_type = MYSQL\_TYPE\_VAR\_STRING; *//IN*

param[2].buffer =tipo;

param[2].buffer\_length = strlen(tipo);

**if** (mysql\_stmt\_bind\_param(prepared\_stmt, param) != 0) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Impossibile associare i parametri.\n", **true**);

}

*// Run procedure*

**if** (mysql\_stmt\_execute(prepared\_stmt) != 0) {

print\_stmt\_error (prepared\_stmt, "Errore durante la creazione dell'utente.");

} **else** {

printf("Utente creato con successo.\n");

}

mysql\_stmt\_close(prepared\_stmt);

}

**static** **void** V\_costo\_ordine(MYSQL \*conn){

MYSQL\_STMT \*prepared\_stmt;

MYSQL\_BIND param[1];

**int** ordine;

printf("Inserisci num. Ordine: ");

scanf("%d", &ordine);

**if**(!setup\_prepared\_stmt(&prepared\_stmt, "call visualizza\_totale\_ordine(?)", conn)) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Unable to initialize student insertion statement\n", **false**);

}

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

param[0].buffer\_type = MYSQL\_TYPE\_LONG; *//IN*

param[0].buffer = &ordine;

param[0].buffer\_length = **sizeof**(ordine);

**if** (mysql\_stmt\_bind\_param(prepared\_stmt, param) != 0) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Impossibile associare i parametri.\n", **true**);

}

*// Run procedure*

**if** (mysql\_stmt\_execute(prepared\_stmt) != 0) {

print\_stmt\_error (prepared\_stmt, "Errore durante la visualizzazione delle costo totale dell'ordine.");

} **else** {

dump\_result\_set(conn, prepared\_stmt, "-------------Costo Totale Ordine-------------");

}

mysql\_stmt\_next\_result(prepared\_stmt);

mysql\_stmt\_close(prepared\_stmt);

}

**static** **void** V\_prezzi(MYSQL \*conn){

MYSQL\_STMT \*prepared\_stmt;

MYSQL\_BIND param[1];

**char** pianta[6];

printf("\nInserisci Pianta:");

getInput(6, pianta, **false**);

**if**(!setup\_prepared\_stmt(&prepared\_stmt, "call Visualizzare\_Storico\_Prezzo(?)", conn)) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Unable to initialize student insertion statement\n", **false**);

}

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

param[0].buffer\_type = MYSQL\_TYPE\_VAR\_STRING; *//IN*

param[0].buffer = pianta;

param[0].buffer\_length = strlen(pianta);

**if** (mysql\_stmt\_bind\_param(prepared\_stmt, param) != 0) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Impossibile associare i parametri.\n", **true**);

}

*// Run procedure*

**if** (mysql\_stmt\_execute(prepared\_stmt) != 0) {

print\_stmt\_error (prepared\_stmt, "Errore durante la visualizzazione dello storico dei prezzi della pianta.");

} **else**{

dump\_result\_set(conn, prepared\_stmt, "-------------Storico Prezzo Pianta-------------");

}

mysql\_stmt\_next\_result(prepared\_stmt);

mysql\_stmt\_close(prepared\_stmt);

}

**static** **void** V\_colorazione(MYSQL \*conn){

MYSQL\_STMT \*prepared\_stmt;

MYSQL\_BIND param[1];

**char** pianta[6];

printf("\nInserisci Pianta: ");

getInput(6, pianta, **false**);

**if**(!setup\_prepared\_stmt(&prepared\_stmt, "call Visualizzare\_Colorazioni(?)", conn)) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Unable to initialize student insertion statement\n", **false**);

}

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

param[0].buffer\_type = MYSQL\_TYPE\_VAR\_STRING; *//IN*

param[0].buffer = pianta;

param[0].buffer\_length = strlen(pianta);

**if** (mysql\_stmt\_bind\_param(prepared\_stmt, param) != 0) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Impossibile associare i parametri.\n", **true**);

}

*// Run procedure*

**if** (mysql\_stmt\_execute(prepared\_stmt) != 0) {

print\_stmt\_error (prepared\_stmt, "Errore durante la visualizzazione dei colori.");

} **else** {

dump\_result\_set(conn, prepared\_stmt, "-------------Visualizza Colorazioni della pianta-------------");

}

mysql\_stmt\_next\_result(prepared\_stmt);

mysql\_stmt\_close(prepared\_stmt);

}

**static** **void** V\_pianta(MYSQL \*conn){

MYSQL\_STMT \*prepared\_stmt;

MYSQL\_BIND param[1];

**char** pianta[6];

printf("\nInserisci Pianta: ");

getInput(6, pianta, **false**);

**if**(!setup\_prepared\_stmt(&prepared\_stmt, "call Visualizza\_Pianta(?)", conn)) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Unable to initialize student insertion statement\n", **false**);

}

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

param[0].buffer\_type = MYSQL\_TYPE\_VAR\_STRING; *//IN*

param[0].buffer = pianta;

param[0].buffer\_length = strlen(pianta);

**if** (mysql\_stmt\_bind\_param(prepared\_stmt, param) != 0) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Impossibile associare i parametri.\n", **true**);

}

*// Run procedure*

**if** (mysql\_stmt\_execute(prepared\_stmt) != 0) {

print\_stmt\_error (prepared\_stmt, "Errore durante la visualizzazione delle informazioni della pianta.");

} **else** {

dump\_result\_set(conn, prepared\_stmt, "-------------Visualizza Pianta-------------");

}

mysql\_stmt\_next\_result(prepared\_stmt);

mysql\_stmt\_close(prepared\_stmt);

}

**static** **void** V\_Fornitori(MYSQL \*conn){

MYSQL\_STMT \*prepared\_stmt;

MYSQL\_BIND param[1];

**char** pianta[6];

printf("\nInserisci Pianta: ");

getInput(6, pianta, **false**);

**if**(!setup\_prepared\_stmt(&prepared\_stmt, "call visualizza\_fornitori\_pianta(?)", conn)) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Unable to initialize student insertion statement\n", **false**);

}

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

param[0].buffer\_type = MYSQL\_TYPE\_VAR\_STRING; *//IN*

param[0].buffer = pianta;

param[0].buffer\_length = strlen(pianta);

**if** (mysql\_stmt\_bind\_param(prepared\_stmt, param) != 0) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Impossibile associare i parametri.\n", **true**);

}

*// Run procedure*

**if** (mysql\_stmt\_execute(prepared\_stmt) != 0) {

print\_stmt\_error (prepared\_stmt, "Errore durante la visualizzazione dei Fornitori.");

} **else** {

dump\_result\_set(conn, prepared\_stmt, "-------------Visualizza Fornitori Pianta-------------");

}

mysql\_stmt\_next\_result(prepared\_stmt);

mysql\_stmt\_close(prepared\_stmt);

}

**static** **void** Aggiorna\_Giacenza(MYSQL \*conn){

MYSQL\_STMT \*prepared\_stmt;

MYSQL\_BIND param[2];

**char** pianta[6];

**int** giacenza;

printf("\nInserisci Pianta: ");

getInput(6, pianta, **false**);

printf("Inserisci la quantità da aggiungere: ");

scanf("%d", &giacenza);

**if**(!setup\_prepared\_stmt(&prepared\_stmt, "call Aggiornare\_Giacenze(?, ?)", conn)) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Unable to initialize insertion statement\n", **false**);

}

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

param[0].buffer\_type = MYSQL\_TYPE\_VAR\_STRING; *//IN*

param[0].buffer = pianta;

param[0].buffer\_length = strlen(pianta);

param[1].buffer\_type = MYSQL\_TYPE\_LONG; *//IN*

param[1].buffer = &giacenza;

param[1].buffer\_length = **sizeof**(giacenza);

**if** (mysql\_stmt\_bind\_param(prepared\_stmt, param) != 0) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Impossibile associare i parametri.\n", **true**);

}

*// Run procedure*

**if** (mysql\_stmt\_execute(prepared\_stmt) != 0) {

print\_stmt\_error (prepared\_stmt, "Errore durante l'aggiornamento della Giacenza.");

} **else** {

printf("-------------Giacenza aggiornata-------------.\n");

}

mysql\_stmt\_close(prepared\_stmt);

}

**static** **void** Aggiungi\_fornitore\_specie(MYSQL \*conn){

MYSQL\_STMT \*prepared\_stmt;

MYSQL\_BIND param[2];

**char** pianta[6];

**int** Fornitore;

printf("\nInserisci Pianta: ");

getInput(6, pianta, **false**);

printf("Inserisci codice Fornitore: ");

scanf("%d", &Fornitore);

clear\_stdin();

**if**(!setup\_prepared\_stmt(&prepared\_stmt, "call aggiungi\_fornitore\_specie(?, ?)", conn)) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Unable to initialize insertion statement\n", **false**);

}

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

param[0].buffer\_type = MYSQL\_TYPE\_LONG; *//IN*

param[0].buffer = &Fornitore;

param[0].buffer\_length = **sizeof**(Fornitore);

param[1].buffer\_type = MYSQL\_TYPE\_VAR\_STRING; *//IN*

param[1].buffer = pianta;

param[1].buffer\_length = strlen(pianta);

**if** (mysql\_stmt\_bind\_param(prepared\_stmt, param) != 0) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Impossibile associare i parametri.\n", **true**);

}

*// Run procedure*

**if** (mysql\_stmt\_execute(prepared\_stmt) != 0) {

print\_stmt\_error (prepared\_stmt, "Errore durante l'aggiunta del fornitore.");

} **else** {

printf("-------------Fornitore aggiunto alla specie-------------.\n");

}

mysql\_stmt\_close(prepared\_stmt);

}

**void** run\_as\_manager(MYSQL \*conn){

**char** options[12] = {'1','2','3','4','5','6','7','8','9','a','b','e'};

**char** op;

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

**if**(!parse\_config("users/Manager.json", &conf)) {

fprintf(stderr, "Impossibile caricare info Manager\n");

exit(EXIT\_FAILURE);

}

**if**(mysql\_change\_user(conn, "root", "root1234", "mydb")) {

fprintf(stderr, "mysql\_change\_user() failed\n");

exit(EXIT\_FAILURE);

}

**while**(**true**) {

printf("----- Cosa desidera fare ? -----\n\n");

printf("1) Aggiungi Specie di Pianta\n");

printf("2) Aggiungi Colore a Pianta\n");

printf("3) Modifica Prezzo di una Pianta\n");

printf("4) Aggiungi un nuovo utente\n");

printf("5) Aggiorna Giacenze Magazzino\n");

printf("6) Visualizza storico dei prezzi\n");

printf("7) Visualizza fornitori di una specie\n");

printf("8) Visualizza Specie\n");

printf("9) Visualizza Colorazioni di una Specie\n");

printf("a) Visualizza Totale dell'ordine\n");

printf("b) Aggiungi Fornitore a Pianta\n");

printf("e) Esci\n\n");

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

**switch**(op) {

**case** '1':

Aggiungi\_Specie\_di\_Pianta(conn);

**break**;

**case** '2':

Aggiungi\_Colore(conn);

**break**;

**case** '3':

Modifica\_Prezzo(conn);

**break**;

**case** '4':

crea\_utente(conn);

**break**;

**case** '5':

Aggiorna\_Giacenza(conn);

**break**;

**case** '6':

V\_prezzi(conn);

**break**;

**case** '7':

V\_Fornitori(conn);

**break**;

**case** '8':

V\_pianta(conn);

**break**;

**case** '9':

V\_colorazione(conn);

**break**;

**case** 'a':

V\_costo\_ordine(conn);

**break**;

**case** 'b':

Aggiungi\_fornitore\_specie(conn);

**break**;

**case**'e':

**return**;

**default**:

fprintf(stderr, "Invalid condition at %s:%d\n", \_\_FILE\_\_, \_\_LINE\_\_);

abort();

}

}

}

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

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <mysql.h>

#include "defines.h"

**void** clear(){

**int** a;

**while**((a = getchar()) != '\n' && a != EOF){}

}

**static** **void** Aggiungi\_pianta(MYSQL \*conn, **int** ordine);

*//1*

**static** **void** Add\_cPrivato(MYSQL \*conn){

MYSQL\_STMT \*prepared\_stmt;

MYSQL\_BIND param[7];

**char** codice[16];

**char** nome[45];

**char** fatturazione[45];

**char** spedizione[45];

**char** residenza[45];

**char** tipo[6];

**char** options[4] ={'1','2', '3'};

**char** p;

**char** valore[45];

printf("\nInserisci Codice Fiscale Cliente: ");

getInput(16, codice, **false**);

printf("\nInserisci Nome Cliente: ");

getInput(45, nome, **false**);

printf("\nInserisci indirizzo fatturazione: ");

getInput(45, fatturazione, **false**);

printf("\nInserisci indirizzo spedizione: ");

getInput(45, spedizione, **false**);

printf("\nInserisci indirizzo residenza: ");

getInput(45, residenza, **false**);

printf("Seleziona Contatto\n");

printf("\t1) email\n");

printf("\t2) telefono\n");

printf("\t3) cellulare\n");

p = multiChoice("Seleziona Posizione", options, 3);

**switch**(p) {

**case** '1':

strcpy(tipo, "email");

**break**;

**case** '2':

strcpy(tipo, "tel");

**break**;

**case** '3':

strcpy(tipo, "cel");

**break**;

**default**:

fprintf(stderr, "Invalid condition at %s:%d\n", \_\_FILE\_\_, \_\_LINE\_\_);

abort();

}

printf("\nInserisci Contatto: ");

getInput(45, valore, **false**);

**if**(!setup\_prepared\_stmt(&prepared\_stmt, "call Aggiungi\_cliente\_privato(?, ?,?,?,?,?,?)", conn)) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Unable to initialize insertion statement\n", **false**);

}

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

param[0].buffer\_type = MYSQL\_TYPE\_VAR\_STRING; *//IN*

param[0].buffer = codice;

param[0].buffer\_length = strlen(codice);

param[1].buffer\_type = MYSQL\_TYPE\_VAR\_STRING; *//IN*

param[1].buffer = nome;

param[1].buffer\_length = strlen(nome);

param[2].buffer\_type = MYSQL\_TYPE\_VAR\_STRING; *//IN*

param[2].buffer = fatturazione;

param[2].buffer\_length = strlen(fatturazione);

param[3].buffer\_type =MYSQL\_TYPE\_VAR\_STRING ; *//IN*

param[3].buffer = spedizione;

param[3].buffer\_length = strlen(spedizione);

param[4].buffer\_type = MYSQL\_TYPE\_VAR\_STRING; *//IN*

param[4].buffer = residenza;

param[4].buffer\_length = strlen(residenza);

param[5].buffer\_type = MYSQL\_TYPE\_VAR\_STRING; *//IN*

param[5].buffer = tipo;

param[5].buffer\_length = strlen(tipo);

param[6].buffer\_type = MYSQL\_TYPE\_VAR\_STRING; *//IN*

param[6].buffer = valore;

param[6].buffer\_length = strlen(valore);

**if** (mysql\_stmt\_bind\_param(prepared\_stmt, param) != 0) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Impossibile associare i parametri.\n", **true**);

}

*// Run procedure*

**if** (mysql\_stmt\_execute(prepared\_stmt) != 0) {

print\_stmt\_error (prepared\_stmt, "Errore durante l'inserimento.");

} **else** {

printf("-------------Cliente aggiunto con successo-------------\n");

}

mysql\_stmt\_close(prepared\_stmt);

}

*//2*

**static** **void** Add\_cRivendita(MYSQL \*conn){

MYSQL\_STMT \*prepared\_stmt;

MYSQL\_BIND param[9];

**char** codice[16];

**char** nome[45];

**char** fatturazione[45];

**char** spedizione[45];

**char** residenza[45];

**char** tipo[6];

**char** options[4] ={'1','2', '3'};

**char** p;

**char** valore[45];

**char** nome\_referente[45];

**char** cognome\_referente[45];

printf("\nInserisci Partita IVA Cliente: ");

getInput(16, codice, **false**);

printf("\nInserisci Nome Rivendita: ");

getInput(45, nome, **false**);

printf("\nInserisci indirizzo fatturazione: ");

getInput(45, fatturazione, **false**);

printf("\nInserisci indirizzo spedizione: ");

getInput(45, spedizione, **false**);

printf("\nInserisci indirizzo residenza: ");

getInput(45, residenza, **false**);

printf("Seleziona Contatto\n");

printf("\t1) email\n");

printf("\t2) telefono\n");

printf("\t3) cellulare\n");

p = multiChoice("Seleziona Posizione", options, 3);

**switch**(p) {

**case** '1':

strcpy(tipo, "email");

**break**;

**case** '2':

strcpy(tipo, "tel");

**break**;

**case** '3':

strcpy(tipo, "cel");

**break**;

**default**:

fprintf(stderr, "Invalid condition at %s:%d\n", \_\_FILE\_\_, \_\_LINE\_\_);

abort();

}

printf("\nInserisci Contatto: ");

getInput(45, valore, **false**);

printf("\nInserisci Nome Referente della Rivendita: ");

getInput(45, nome\_referente, **false**);

printf("\nInserisci Cognome Referente della Rivendita: ");

getInput(45, cognome\_referente, **false**);

**if**(!setup\_prepared\_stmt(&prepared\_stmt, "call Aggiungi\_cliente\_rivendita(?,?,?,?,?,?,?,?,?)", conn)) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Unable to initialize insertion statement\n", **false**);

}

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

param[0].buffer\_type = MYSQL\_TYPE\_VAR\_STRING; *//IN*

param[0].buffer = codice;

param[0].buffer\_length = strlen(codice);

param[1].buffer\_type = MYSQL\_TYPE\_VAR\_STRING; *//IN*

param[1].buffer = nome;

param[1].buffer\_length = strlen(nome);

param[2].buffer\_type = MYSQL\_TYPE\_VAR\_STRING; *//IN*

param[2].buffer = fatturazione;

param[2].buffer\_length = strlen(fatturazione);

param[3].buffer\_type =MYSQL\_TYPE\_VAR\_STRING ; *//IN*

param[3].buffer = spedizione;

param[3].buffer\_length = strlen(spedizione);

param[4].buffer\_type = MYSQL\_TYPE\_VAR\_STRING; *//IN*

param[4].buffer = residenza;

param[4].buffer\_length = strlen(residenza);

param[5].buffer\_type = MYSQL\_TYPE\_VAR\_STRING; *//IN*

param[5].buffer = tipo;

param[5].buffer\_length = strlen(tipo);

param[6].buffer\_type = MYSQL\_TYPE\_VAR\_STRING; *//IN*

param[6].buffer = valore;

param[6].buffer\_length = strlen(valore);

param[7].buffer\_type = MYSQL\_TYPE\_VAR\_STRING; *//IN*

param[7].buffer = nome\_referente;

param[7].buffer\_length = strlen(nome\_referente);

param[8].buffer\_type = MYSQL\_TYPE\_VAR\_STRING; *//IN*

param[8].buffer = cognome\_referente;

param[8].buffer\_length = strlen(cognome\_referente);

**if** (mysql\_stmt\_bind\_param(prepared\_stmt, param) != 0) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Impossibile associare i parametri.\n", **true**);

}

*// Run procedure*

**if** (mysql\_stmt\_execute(prepared\_stmt) != 0) {

print\_stmt\_error (prepared\_stmt, "Errore durante l'inserimento.");

} **else** {

printf("-------------Cliente aggiunto con successo-------------\n");

}

mysql\_stmt\_close(prepared\_stmt);

}

*//3*

**static** **void** Add\_contatti(MYSQL \*conn){

MYSQL\_STMT \*prepared\_stmt;

MYSQL\_BIND param[3];

**char** cliente[45];

**char** tipo[6];

**char** options[4] ={'1','2', '3'};

**char** p;

**char** valore[45];

printf("\nInserisci Codice del Cliente: ");

getInput(45, cliente, **false**);

printf("Seleziona Contatto\n");

printf("\t1) email\n");

printf("\t2) telefono\n");

printf("\t3) cellulare\n");

p = multiChoice("Seleziona Posizione", options, 3);

**switch**(p) {

**case** '1':

strcpy(tipo, "email");

**break**;

**case** '2':

strcpy(tipo, "tel");

**break**;

**case** '3':

strcpy(tipo, "cel");

**break**;

**default**:

fprintf(stderr, "Invalid condition at %s:%d\n", \_\_FILE\_\_, \_\_LINE\_\_);

abort();

}

printf("\nInserisci Contatto: ");

getInput(45, valore, **false**);

**if**(!setup\_prepared\_stmt(&prepared\_stmt, "call aggiungi\_contatto(?,?,?)", conn)) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Unable to initialize insertion statement\n", **false**);

}

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

param[0].buffer\_type = MYSQL\_TYPE\_VAR\_STRING; *//IN*

param[0].buffer = cliente;

param[0].buffer\_length = strlen(cliente);

param[1].buffer\_type = MYSQL\_TYPE\_VAR\_STRING; *//IN*

param[1].buffer = tipo;

param[1].buffer\_length = strlen(tipo);

param[2].buffer\_type = MYSQL\_TYPE\_VAR\_STRING; *//IN*

param[2].buffer = valore;

param[2].buffer\_length = strlen(valore);

**if** (mysql\_stmt\_bind\_param(prepared\_stmt, param) != 0) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Impossibile associare i parametri.\n", **true**);

}

*// Run procedure*

**if** (mysql\_stmt\_execute(prepared\_stmt) != 0) {

print\_stmt\_error (prepared\_stmt, "Errore durante l'inserimento.");

} **else** {

printf("-------------Contatto aggiunto con successo-------------\n");

}

mysql\_stmt\_close(prepared\_stmt);

}

*//4*

**static** **void** scegli\_Contatto(MYSQL \*conn){

MYSQL\_STMT \*prepared\_stmt;

MYSQL\_BIND param[2];

**char** cliente[16];

**char** valore[45];

printf("Inserisci Cliente: ");

getInput(16, cliente, **false**);

printf("Inserisci Contatto Preferito:");

getInput(45, valore, **false**);

**if**(!setup\_prepared\_stmt(&prepared\_stmt, "call scegli\_contatto\_preferito(?, ?)", conn)) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Unable to initialize student insertion statement\n", **false**);

}

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

param[0].buffer\_type = MYSQL\_TYPE\_VAR\_STRING; *//IN*

param[0].buffer = cliente;

param[0].buffer\_length = strlen(cliente);

param[1].buffer\_type = MYSQL\_TYPE\_VAR\_STRING; *//IN*

param[1].buffer = valore;

param[1].buffer\_length = strlen(valore);

**if** (mysql\_stmt\_bind\_param(prepared\_stmt, param) != 0) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Impossibile associare i parametri.\n", **true**);

}

*// Run procedure*

**if** (mysql\_stmt\_execute(prepared\_stmt) != 0) {

print\_stmt\_error (prepared\_stmt, "Errore durante l'aggiornamento del contatto preferito.");

} **else** {

printf("-------------Contatto preferito aggiornato con successo-------------\n");

}

mysql\_stmt\_close(prepared\_stmt);

}

*//5*

**static** **void** Add\_ordine(MYSQL \*conn){

MYSQL\_STMT \*prepared\_stmt;

MYSQL\_BIND param[2];

**int** ordine;

**char** cliente[17];

printf("\nInserisci num. Ordine: ");

scanf("%d", &ordine);

clear();

printf("Inserisci Cliente: ");

getInput(17, cliente, **false**);

clear();

**if**(!setup\_prepared\_stmt(&prepared\_stmt, "call Aggiungi\_ordine(?, ?)", conn)) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Unable to initialize student insertion statement\n", **false**);

}

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

param[0].buffer\_type = MYSQL\_TYPE\_LONG; *//IN*

param[0].buffer = &ordine;

param[0].buffer\_length = **sizeof**(ordine);

param[1].buffer\_type = MYSQL\_TYPE\_VAR\_STRING; *//IN*

param[1].buffer = cliente;

param[1].buffer\_length = strlen(cliente);

**if** (mysql\_stmt\_bind\_param(prepared\_stmt, param) != 0) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Impossibile associare i parametri.\n", **true**);

}

*// Run procedure*

**if** (mysql\_stmt\_execute(prepared\_stmt) != 0) {

print\_stmt\_error (prepared\_stmt, "Errore durante l'inserimento dell'ordine.");

}**else**{

printf("-------------Ordine Registrato-------------\n");

**int** i=0;

**int** loop;

printf("Quante piante si vogliono inserire: ");

scanf("%d", &loop);

**while**(loop>i){

Aggiungi\_pianta(conn, ordine);

i++;

}

}

}

*//6*

**static** **void** Aggiungi\_pianta(MYSQL \*conn, **int** ordine){

MYSQL\_STMT \*prepared\_stmt;

MYSQL\_BIND param[3];

**char** pianta[6];

**int** quantita;

printf("\nInsersci codice Pianta: " );

scanf("%s", pianta);

clear();

printf("Inserisci quantità acquistata: ");

scanf("%d", &quantita);

clear();

**if**(!setup\_prepared\_stmt(&prepared\_stmt, "call agg\_piante\_ordine(?,?,?)", conn)) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Unable to initialize insertion statement\n", **false**);

}

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

param[0].buffer\_type = MYSQL\_TYPE\_LONG; *//IN*

param[0].buffer = &ordine;

param[0].buffer\_length = **sizeof**(ordine);

param[1].buffer\_type = MYSQL\_TYPE\_VAR\_STRING; *//IN*

param[1].buffer = pianta;

param[1].buffer\_length = strlen(pianta);

param[2].buffer\_type = MYSQL\_TYPE\_LONG; *//IN*

param[2].buffer = &quantita;

param[2].buffer\_length = **sizeof**(quantita);

**if** (mysql\_stmt\_bind\_param(prepared\_stmt, param) != 0) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Impossibile associare i parametri.\n", **true**);

}

*// Run procedure*

**if** (mysql\_stmt\_execute(prepared\_stmt) != 0) {

print\_stmt\_error (prepared\_stmt, "Errore durante l'inserimento.");

} **else** {

printf("-------------Pianta aggiunta con successo-------------\n");

}

mysql\_stmt\_close(prepared\_stmt);

}

*//7*

**static** **void** Aggiungi\_pianta\_pacco(MYSQL \*conn, **int** pacco){

MYSQL\_STMT \*prepared\_stmt;

MYSQL\_BIND param[3];

**char** pianta[6];

printf("\nInsersci codice Pianta: " );

scanf("%s", pianta);

clear();

**if**(!setup\_prepared\_stmt(&prepared\_stmt, "call Aggiungi\_piante\_a\_pacco(?,?)", conn)) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Unable to initialize insertion statement\n", **false**);

}

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

param[0].buffer\_type = MYSQL\_TYPE\_LONG; *//IN*

param[0].buffer = &pacco;

param[0].buffer\_length = **sizeof**(pacco);

param[1].buffer\_type = MYSQL\_TYPE\_VAR\_STRING; *//IN*

param[1].buffer = pianta;

param[1].buffer\_length = strlen(pianta);

**if** (mysql\_stmt\_bind\_param(prepared\_stmt, param) != 0) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Impossibile associare i parametri.\n", **true**);

}

*// Run procedure*

**if** (mysql\_stmt\_execute(prepared\_stmt) != 0) {

print\_stmt\_error (prepared\_stmt, "Errore durante l'inserimento.");

} **else** {

printf("-------------Pianta aggiunta con successo-------------\n");

}

mysql\_stmt\_close(prepared\_stmt);

}

*//8*

**static** **void** Add\_pacco(MYSQL \*conn){

MYSQL\_STMT \*prepared\_stmt;

MYSQL\_BIND param[2];

**int** ordine;

**int** pacco;

printf("\nInserisci num. Ordine: ");

scanf("%d", &ordine);

clear();

printf("Inserisci numero Pacco: ");

scanf("%d", &pacco);

clear();

**if**(!setup\_prepared\_stmt(&prepared\_stmt, "call Aggiungi\_pacco(?, ?)", conn)) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Unable to initialize student insertion statement\n", **false**);

}

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

param[0].buffer\_type = MYSQL\_TYPE\_LONG; *//IN*

param[0].buffer = &ordine;

param[0].buffer\_length = **sizeof**(ordine);

param[1].buffer\_type = MYSQL\_TYPE\_LONG; *//IN*

param[1].buffer = &pacco;

param[1].buffer\_length = **sizeof**(pacco);

**if** (mysql\_stmt\_bind\_param(prepared\_stmt, param) != 0) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Impossibile associare i parametri.\n", **true**);

}

*// Run procedure*

**if** (mysql\_stmt\_execute(prepared\_stmt) != 0) {

print\_stmt\_error (prepared\_stmt, "Errore durante la creazione del pacco.");

}**else**{

printf("-------------Pacco creato con successo-------------\n");

**int** i=0;

**int** loop;

printf("Quante specie di piante si vogliono inserire nel pacco %d: ", pacco);

scanf("%d", &loop);

**while**(loop>i){

Aggiungi\_pianta\_pacco(conn, pacco);

i++;

}

}

}

*//9*

**static** **void** V\_ordine(MYSQL \*conn){

MYSQL\_STMT \*prepared\_stmt;

MYSQL\_BIND param[1];

**int** ordine;

printf("Inserisci num. Ordine: ");

scanf("%d", &ordine);

**if**(!setup\_prepared\_stmt(&prepared\_stmt, "call Visualizza\_ordine(?)", conn)) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Unable to initialize student insertion statement\n", **false**);

}

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

param[0].buffer\_type = MYSQL\_TYPE\_LONG; *//IN*

param[0].buffer = &ordine;

param[0].buffer\_length = **sizeof**(ordine);

**if** (mysql\_stmt\_bind\_param(prepared\_stmt, param) != 0) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Impossibile associare i parametri.\n", **true**);

}

*// Run procedure*

**if** (mysql\_stmt\_execute(prepared\_stmt) != 0) {

print\_stmt\_error (prepared\_stmt, "Errore durante la visualizzazione dell'ordine.");

} **else** {

dump\_result\_set(conn, prepared\_stmt, "-------------Visualizza informazioni ordine-------------");

}

mysql\_stmt\_next\_result(prepared\_stmt);

mysql\_stmt\_close(prepared\_stmt);

}

*//10*

**static** **void** V\_Ordini\_Cliente(MYSQL \*conn){

MYSQL\_STMT \*prepared\_stmt;

MYSQL\_BIND param[1];

**char** cliente[16];

printf("\nInserisci codice Cliente: ");

getInput(16, cliente, **false**);

**if**(!setup\_prepared\_stmt(&prepared\_stmt, "call Visualizza\_ordini\_cliente(?)", conn)) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Unable to initialize student insertion statement\n", **false**);

}

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

param[0].buffer\_type = MYSQL\_TYPE\_VAR\_STRING; *//IN*

param[0].buffer = cliente;

param[0].buffer\_length = strlen(cliente);

**if** (mysql\_stmt\_bind\_param(prepared\_stmt, param) != 0) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Impossibile associare i parametri.\n", **true**);

}

*// Run procedure*

**if** (mysql\_stmt\_execute(prepared\_stmt) != 0) {

print\_stmt\_error (prepared\_stmt, "Errore durante la visualizzazione degli ordini del cliente.");

} **else** {

dump\_result\_set(conn, prepared\_stmt, "-------------Visualizza Ordini di un Cliente-------------");

}

mysql\_stmt\_next\_result(prepared\_stmt);

mysql\_stmt\_close(prepared\_stmt);

}

*//11*

**static** **void** V\_pianta(MYSQL \*conn){

MYSQL\_STMT \*prepared\_stmt;

MYSQL\_BIND param[1];

**char** pianta[6];

printf("\nInserisci Pianta: ");

getInput(6, pianta, **false**);

**if**(!setup\_prepared\_stmt(&prepared\_stmt, "call Visualizza\_Pianta(?)", conn)) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Unable to initialize student insertion statement\n", **false**);

}

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

param[0].buffer\_type = MYSQL\_TYPE\_VAR\_STRING; *//IN*

param[0].buffer = pianta;

param[0].buffer\_length = strlen(pianta);

**if** (mysql\_stmt\_bind\_param(prepared\_stmt, param) != 0) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Impossibile associare i parametri.\n", **true**);

}

*// Run procedure*

**if** (mysql\_stmt\_execute(prepared\_stmt) != 0) {

print\_stmt\_error (prepared\_stmt, "Errore durante la visualizzazione delle informazioni della pianta.");

} **else** {

dump\_result\_set(conn, prepared\_stmt, "-------------Visualizza Pianta-------------");

}

mysql\_stmt\_next\_result(prepared\_stmt);

mysql\_stmt\_close(prepared\_stmt);

}

*//12*

**static** **void** V\_colorazione(MYSQL \*conn){

MYSQL\_STMT \*prepared\_stmt;

MYSQL\_BIND param[1];

**char** pianta[6];

printf("\nInserisci Pianta: ");

getInput(6, pianta, **false**);

**if**(!setup\_prepared\_stmt(&prepared\_stmt, "call Visualizzare\_Colorazioni(?)", conn)) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Unable to initialize student insertion statement\n", **false**);

}

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

param[0].buffer\_type = MYSQL\_TYPE\_VAR\_STRING; *//IN*

param[0].buffer = pianta;

param[0].buffer\_length = strlen(pianta);

**if** (mysql\_stmt\_bind\_param(prepared\_stmt, param) != 0) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Impossibile associare i parametri.\n", **true**);

}

*// Run procedure*

**if** (mysql\_stmt\_execute(prepared\_stmt) != 0) {

print\_stmt\_error (prepared\_stmt, "Errore durante la visualizzazione dei colori.");

} **else** {

dump\_result\_set(conn, prepared\_stmt, "-------------Visualizza Colorazioni della pianta-------------");

}

mysql\_stmt\_next\_result(prepared\_stmt);

mysql\_stmt\_close(prepared\_stmt);

}

*//13*

**static** **void** V\_prezzi(MYSQL \*conn){

MYSQL\_STMT \*prepared\_stmt;

MYSQL\_BIND param[1];

**char** pianta[6];

printf("\nInserisci Pianta:");

getInput(6, pianta, **false**);

**if**(!setup\_prepared\_stmt(&prepared\_stmt, "call Visualizzare\_Storico\_Prezzo(?)", conn)) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Unable to initialize student insertion statement\n", **false**);

}

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

param[0].buffer\_type = MYSQL\_TYPE\_VAR\_STRING; *//IN*

param[0].buffer = pianta;

param[0].buffer\_length = strlen(pianta);

**if** (mysql\_stmt\_bind\_param(prepared\_stmt, param) != 0) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Impossibile associare i parametri.\n", **true**);

}

*// Run procedure*

**if** (mysql\_stmt\_execute(prepared\_stmt) != 0) {

print\_stmt\_error (prepared\_stmt, "Errore durante la visualizzazione dello storico dei prezzi della pianta.");

} **else**{

dump\_result\_set(conn, prepared\_stmt, "-------------Storico Prezzo Pianta-------------");

}

mysql\_stmt\_next\_result(prepared\_stmt);

mysql\_stmt\_close(prepared\_stmt);

}

*//14*

**static** **void** V\_piante\_inserire(MYSQL \*conn){

MYSQL\_STMT \*prepared\_stmt;

MYSQL\_BIND param[1];

**int** ordine;

printf("Inserisci num. Ordine: ");

scanf("%d", &ordine);

**if**(!setup\_prepared\_stmt(&prepared\_stmt, "call visualizzare\_piante\_inserire\_pacco(?)", conn)) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Unable to initialize student insertion statement\n", **false**);

}

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

param[0].buffer\_type = MYSQL\_TYPE\_LONG; *//IN*

param[0].buffer = &ordine;

param[0].buffer\_length = **sizeof**(ordine);

**if** (mysql\_stmt\_bind\_param(prepared\_stmt, param) != 0) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Impossibile associare i parametri.\n", **true**);

}

*// Run procedure*

**if** (mysql\_stmt\_execute(prepared\_stmt) != 0) {

print\_stmt\_error (prepared\_stmt, "Errore durante la visualizzazione delle piante da inserire nel pacco.");

} **else** {

dump\_result\_set(conn, prepared\_stmt, "-------------Piante da Inserire nel Pacco-------------");

}

mysql\_stmt\_next\_result(prepared\_stmt);

mysql\_stmt\_close(prepared\_stmt);

}

*//15*

**static** **void** V\_piante\_inserite(MYSQL \*conn){

MYSQL\_STMT \*prepared\_stmt;

MYSQL\_BIND param[1];

**int** Pacco;

printf("Inserisci num. Pacco: ");

scanf("%d", &Pacco);

**if**(!setup\_prepared\_stmt(&prepared\_stmt, "call Visualizzare\_piante\_Pacco(?)", conn)) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Unable to initialize student insertion statement\n", **false**);

}

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

param[0].buffer\_type = MYSQL\_TYPE\_LONG; *//IN*

param[0].buffer = &Pacco;

param[0].buffer\_length = **sizeof**(Pacco);

**if** (mysql\_stmt\_bind\_param(prepared\_stmt, param) != 0) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Impossibile associare i parametri.\n", **true**);

}

*// Run procedure*

**if** (mysql\_stmt\_execute(prepared\_stmt) != 0) {

print\_stmt\_error (prepared\_stmt, "Errore durante la visualizzazione delle piante inserite nel pacco.");

} **else** {

dump\_result\_set(conn, prepared\_stmt, "-------------Piante inserite nel Pacco-------------");

}

mysql\_stmt\_next\_result(prepared\_stmt);

mysql\_stmt\_close(prepared\_stmt);

}

*//16*

**static** **void** V\_costo\_ordine(MYSQL \*conn){

MYSQL\_STMT \*prepared\_stmt;

MYSQL\_BIND param[1];

**int** ordine;

printf("Inserisci num. Ordine: ");

scanf("%d", &ordine);

**if**(!setup\_prepared\_stmt(&prepared\_stmt, "call visualizza\_totale\_ordine(?)", conn)) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Unable to initialize student insertion statement\n", **false**);

}

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

param[0].buffer\_type = MYSQL\_TYPE\_LONG; *//IN*

param[0].buffer = &ordine;

param[0].buffer\_length = **sizeof**(ordine);

**if** (mysql\_stmt\_bind\_param(prepared\_stmt, param) != 0) {

finish\_with\_stmt\_error(conn, prepared\_stmt, "Impossibile associare i parametri.\n", **true**);

}

*// Run procedure*

**if** (mysql\_stmt\_execute(prepared\_stmt) != 0) {

print\_stmt\_error (prepared\_stmt, "Errore durante la visualizzazione delle costo totale dell'ordine.");

} **else** {

dump\_result\_set(conn, prepared\_stmt, "-------------Costo Totale Ordine-------------");

}

mysql\_stmt\_next\_result(prepared\_stmt);

mysql\_stmt\_close(prepared\_stmt);

}

**void** run\_as\_operatore(MYSQL \*conn){

**char** options[17]={'1','2','3','4','5','6','7','8','9','a','b','c','d','e','f','g', 'h'};

**char** op;

**int** ordine, pacco;

**int** i, loop;

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

**if**(!parse\_config("users/operatore.json", &conf)) {

fprintf(stderr, "Impossibile caricare info Operatore\n");

exit(EXIT\_FAILURE);

}

**if**(mysql\_change\_user(conn, "root", "root1234", "mydb")) {

fprintf(stderr, "mysql\_change\_user() failed\n");

exit(EXIT\_FAILURE);

}

**while**(**true**) {

printf("\*\*\* Cosa desidera fare ? \*\*\*\n\n");

printf("1) Aggiungi Cliente Privato\n");

printf("2) Modifica Cliente Rivendita\n");

printf("3) Aggiungi Contatti\n");

printf("4) Scegli Contatto Preferito del Cliente\n");

printf("5) Aggiungi Ordine\n");

printf("6) Aggiungere Piante ad Ordine\n");

printf("7) Aggiungi Pacco\n");

printf("8) Inserisci Piante in un Pacco\n");

printf("9) Visualizza Ordine\n");

printf("a) Visualizza Ordini Cliente\n");

printf("b) Visualizza Pianta\n");

printf("c) Visualizza Colorazione delle Pianta\n");

printf("d) Visualizzare Storico Prezzi di Pianta\n");

printf("e) Visualizzare Piante da inserire nel Pacco\n");

printf("f) Visualizza Piante del Pacco\n");

printf("g) Visualizza Costo Totale Ordine\n");

printf("h) Esci\n");

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

**switch**(op) {

**case** '1':

Add\_cPrivato(conn);

**break**;

**case** '2':

Add\_cRivendita(conn);

**break**;

**case** '3':

Add\_contatti(conn);

**break**;

**case** '4':

scegli\_Contatto(conn);

**break**;

**case** '5':

Add\_ordine(conn);

**break**;

**case** '6':

printf("Inserire numero dell'ordine: ");

scanf("%d", &ordine);

i=0;

printf("Quante piante si vogliono inserire: ");

scanf("%d", &loop);

**while**(loop>i){

Aggiungi\_pianta(conn, ordine);

i++;

}

**break**;

**case** '7':

Add\_pacco(conn);

**break**;

**case** '8':

printf("Inserire numero del Pacco: ");

scanf("%d", &pacco);

i=0;

**int** loop;

printf("Quante specie di piante si vogliono inserire nel pacco %d: ", pacco);

scanf("%d", &loop);

**while**(loop>i){

Aggiungi\_pianta\_pacco(conn, pacco);

i++;

}

**break**;

**case** '9':

V\_ordine(conn);

**break**;

**case** 'a':

V\_Ordini\_Cliente(conn);

**break**;

**case** 'b':

V\_pianta(conn);

**break**;

**case** 'c':

V\_colorazione(conn);

**break**;

**case** 'd':

V\_prezzi(conn);

**break**;

**case** 'e':

V\_piante\_inserire(conn);

**break**;

**case** 'f':

V\_piante\_inserite(conn);

**break**;

**case** 'g':

V\_costo\_ordine(conn);

**break**;

**case** 'h':

**return**;

**default**:

fprintf(stderr, "Invalid condition at %s:%d\n", \_\_FILE\_\_, \_\_LINE\_\_);

abort();

}

}

}

1. Indicare con E le entità, con R le relazioni [↑](#footnote-ref-1)
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. [↑](#footnote-ref-2)
3. PK = primary key, NN = not null, UQ = unique, UN = unsigned, AI = auto increment. È ovviamente possibile specificare più di un attributo per ciascuna colonna. [↑](#footnote-ref-3)
4. PK = primary key, NN = not null, UQ = unique, UN = unsigned, AI = auto increment. È ovviamente possibile specificare più di un attributo per ciascuna colonna. [↑](#footnote-ref-4)
5. PK = primary key, NN = not null, UQ = unique, UN = unsigned, AI = auto increment. È ovviamente possibile specificare più di un attributo per ciascuna colonna. [↑](#footnote-ref-5)
6. PK = primary key, NN = not null, UQ = unique, UN = unsigned, AI = auto increment. È ovviamente possibile specificare più di un attributo per ciascuna colonna. [↑](#footnote-ref-6)
7. PK = primary key, NN = not null, UQ = unique, UN = unsigned, AI = auto increment. È ovviamente possibile specificare più di un attributo per ciascuna colonna. [↑](#footnote-ref-7)
8. PK = primary key, NN = not null, UQ = unique, UN = unsigned, AI = auto increment. È ovviamente possibile specificare più di un attributo per ciascuna colonna. [↑](#footnote-ref-8)
9. PK = primary key, NN = not null, UQ = unique, UN = unsigned, AI = auto increment. È ovviamente possibile specificare più di un attributo per ciascuna colonna. [↑](#footnote-ref-9)
10. PK = primary key, NN = not null, UQ = unique, UN = unsigned, AI = auto increment. È ovviamente possibile specificare più di un attributo per ciascuna colonna. [↑](#footnote-ref-10)
11. PK = primary key, NN = not null, UQ = unique, UN = unsigned, AI = auto increment. È ovviamente possibile specificare più di un attributo per ciascuna colonna. [↑](#footnote-ref-11)
12. PK = primary key, NN = not null, UQ = unique, UN = unsigned, AI = auto increment. È ovviamente possibile specificare più di un attributo per ciascuna colonna. [↑](#footnote-ref-12)
13. PK = primary key, NN = not null, UQ = unique, UN = unsigned, AI = auto increment. È ovviamente possibile specificare più di un attributo per ciascuna colonna. [↑](#footnote-ref-13)
14. PK = primary key, NN = not null, UQ = unique, UN = unsigned, AI = auto increment. È ovviamente possibile specificare più di un attributo per ciascuna colonna. [↑](#footnote-ref-14)
15. PK = primary key, NN = not null, UQ = unique, UN = unsigned, AI = auto increment. È ovviamente possibile specificare più di un attributo per ciascuna colonna. [↑](#footnote-ref-15)
16. PK = primary key, NN = not null, UQ = unique, UN = unsigned, AI = auto increment. È ovviamente possibile specificare più di un attributo per ciascuna colonna. [↑](#footnote-ref-16)
17. PK = primary key, NN = not null, UQ = unique, UN = unsigned, AI = auto increment. È ovviamente possibile specificare più di un attributo per ciascuna colonna. [↑](#footnote-ref-17)
18. IDX = index, UQ = unique, FT = full text, PR = primary. [↑](#footnote-ref-18)
19. IDX = index, UQ = unique, FT = full text, PR = primary. [↑](#footnote-ref-19)
20. IDX = index, UQ = unique, FT = full text, PR = primary. [↑](#footnote-ref-20)
21. IDX = index, UQ = unique, FT = full text, PR = primary. [↑](#footnote-ref-21)
22. IDX = index, UQ = unique, FT = full text, PR = primary. [↑](#footnote-ref-22)
23. IDX = index, UQ = unique, FT = full text, PR = primary. [↑](#footnote-ref-23)
24. IDX = index, UQ = unique, FT = full text, PR = primary. [↑](#footnote-ref-24)
25. IDX = index, UQ = unique, FT = full text, PR = primary. [↑](#footnote-ref-25)
26. IDX = index, UQ = unique, FT = full text, PR = primary. [↑](#footnote-ref-26)
27. IDX = index, UQ = unique, FT = full text, PR = primary. [↑](#footnote-ref-27)
28. IDX = index, UQ = unique, FT = full text, PR = primary. [↑](#footnote-ref-28)
29. IDX = index, UQ = unique, FT = full text, PR = primary. [↑](#footnote-ref-29)
30. IDX = index, UQ = unique, FT = full text, PR = primary. [↑](#footnote-ref-30)
31. IDX = index, UQ = unique, FT = full text, PR = primary. [↑](#footnote-ref-31)
32. IDX = index, UQ = unique, FT = full text, PR = primary. [↑](#footnote-ref-32)
33. IDX = index, UQ = unique, FT = full text, PR = primary. [↑](#footnote-ref-33)
34. IDX = index, UQ = unique, FT = full text, PR = primary. [↑](#footnote-ref-34)
35. IDX = index, UQ = unique, FT = full text, PR = primary. [↑](#footnote-ref-35)
36. IDX = index, UQ = unique, FT = full text, PR = primary. [↑](#footnote-ref-36)
37. IDX = index, UQ = unique, FT = full text, PR = primary. [↑](#footnote-ref-37)
38. IDX = index, UQ = unique, FT = full text, PR = primary. [↑](#footnote-ref-38)
39. IDX = index, UQ = unique, FT = full text, PR = primary. [↑](#footnote-ref-39)
40. IDX = index, UQ = unique, FT = full text, PR = primary. [↑](#footnote-ref-40)