Basi di Dati e Conoscenza

Progetto A.A. 2021/2022

Compra e Vendi

0244632

Manuela Reali

Indice

1. Descrizione del Minimondo

Bacheca elettronica di annunci

Si vuole realizzare un sistema informativo per la gestione di una bacheca elettronica di annunci. Tale bacheca permette agli utenti del sistema di inserire annunci per la vendita di materiale usato, di scambiare messaggi tra di loro (in maniera privata) per accordarsi sulla vendita/consegna dell'oggetto, o di inserire domande (in maniera pubblica) sull'oggetto.

Un utente del sistema si registra scegliendo un username univoco, inserendo tutte le sue informazioni anagrafiche, indicando un indirizzo di residenza ed eventualmente un indirizzo di fatturazione, un numero arbitrario di recapiti (telefono, cellulare, email) indicandone uno come mezzo di comunicazione preferito, ed inserendo i dati relativi alla sua carta di credito. I dati della carta di credito non sono obbligatori.

I gestori del servizio possono creare una gerarchia di categorie per gli annunci. Un utente, per creare un annuncio, seleziona una categoria e scrive una descrizione dell'oggetto. Eventualmente, può decidere di caricare una foto dell'oggetto. Per creare un annuncio, un utente deve necessariamente aver inserito i dati della sua carta di credito. Quando un oggetto inserito in bacheca è stato venduto, l'utente lo indica come tale e questo non viene più visualizzato nella bacheca pubblica.

Un utente del sistema, una volta letto e scelto un annuncio, può decidere di inserire un commento pubblico o di inviare un messaggio privato all'utente che ha inserito l'annuncio. Similmente, un utente può "seguire" uno degli annunci, venendo così informato ogni volta che su questo viene effettuata una modifica (ad esempio, viene inserita una nuova nota).

In generale, un utente può:
*Inserire/rimuovere nuovi annunci
* Modificare le sue informazioni anagrafiche
* Seguire annunci
* Mostrare gli annunci che sta seguendo visualizzando un'indicazione legata al fatto se uno degli annunci che sta seguendo è stato modificato (un oggetto segnato come venduto o rimosso compare comunque nell'elenco degli annunci seguiti dagli utenti, portando

l’indicazione del suo stato)
* Inviare messaggi agli altri utenti e mostrare lo storico delle sue conversazioni, anche con la possibilità di rispondere ad una conversazione specifica
* Inserire commenti agli annunci ancora attivi.

I gestori del servizio prendono una percentuale su ciascun oggetto indicato come venduto. Per questo motivo, essi possono generare un report indicante per ciascun utente del sistema quanti annunci sono stati contrassegnati come venduti. Il sistema calcola un percentuale pari al 3% della somma degli importi di tali oggetti, nel caso in cui la percentuale associata non sia già stata riscossa. Il report riporta anche le informazioni sulla carta di credito dell’utente, al fine di permettere la riscossione della percentuale.

2. Analisi dei Requisiti

Identificazione dei termini ambigui e correzioni possibili

Linea	Termine	Nuovo termine	Motivo correzione
4	Materiale usato	Oggetto	Si è preferito sostituire il termine "materiale usato" con "oggetto" poiché viene ripetutamente usato nella specifica ed è più generico.
5	Domande	Commenti	Il termine "Domande" è troppo specifico, si preferisce un termine utilizzato successivamente nella specifica.
7-23	Informazioni anagrafiche	Nome Cognome Data di nascita Codice fiscale Indirizzo di residenza	Si preferisce indicare che le "informazioni anagrafiche" riguardano tutto ciò che è registrato presso l'anagrafe comunale.
9-14	Dati relativi alla sua carta di credito	Codice carta di credito Data di scadenza Intestatario carta CVC/CVV	Si preferisce indicare quali siano effettivamente i "dati relativi alla sua carta di credito".
20	Nota	Commento	Il termine "nota", può essere considerato come "commento".

Specifica disambiguata

Bacheca elettronica di annunci

Si vuole realizzare un sistema informativo per la gestione di una bacheca elettronica di annunci. Tale bacheca permette agli utenti del sistema di inserire annunci per la vendita di oggetti, di scambiare messaggi tra di loro (in maniera privata) per accordarsi sulla vendita/consegna dell'oggetto, o di inserire commenti (in maniera pubblica) sull'oggetto.

Un utente del sistema si registra scegliendo un username univoco, inserendo: nome, cognome, data di nascita, codice fiscale, indirizzo di residenza, eventualmente un indirizzo di fatturazione e un numero arbitrario di recapiti (telefono, cellulare, email) indicandone uno come mezzo di comunicazione preferito. Inoltre un utente può inserire della sua carta di credito: intestatario, numero identificativo della carta, mese e anno di scadenza, CVC/CVV. I dati della carta di credito non sono

obbligatori.

I gestori del servizio possono creare una gerarchia di categorie per gli annunci. Un utente, per creare un annuncio, seleziona una categoria e scrive una descrizione dell'oggetto. Eventualmente, l'utente può decidere di caricare una foto dell'oggetto. Per creare un annuncio, un utente deve necessariamente aver inserito i dati della sua carta di credito. Quando un oggetto inserito in bacheca è stato venduto, l'utente lo indica come tale e questo non viene più visualizzato nella bacheca pubblica.

Un utente del sistema, una volta letto e scelto un annuncio, può decidere di inserire un commento pubblico o di inviare un messaggio privato all'utente che ha inserito l'annuncio. Similmente, un utente può "seguire" uno degli annunci, venendo così informato ogni volta che su questo viene effettuata una modifica (ad esempio, viene inserito un nuovo commento).

In generale, un utente può:

*Inserire/rimuovere nuovi annunci
* Modificare nome, cognome, data di nascita, codice fiscale, indirizzo di residenza
*Seguire annunci
* Mostrare gli annunci che sta seguendo e relativa indicazione di una loro eventuale modifica (un oggetto segnato come venduto o riscosso compare comunque nell'elenco degli annunci seguiti dagli utenti, portando l'indicazione del suo stato)

* Inviare messaggi agli altri utenti e mostrare lo storico delle sue conversazioni, anche con la possibilità di rispondere ad una conversazione specifica

* Inserire commenti agli annunci ancora attivi.

I gestori del servizio prendono una percentuale su ciascun oggetto indicato come venduto. Per questo motivo, i gestori del servizio possono generare un report indicante per ciascun utente del sistema quanti annunci sono stati contrassegnati come venduti. Il sistema calcola un percentuale pari al 3% della somma degli importi di tali oggetti, nel caso in cui la percentuale associata non sia già stata riscossa. Il report riporta anche le informazioni sulla carta di credito dell'utente, al fine di permettere la riscossione della percentuale.

Glossario dei Termini

Termine	Descrizione	Sinonimi	Collegamenti
Utente	Colui che utilizza il sistema, ma non può inserire gli annunci.		Annuncio, messaggio, commento.
Utente certificato	Colui che utilizza il sistema e può inserire gli annunci a seguito dell'inserimento dei dati della carta di credito.		Annuncio, messaggio, commento.
Gestore del servizio	Colui che gestisce il servizio.		Categoria, annuncio, report.
Annuncio	Oggetto che viene messo in vendita.	Oggetto	Utente, utente certificato, commento, messaggio, categoria, report.
Messaggio	Messaggio privato che un utente invia ad un altro per avere info sull'oggetto messo in vendita da quest' ultimo.	Messaggio privato	Utente, utente certificato.
Commento	Commento pubblico usato per poter chiedere info a qualsiasi utente sull'oggetto del relativo annuncio.	Commento pubblico	Utente, utente certificato, annuncio.
Categoria	Creata per poter suddividere gli annunci secondo un certo ordine.		Annuncio, gestore del servizio.
Report	Riporta informazioni riguardante gli oggetti venduti.		Gestore servizio, utente certificato, annuncio.

Raggruppamento dei requisiti in insiemi omogenei

Frasi di carattere generale
Si vuole realizzare un sistema informativo per la gestione di una bacheca elettronica di annunci. Tale bacheca permette agli utenti del sistema di inserire annunci per la vendita di oggetti, di scambiare messaggi tra di loro (in maniera privata) per accordarsi sulla vendita/consegna dell'oggetto, o di inserire commenti (in maniera pubblica) sull'oggetto.

Frasi relative a utenti
Un utente del sistema si registra scegliendo un username univoco, inserendo: nome, cognome, data di nascita, codice fiscale, indirizzo di residenza, eventualmente un indirizzo di fatturazione e un numero arbitrario di recapiti (telefono, cellulare, email) indicandone uno come mezzo di comunicazione preferito. Un utente del sistema, una volta letto e scelto un annuncio, può decidere di inserire un commento pubblico o di inviare un messaggio privato all'utente che ha inserito l'annuncio. Similmente, un utente può "seguire" uno degli annunci, venendo così informato ogni volta che su questo viene effettuata una modifica (ad esempio, viene inserito un nuovo commento). In generale, un utente può: * Modificare nome, cognome, data di nascita, codice fiscale, indirizzo di residenza * Seguire annunci * Mostrare gli annunci che sta seguendo e relativa indicazione di una loro eventuale modifica (un oggetto segnato come venduto o rimosso compare comunque nell'elenco degli annunci seguiti dagli utenti, portando l'indicazione del suo stato) * Inviare messaggi agli altri utenti e mostrare lo storico delle sue conversazioni, anche con la possibilità di rispondere ad una conversazione specifica * Inserire commenti agli annunci ancora attivi.

Frasi relative a utenti certificati
Per creare un annuncio, un utente deve necessariamente aver inserito i dati della sua carta di credito. Quando un oggetto inserito in bacheca è stato venduto, l'utente lo indica come tale e questo non viene più visualizzato nella bacheca pubblica. Un utente del sistema, una volta letto e scelto un annuncio, può decidere di inserire un commento pubblico o di inviare un messaggio privato all'utente che ha inserito l'annuncio. Similmente, un utente può "seguire" uno degli annunci, venendo così informato ogni volta che su questo viene effettuata una modifica (ad esempio, viene inserito un nuovo commento).

In generale, un utente può:
*Inserire/rimuovere nuovi annunci
* Modificare nome, cognome, data di nascita, codice fiscale, indirizzo di residenza
* Seguire annunci
* Mostrare gli annunci che sta seguendo e relativa indicazione di una loro eventuale modifica (un oggetto segnato come venduto o rimosso compare comunque nell'elenco degli annunci seguiti dagli utenti, portando l'indicazione del suo stato)

* Inviare messaggi agli altri utenti e mostrare lo storico delle sue conversazioni, anche con la possibilità di rispondere ad una conversazione specifica

* Inserire commenti agli annunci ancora attivi.

Frasi relative ad annunci

Un utente, per creare un annuncio, seleziona una categoria e scrive una descrizione dell'oggetto. Eventualmente, l'utente può decidere di caricare una foto dell'oggetto.

Quando un oggetto inserito in bacheca è stato venduto, l'utente lo indica come tale e questo non viene più visualizzato nella bacheca pubblica.

Mostrare gli annunci che sta seguendo e relativa indicazione di una loro eventuale modifica (un oggetto segnato come venduto o rimosso compare comunque nell'elenco degli annunci seguiti dagli utenti, portando l'indicazione del suo stato).

Frasi relative a gestori del servizio

I gestori del servizio possono creare una gerarchia di categorie per gli annunci.

I gestori del servizio prendono una percentuale su ciascun oggetto indicato come venduto. Per questo motivo, i gestori del servizio possono generare un report indicante per ciascun utente del sistema quanti annunci sono stati contrassegnati come venduti.

Il sistema calcola un percentuale pari al 3% della somma degli importi di tali oggetti, nel caso in cui la percentuale associata non sia già stata riscossa.

Frasi relative a messaggi

Un utente del sistema, una volta letto e scelto un annuncio, può decidere di inserire un commento pubblico o di inviare un messaggio privato all'utente che ha inserito l'annuncio.

Inviare messaggi agli altri utenti e mostrare lo storico delle sue conversazioni, anche con la possibilità di rispondere ad una conversazione specifica.

Frasi relative a commenti

Un utente del sistema, una volta letto e scelto un annuncio, può decidere di inserire un commento

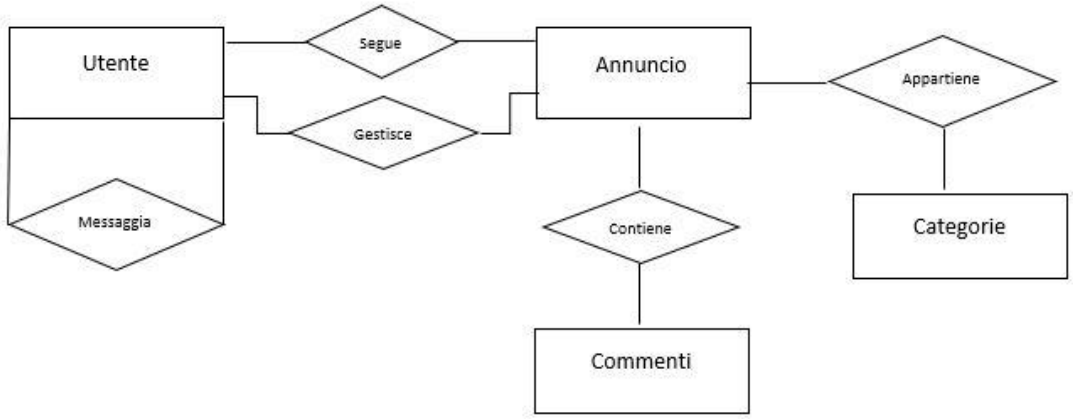
pubblico o di inviare un messaggio privato all'utente che ha inserito l'annuncio. Inserire commenti agli annunci ancora attivi.

Frasi relative a report
Per questo motivo, essi possono generare un report indicante per ciascun utente del sistema quanti annunci sono stati contrassegnati come venduti. Il report riporta anche le informazioni sulla carta di credito dell'utente, al fine di permettere la riscossione della percentuale.

3. Progettazione concettuale

Costruzione dello schema E-R

In seguito, viene proposto uno schema che rappresenta una panoramica generale:

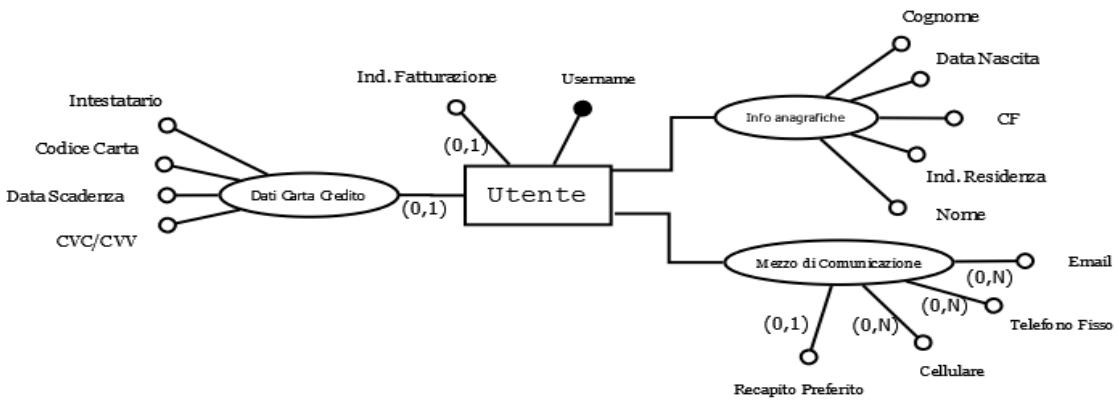


Sono state individuate 4 entità e 5 relazioni.
Le entità sono: *Utente, Annuncio, Commenti, Categorie.*
Le relazioni sono: *Messaggia, Contiene, Appartiene, Gestisce.*

Leggendo lo schema E-R da sinistra verso destra e collegando le entità alle rispettive relazioni si ha che:

- Gli *Utenti* hanno la possibilità di messaggiare con un altro utente e poter seguire e gestire un *Annuncio.*
- Un *Annuncio* può contenere dei commenti e appartenere ad una *Categoria ed* essere gestiti e seguiti dagli utenti.
- I *Commenti* possono essere contenuti negli *Annunci.*
- Ad una *Categoria* appartengono degli *Annunci*.

Raffinamento entità Utente



Nella rappresentazione dell'entità *Utente*, si ha:

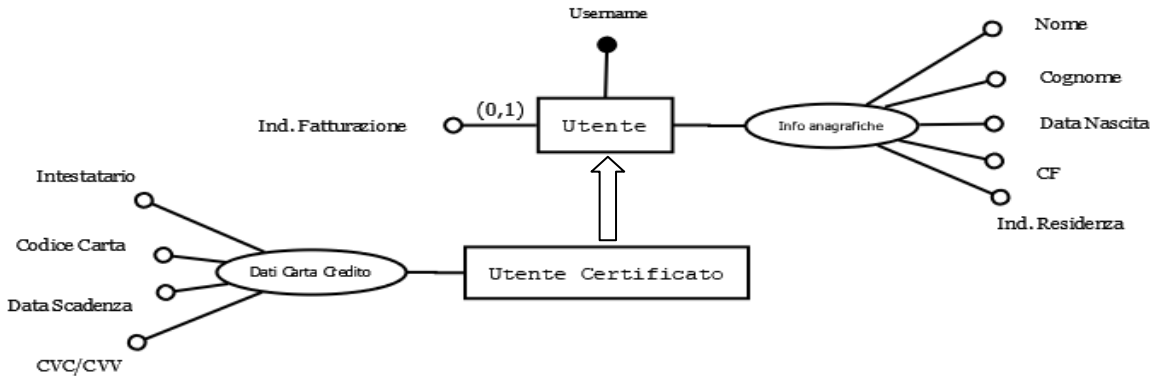
- l'attributo *Username* che identifica l'Utente;
- l'attributo opzionale *Indirizzo di fatturazione*;
- tre attributi composti ovvero *Informazioni Anagrafiche, Mezzo di Comunicazione, Dati Carta di Credito*.

In questa rappresentazione si può riscontrare:

1) Non si riesce a distinguere l'utente che può "creare" l'annuncio da quello che può "solamente seguirlo".
2) L'attributo composto *Mezzo di Comunicazione* è poco chiaro.

Soluzione 1

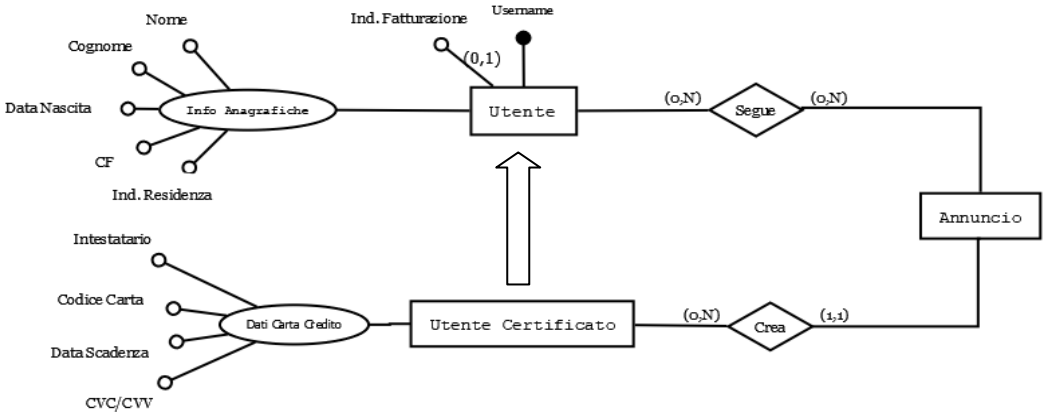
Evidenziando la seguente frase della specifica "Per creare un annuncio, un utente deve necessariamente aver inserito i dati della sua carta di credito", ci si rende conto che sarebbe ottimale proporre una specializzazione dell'entità *Utente*.
Come rappresentato nella figura sottostante, la nuova entità *Utente Certificato* (specializzazione di *Utente*), prenderà come attributo *Dati Carta Credito* il quale non sarà più opzionale, ma obbligatorio. L'obbligatorietà dell'attributo *Dati Carta Credito* consente di specificare che l'entità a cui è associato rappresenta l'utente a cui sarà concesso creare nuovi annunci.



A questo punto si possono creare due relazioni:

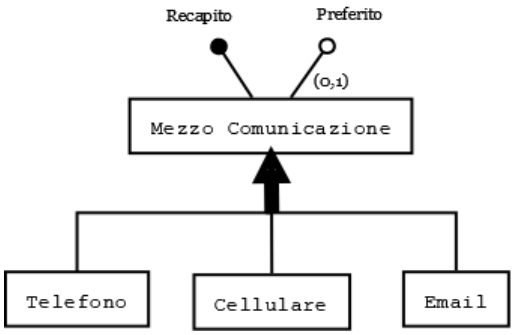
- Tra l'entità *Utente* e *Annuncio* c'è la relazione *Segue* di tipo molti a molti.
  Un utente può seguire 0 o N annunci e viceversa un annuncio può essere seguito da 0 o N utenti.

- Tra l'entità *Utente Certificato* e *Annuncio* c'è la relazione *Crea* di tipo uno a molti. Un determinato annuncio può essere creato da un solo utente mentre un utente può creare 0 o N annunci.



Soluzione 2

L'attributo *Mezzo di Comunicazione* ha un'importanza maggiore rispetto ad un semplice attributo composto.



Come si può vedere dall'immagine sovrastante, sfruttando la reificazione dell'attributo è stata creata innanzitutto l'entità *Mezzo Comunicazione* e conseguentemente si è applicata la generalizzazione.

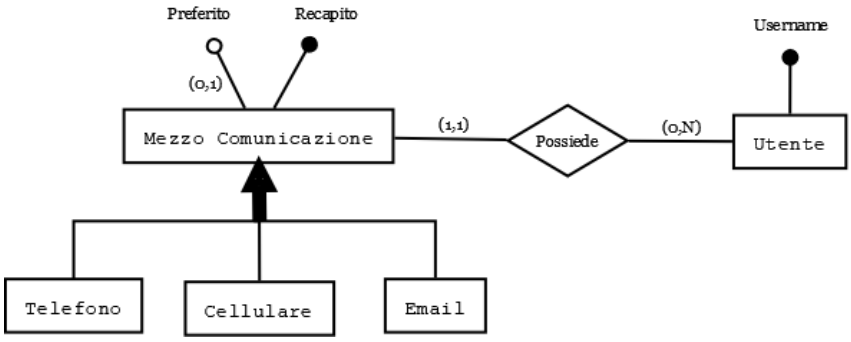
L'entità *Mezzo Comunicazione* in questo caso avrà tre diverse specializzazioni ovvero *Telefono* (fisso), *Cellulare*, *E-mail* che non avranno caratteristiche aggiuntive (altri attributi) rispetto al loro "padre", ma forniscono una visione più dettagliata e chiara di questa parte di schema.

Gli unici attributi dell'entità *Mezzo Comunicazione* sono:

- *Recapito:* attributo identificativo;
- *Preferito:* attributo opzionale che servirà a determinare se un recapito sia preferito o meno;

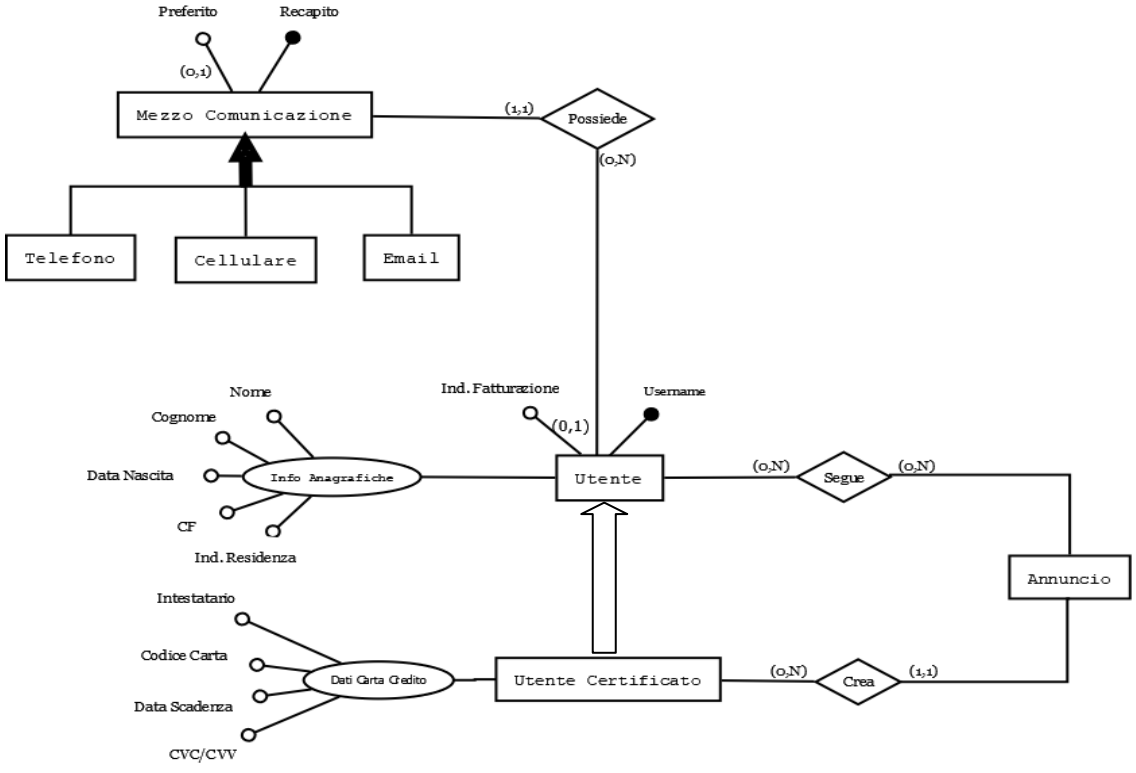
A questo punto, come si può vedere nella successiva parte di schema, si avrà:

- Tra l'entità *Mezzo Comunicazione* e *Utente* la relazione *Possiede* di tipo uno a molti. Un mezzo di comunicazione è posseduto da un solo utente mentre un utente può possedere 0 o N mezzi di comunicazione.

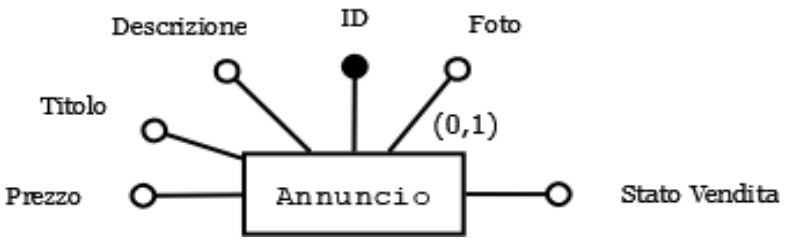


Schema conclusivo raffinamento entità Utente

Il seguente schema racchiude tutto ciò che è stato precedentemente spiegato.



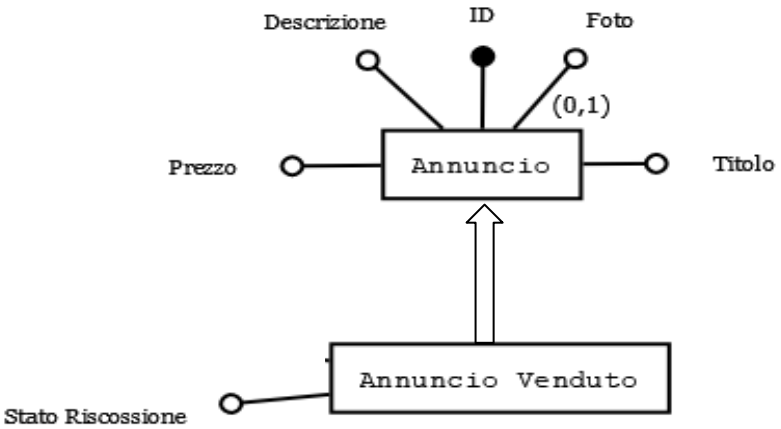
Raffinamento entità Annuncio



Nella rappresentazione dell'entità Annuncio sono stati introdotti sei attributi:

- *ID*: attributo identificatore progressivo;
- *Titolo*: utile per contraddistinguere al meglio l'oggetto messo in vendita;
- *Descrizione*: descrizione dell'oggetto messo in vendita;
- *Prezzo*: utile per fornire agli utenti una stima del prezzo dell'oggetto in vendita;
- *Stato Vendita*: per sapere se l'oggetto è stato venduto o meno;
- *Foto*: attributo opzionale;

Facendo affidamento all'immagine sovrastante, ci sono degli accorgimenti da fare.
Considerando in particolare le seguenti due affermazioni, riportate nella specifica, "Quando un oggetto inserito in bacheca è stato venduto, l'utente lo indica come tale" e "I gestori del servizio prendono una percentuale su ciascun oggetto indicato come venduto", viene naturale andare ad utilizzare per l'entità *Annuncio* una generalizzazione. In particolare, si può notare una evoluzione nel tempo del concetto di Annuncio.
Quindi, come riportato nell'immagine sottostante, si propone la nuova entità *Annuncio Venduto* che è "figlia" di *Annuncio*.

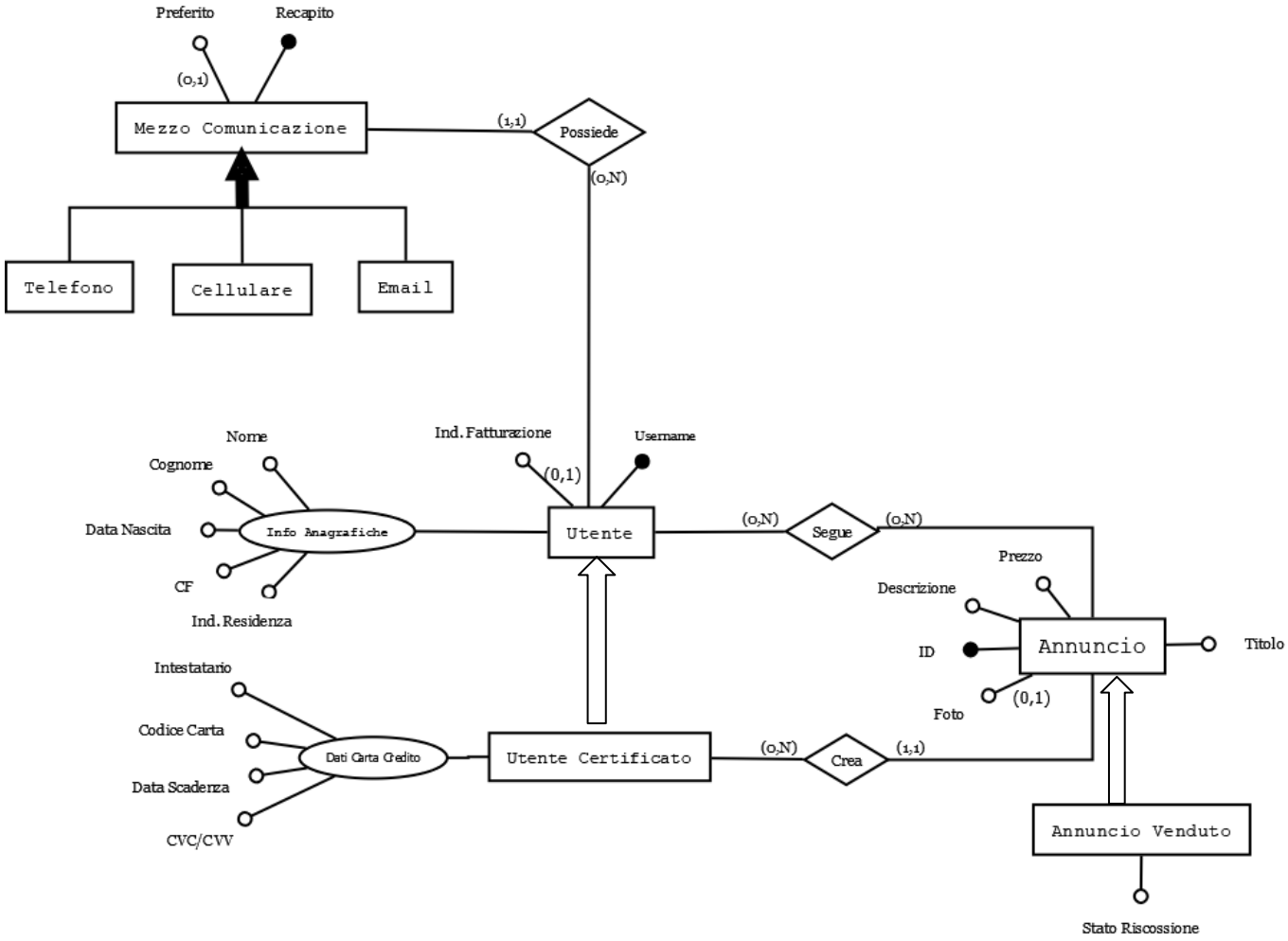


Per l'entità *Annuncio Venduto* si ha un solo attributo:

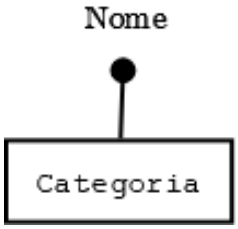
- *Stato Riscossione*: indica se i gestori di servizio hanno riscosso o meno la loro percentuale sull'oggetto venduto.

Inoltre, l'attributo *Stato Vendita* che apparteneva all'entità *Annuncio* è stato eliminato perché ritenuto ridondante dal momento in cui ora esiste proprio l'entità *Annuncio Venduto* che sottolinea che l'oggetto è stato venduto.

Schema conclusivo raffinamento entità Utente + Annuncio

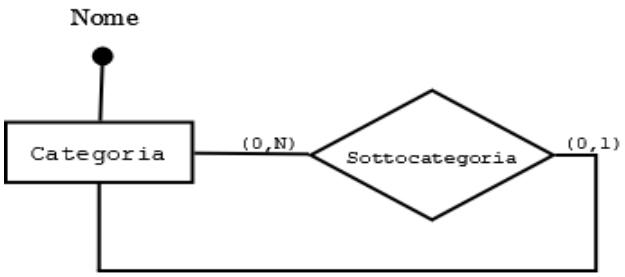


Raffinamento entità Categoria

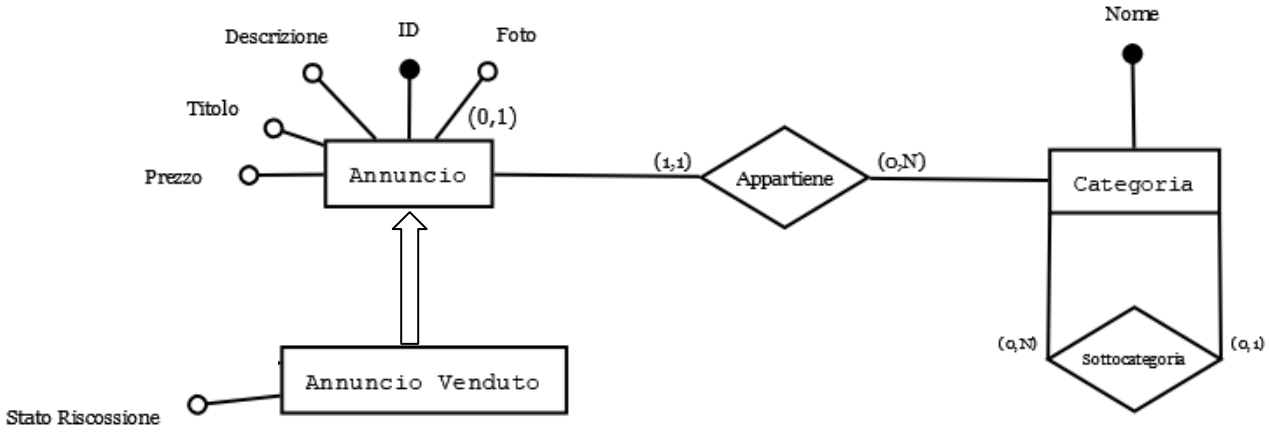


Ogni annuncio deve appartenere ad una categoria, per cui si costruisce l'entità *Categoria* e si pone come attributo univoco *Nome* in tal modo non possono esistere due o più categorie con lo stesso nome.

Inoltre come riportato nella specifica si è a conoscenza che "I gestori del servizio possono creare una gerarchia di categorie per gli annunci" per cui una buona rappresentazione di ciò è la sottostante.

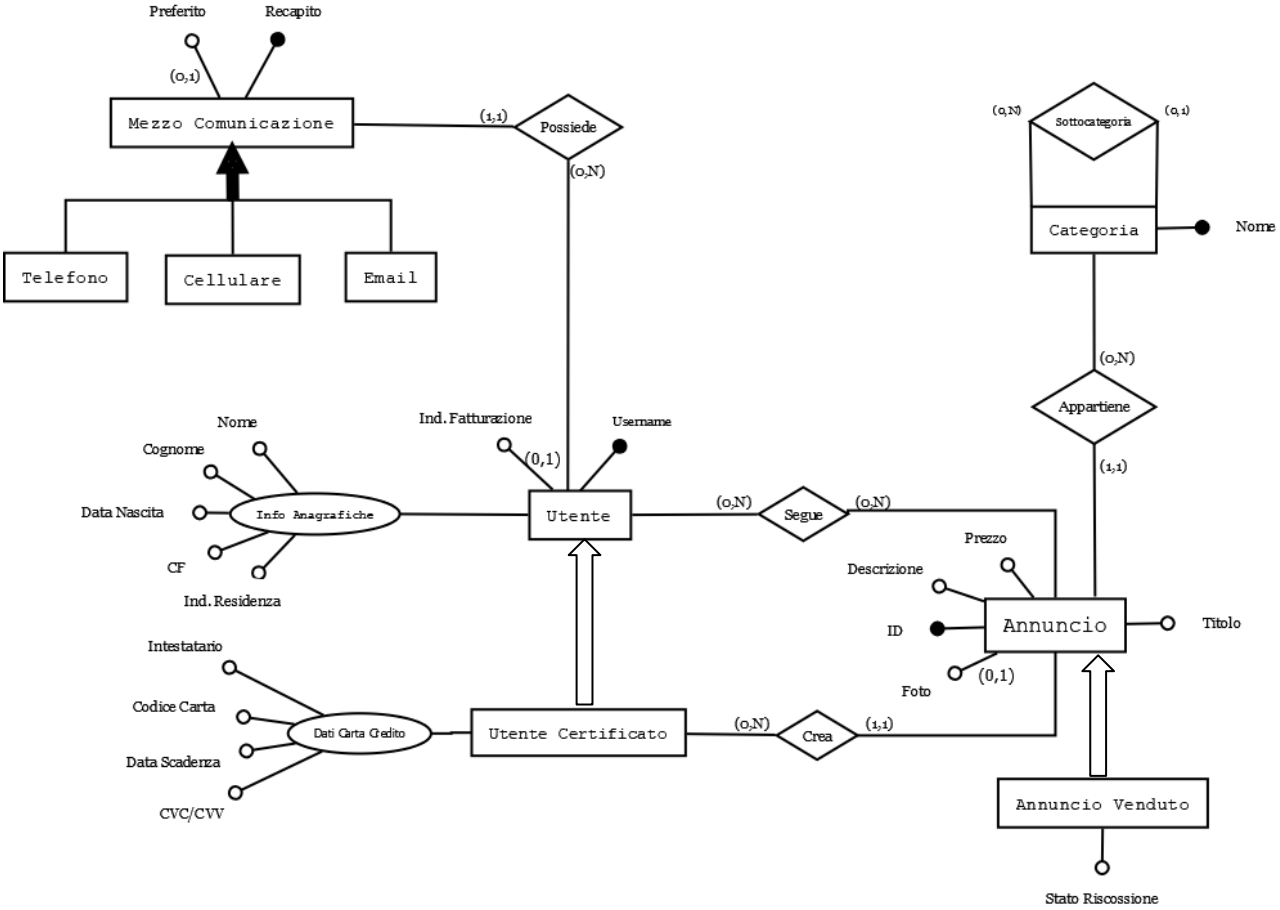


A questo punto, si può mettere in relazione l'entità *Annuncio* e *Categoria*.

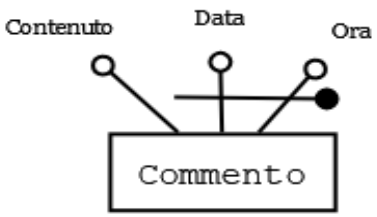


La relazione *Appartiene* di tipo uno a molti, descrive che ad ogni annuncio è associata una categoria e ad ogni categoria possono appartenere 0 o N annunci.

Schema conclusivo raffinamento entità Utente + Annuncio + Categoria



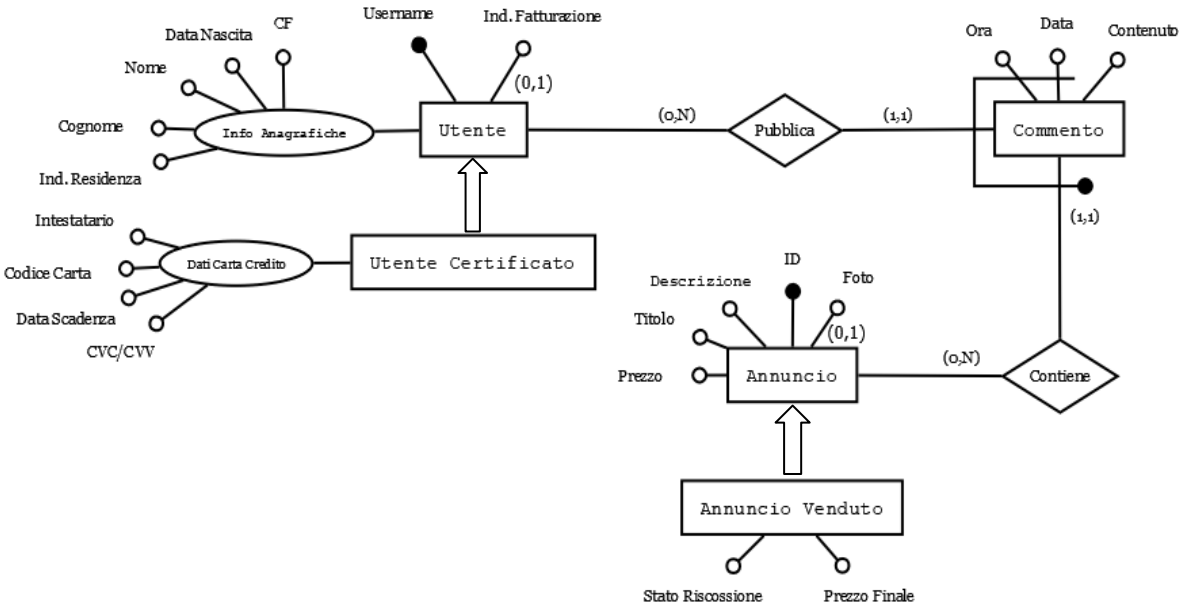
Raffinamento entità Commento



In questa prima fase del raffinamento dell'entità *Commento* si hanno tre attributi:

- *Ora*: ora in cui è stato scritto il commento;
- *Data*: data in cui è stato scritto il commento;
- *Contenuto*: ciò che viene scritto nel commento;

Prendendo l'entità *Commento* come concetto a sestante, *Data* e *Ora* vanno a essere i suoi identificativi. Ora pensando di inserire l'entità nel contesto del mini-mondo, si avrà il seguente schema:



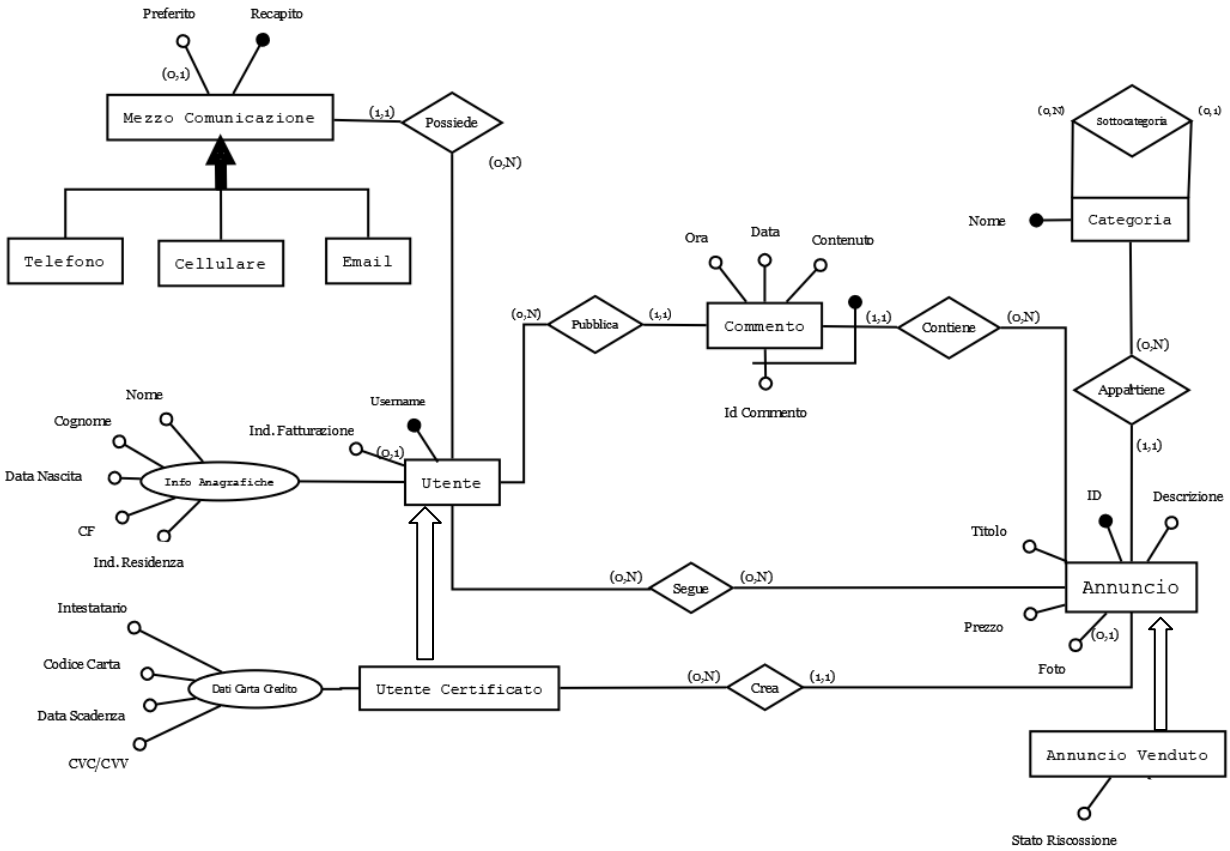
Sono state introdotte due relazioni:

- Tra l'entità *Commento* e *Utente* c'è la relazione *Pubblica* di tipo uno a molti. Un *Utente* può pubblicare 0 o N commenti, mentre uno e un solo *Commento* la volta può essere pubblicato da un *Utente;*
- Tra l'entità *Commento* e *Annuncio* c'è la relazione *Contiene* di tipo uno a molti. Un *Annuncio* può contenere 0 o N commenti, mentre un determinato *Commento* deve essere necessariamente contenuto in un solo *Annuncio*.

Si può notare che l'entità *Commento* ora è identificata da *Ora* + *Data* + *ID* + *Username* poiché *Ora* + *Data* in questo contesto non sono più sufficienti ad essere degli identificatori dell'entità *Commento*.

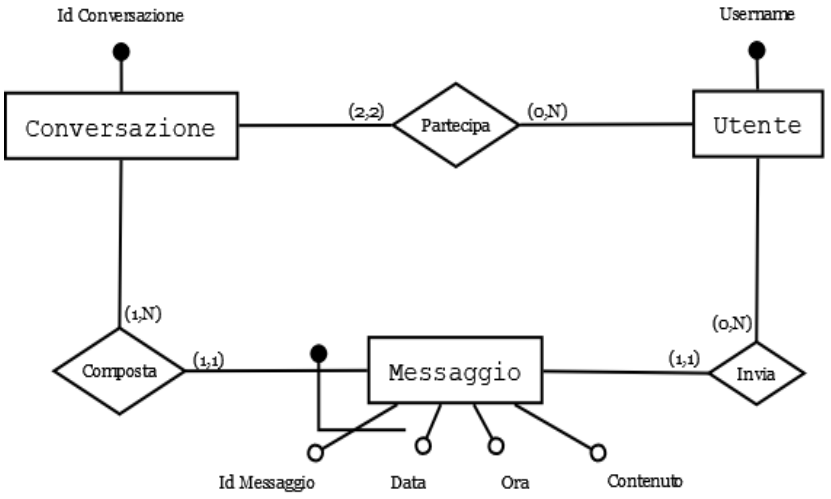
Tutto ciò però, rende lo schema di più complessa lettura, per cui è stato deciso di introdurre un identificatore progressivo per l'entità *Commento* e identificarla "solamente" attraverso *Id Commento* e *ID* (Annuncio).

Schema conclusivo raffinamento entità Utente + Annuncio + Categoria + Commento



Raffinamento entità Messaggio

Per quanto riguarda l'entità *Messaggio*, andando a tener conto di questa frase "Inviare messaggi agli altri utenti e mostrare lo storico delle sue conversazioni, anche con la possibilità di rispondere ad una conversazione specifica", ci si rende conto che i messaggi scambiati tra due utenti sono parte di una conversazione tra di essi.



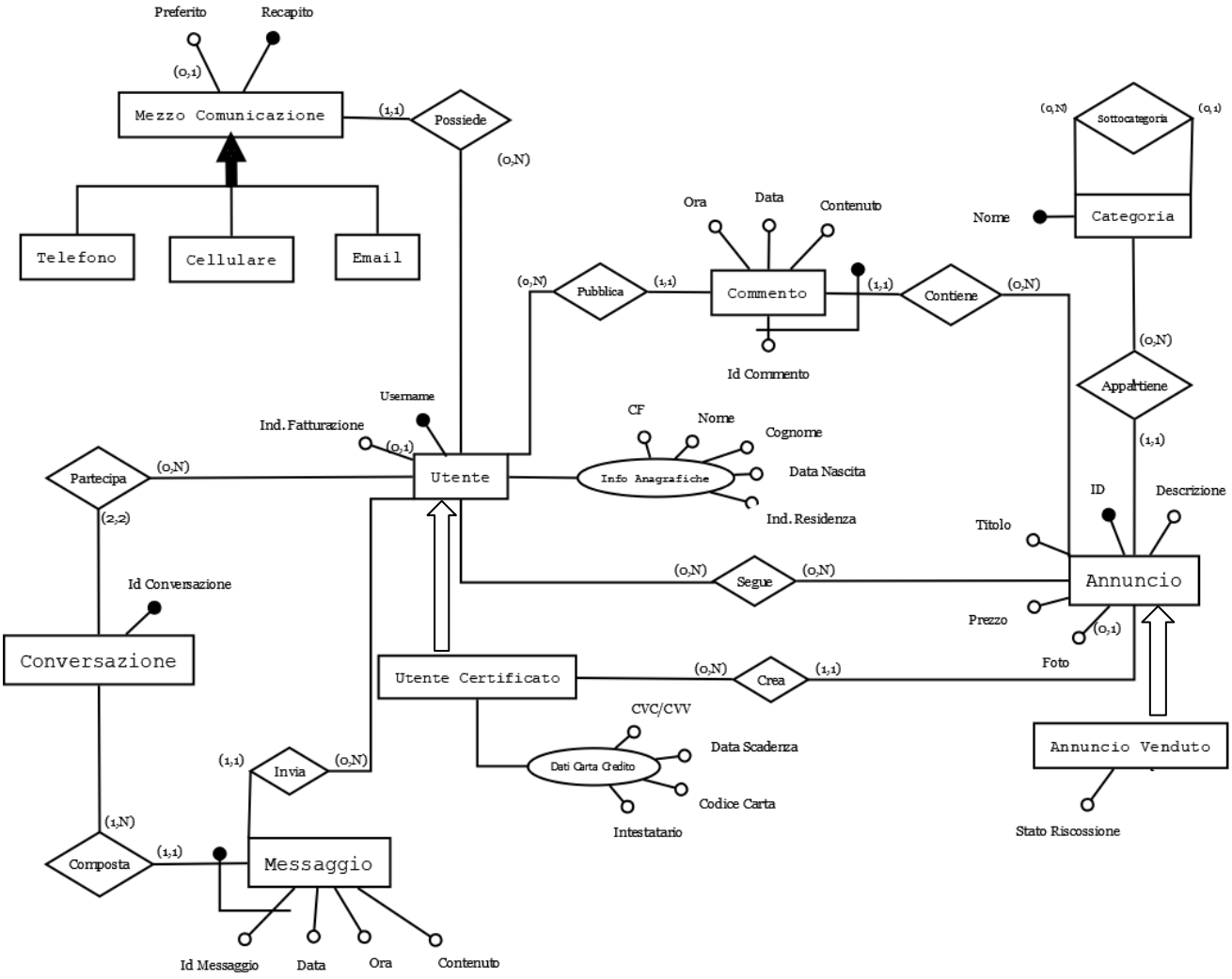
L'entità *Conversazione*, identificata dall'attributo *Id Conversazione* è messa in relazione con le entità *Utente* e *Messaggio*.
Per mettere in relazione queste tre entità si avrà:

- Tra *Conversazione* e *Utente* c'è la relazione *Partecipa* in cui si va a dire che ad ogni conversazione come minimo e massimo partecipano due utenti, mentre 0 o N utenti possono partecipare a delle conversazioni;
- Tra *Conversazione* e *Messaggio* c'è la relazione *Composta* di tipo uno a molti in cui si ha che una conversazione è composta da 1 a N messaggi, mentre uno e un solo messaggio con un determinato id è associato a una determinata conversazione.

Per quanto riguarda l'entità *Messaggio* si avrà che i suoi identificativi sono *Id Messaggio* e *Id Conversazione*. Inoltre è presente una relazione:

- Tra *Messaggio* e *Utente* c'è la relazione *Invia* di tipo uno a molti in cui si ha che un utente può inviare 0 o N messaggi, ma uno e un solo messaggio la volta può essere inviato ad un utente.

Integrazione finale



Regole aziendali

Due utenti non devono avviare più di una conversazione tra di loro.

Un utente non deve avere una conversazione con se stesso.

Una conversazione deve avvenire tra due utenti di cui almeno uno ha un annuncio pubblicato.

Un utente deve esprimere una sola preferenza tra i mezzi di comunicazione posseduti.

Una categoria non deve essere sottocategoria di se stessa.

Un utente non deve seguire un annuncio venduto.

Un utente non deve commentare un annuncio venduto.

I gestori di servizio devono prendere una percentuale sugli oggetti venduti pari al 3% della somma di tali oggetti.

Dizionario dei dati

Entità	Descrizione	Attributi	Identificatori
Utente	Colui che utilizza il sistema	Username, Nome, Cognome, Cf, Data di Nascita, Ind. Di Residenza, Ind. di Fatturazione	Username
Utente Certificato	Specializzazione di Utente, è colui che oltre ad utilizzare il sistema avendo inserito i dati della carta di credito può creare gli annunci.	Username, Nome, Cognome, Cf, Data di Nascita, Ind. Di Residenza, Ind. di Fatturazione, Codice Carta, Intestatario, Scadenza, CVC/ CVV	Username
Annuncio	Oggetto messo in vendita	ID, Descrizione, Foto, Titolo, Prezzo	ID
Annuncio Venduto	Specializzazione di Annuncio. Oggetto messo in vendita	Stato di Riscossione	ID
Categoria	Le diverse categorie in cui possono essere divisi gli annunci	Nome	Nome
Commento	Commento pubblico scritto da parte di un utente in un annuncio	ID Commento, Ora, Data, Contenuto	ID, ID Commento
Mezzo di Comunicazione	Rappresenta i diversi mezzi di comunicazione per poter contattare l'utente che ha pubblicato l'annuncio	Preferito, Recapito	Recapito
Telefono	Specializzazione di Mezzo di Comunicazione		Recapito
Cellulare	Specializzazione di Mezzo di Comunicazione		Recapito
Email	Specializzazione di Mezzo di		Recapito

	Comunicazione		
Conversazione	Insieme di messaggi scambiati tra due utenti	Id Conversazione	Id Conversazione
Messaggio	Messaggio inviato da un utente ad un altro e facente parte di una determinata conversazione	Id Messaggio, Data, Ora, Contenuto	Id Messaggio, Id Conversazione

4. Progettazione logica

Volume dei dati

Concetto nello schema	Tipo[1]	Volume atteso
Utente	E	60000
Utente Certificato	E	24000
Partecipa	R	24000
Conversazione	E	12000
Composta	R	120000
Messaggio	E	120000
Invia	R	120000
Crea	R	6000
Annuncio	E	6000
Segue	R	120000
Annuncio Venduto	E	4200
Appartiene	R	6000
Categoria	E	90
Sottocategoria	R	75
Contiene	R	30000
Commento	E	30000
Pubblica	R	30000
Possiede	R	60000
Mezzo Comunicazione	E	60000
Email	E	27000
Cellulare	E	27000
Telefono	E	6000

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

Tavola delle operazioni

Cod.	Descrizione	Frequenza attesa
OP1	Aggiorna contatto preferito	100/anno
OP2	Aggiorna info anagrafiche	50/anno
OP3	Certifica utente	200/gg
OP4	Crea annuncio	200/gg
OP5	Crea categoria	3/anno
OP6	Crea commento	50/gg
OP7	Crea contatto	100/mese
OP8	Crea utente	200/mese
OP9	Elimina annuncio	200/gg
OP10	Elimina contatto	100/mese
OP11	Avvia conversazione	1000/mese
OP12	Visualizza annunci posseduti	200/gg
OP13	Riscuoti percentuale	20/gg
OP14	Segna oggetto come venduto	140/gg
OP15	Segui annuncio	500/gg
OP16	Visualizza annunci seguiti	100/gg
OP17	Visualizza bacheca pubblica	1000/gg
OP18	Visualizza categorie	200/gg
OP19	Visualizza conversazioni	200/gg
OP20	Visualizza intera chat	300/gg
OP21	Visualizza commenti	50/gg
OP22	Visualizza contatti	200/gg
OP23	Visualizza info anagrafiche	100/mese
OP24	Rispondi a conversazione	280/gg
OP25	Genera report per utente	1/gg

Costo delle operazioni

OP1: Aggiorna contatto preferito

Concetto	Costrutto	Accessi	Tipologia
Utente	E	1	L
Possiede	R	1	L
Mezzo di Comunicazione	E	1	L
Mezzo di comunicazione	E	1	S

OP1: 100 * (1+1+1+2) = 500

OP2: Aggiorna info anagrafiche

Concetto	Costrutto	Accessi	Tipologia
Utente	E	1	S

OP2: 50 * (2) = 100

OP3: Certifica utente

Concetto	Costrutto	Accessi	Tipologia
Utente certificato	E	1	S

OP3: 200 * (2) = 400

OP4: Crea annuncio

Concetto	Costrutto	Accessi	Tipologia
Crea	R	1	S
Annuncio	E	1	S

OP4: 200 * (2+2) = 800

OP5: Crea categoria

Concetto	Costrutto	Accessi	Tipologia

Categoria	E	1	S
Sottocategoria	R	1	S

OP5: 3 * (2+2) = 12

OP6: Crea commento

Concetto	**Costrutto**	**Accessi**	**Tipologia**
Pubblica	R	1	S
Commento	E	1	S

OP6: 50 * (2+2) = 200

OP7: Crea contatto

Concetto	**Costrutto**	**Accessi**	**Tipologia**
Mezzo di comunicazione	E	1	S

OP7: 100 * (2) = 200

OP8: Crea utente

Concetto	**Costrutto**	**Accessi**	**Tipologia**
Utente	E	1	S

OP8: 200 * (2) = 400

OP9: Elimina annuncio

Concetto	**Costrutto**	**Accessi**	**Tipologia**
Annuncio	E	1	S

OP9 = 200 * (2) = 400

OP10: Elimina contatto

Concetto	**Costrutto**	**Accessi**	**Tipologia**
Mezzo comunicazione	E	1	S

OP10: 100 * (2) = 200

OP11: Avvia conversazione

Concetto	Costrutto	Accessi	Tipologia
Invia	R	1	S
Messaggio	E	1	S
Composta	R	1	S
Conversazione	E	1	S
Partecipa	R	1	S

OP11: 1000 * (2+2+2+2+2) = 10000

OP12: Visualizza annunci posseduti

Concetto	Costrutto	Accessi	Tipologia
Utente Certificato	E	1	L
Crea	R	1	L
Annuncio	E	1	L

OP12: 200 * (1+1+1) = 600

OP13: Riscuoti percentuale

Concetto	Costrutto	Accessi	Tipologia
Annuncio Venduto	E	1	S

OP13: 20 * (2) = 40

OP14: Segna oggetto come venduto

Concetto	Costrutto	Accessi	Tipologia
Annuncio Venduto	E	1	S

OP14: 140 * (2) = 280

OP15: Seguire annuncio

Concetto	Costrutto	Accessi	Tipologia
Segue	R	1	S

OP15: 500 * (2) = 1000

OP16: Visualizza annunci seguiti

Concetto	Costrutto	Accessi	Tipologia
Utente	E	1	L
Segue	R	2	L
Annuncio	E	2	L

OP16: 100 * (1+ 2 + 2) = 500

OP17: Visualizza bacheca pubblica

Concetto	Costrutto	Accessi	Tipologia
Annuncio	E	1	L

OP17: 1000 * 1 = 1000

OP18: Visualizza categorie

Concetto	Costrutto	Accessi	Tipologia
Categoria	E	1	L

OP18: 200 * 1 = 200

OP19: Visualizza conversazioni

Concetto	Costrutto	Accessi	Tipologia
Conversazioni	E	1	L

OP19: 200 * 1 = 200

OP20: Visualizza intera chat

Concetto	Costrutto	Accessi	Tipologia
Conversazione	E	1	L
Composta	R	10	L
Messaggio	E	10	L

OP20: 300 * (1+10+10) = 6300

OP21: Visualizza commenti

Concetto	Costrutto	Accessi	Tipologia
Commento	E	1	L

OP21: 50 * 1 = 50

OP22: Visualizza contatti

Concetto	Costrutto	Accessi	Tipologia
Mezzo comunicazione	E	1	L

OP22: 200 * 1 = 200

OP23: Visualizza info anagrafiche

Concetto	Costrutto	Accessi	Tipologia
Utente	E	1	L

OP23: 100 * 1= 100

OP24: Rispondi a conversazione

Concetto	Costrutto	Accessi	Tipologia
Invia	E	1	S
Messaggio	E	1	S
Composta	R	1	S

OP24: 280 * (2+2+2) = 1680

OP25: Genera report

Concetto	Costrutto	Accessi	Tipologia
Annuncio venduto	E	1	L

OP25: 1 * 1= 1

Ristrutturazione dello schema E-R

1) **Analisi delle ridondanze**

   Non sono presenti ridondanze che ottimizzano lo schema.

2) **Eliminazione delle generalizzazioni**

   Nello schema sono presenti 3 generalizzazioni:

   - **Generalizzazione Utente**

     In questo caso avendo una generalizzazione parziale e delle operazioni che si riferiscono solo a occorrenze di Utente o Utente Autenticato, conviene sostituire la generalizzazione con un'associazione.

   - **Generalizzazione Mezzo di Comunicazione**

     In questo caso si va ad accorpare le entità figlie della generalizzazione nel genitore. Si è presa questa decisione perché i "figli" dell'entità Mezzo di Comunicazione, non hanno degli attributi propri e non sono legati con relazioni ad altre entità.

   - **Generalizzazione Annuncio**

     In questo caso si hanno due opportunità o accorpare l'entità figlia della generalizzazione nel padre o sostituire la generalizzazione con un'associazione. Vedendo nella tabella delle operazioni, si può osservare che ci sono operazioni che si riferiscono o solo ad Annuncio o ad Annuncio Venduto.

     Andando ad analizzare:

     1) Accorpando l'entità figlia nel "padre" si avrebbero un numero minore di accessi a discapito di uno spreco di memoria, anche se minima, per la presenza di valori nulli.

     2) Sostituendo la generalizzazione con un'associazione, si ha un risparmio di memoria, ma un numero di accessi superiori.

     Tenendo conto di queste considerazioni, si è preferito inglobare l'entità *Annuncio Venduto* nel padre poiché lo spreco di memoria sarebbe davvero minimo, mentre il

numero di accessi che bisognerebbe fare sostituendo la generalizzazione con un'associazione, sarebbero molto più problematici.

Questo a discapito di operazioni come genera report e riscuoti percentuale che vengono però eseguite meno spesso rispetto ad altre operazioni riguardanti gli annunci.

Si avrà quindi, l'entità *Annuncio* che oltre ad avere i suoi attributi, ne avrà altri due:

1) *Stato Vendita:* indica se l'oggetto è stato venduto o meno;
2) *Stato Riscossione:* attributo appartenente ad *Annuncio Venduto*

3) **Scelta degli identificatori primari**

Utente: Username

Utente Certificato: Username

Annuncio: Id_Annuncio

Categoria: Nome

Mezzo di Comunicazione: Recapito

Messaggio: Id_Messaggio, Id_Conversazione

Conversazione: Id_Conversazione

Commento: Id_Commento, Id_Annuncio

Trasformazione di attributi e identificatori

Le trasformazioni che vengono realizzate sugli attributi:

- creare attributi singoli sia per *Informazioni Anagrafiche* che per *Dati Carta di Credito.*

Traduzione di entità e associazioni

Utente (Username, Nome, Cognome, CodiceFiscale, DataNascita, IndirizzoResidenza, IndirizzoFatturazione)

Utente Certificato (Username, CodiceCarta, IntestatarioCarta, CVC/CVV, DataScadenza)

Vincoli di integrità referenziale:

- Utente Certificato (Username) -> Utente (Username)

Mezzo di Comunicazione (Recapito, Preferito, Tipologia, Utente)

Vincoli di integrità referenziale:

- Mezzo di Comunicazione (Utente) -> Utente(Username)

Annuncio (IdAnnuncio, Titolo, Descrizione, Categoria, Prezzo, Foto, StatoVendita, Stato Riscossione, UtenteCertificato)

Vincoli di integrità referenziale:

- Annuncio (UtenteCertificato) -> UtenteCertificato(Username)
- Annuncio (Categoria) -> Categoria(Nome)

Categoria (Nome)

Sottocategoria (Figlia, Genitore)

Vincoli di integrità referenziale:

- Sottocategoria (Figlia) -> Categoria(Nome)
- Sottocategoria (Genitore) -> Categoria(Nome)

Commento (IdCommento, IdAnnuncio, Utente, Contenuto, Data, Ora)

Vincoli di integrità referenziale:

- Commento (IdAnnuncio) -> Annuncio (IdAnnuncio)
- Commento (Utente) -> Utente(Username)

Messaggio (IdMessaggio, IdConversazione, Contenuto, Data, Ora, UtenteMittente)

Vincoli di integrità referenziale:

- Messaggio (IdConversazione) -> Conversazione (IdConversazione)
- Messaggio (UtenteMittente) -> Utente(Username)

Conversazione (IdConversazione, Utente1, Utente2)

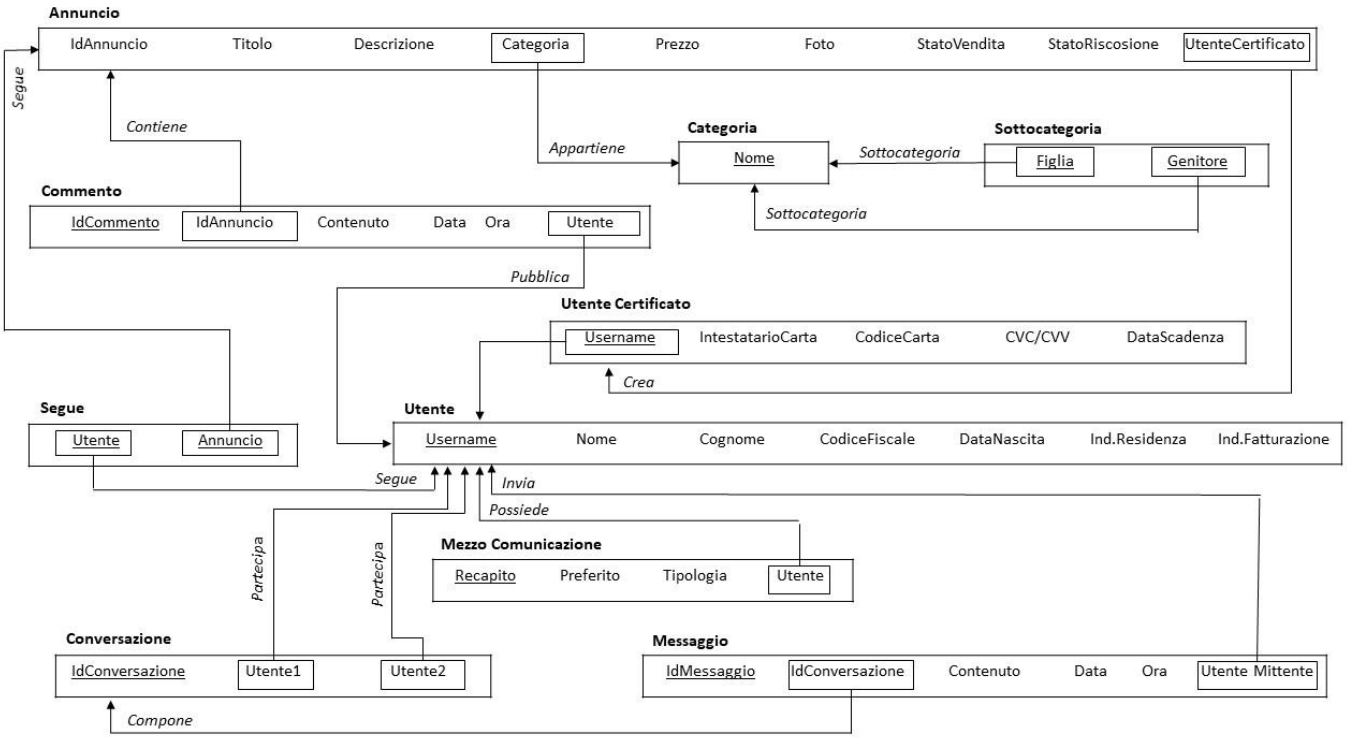
Vincoli di integrità referenziale:

- Conversazione (Utente1) -> Utente(Username)
- Conversazione (Utente2) -> Utente(Username)

Segue (Utente, Annuncio)

Vincoli di integrità referenziale:

- Segue (Utente) -> Utente (Username)
- Segue(Annuncio) -> Annuncio (IdAnnuncio)



Normalizzazione del modello relazionale

Verifichiamo che le relazioni ideate rispettino le 3 forme normali:

- **Prima forma normale:**
  1. È presente una chiave primaria (tuple non duplicate)
  2. Non vi sono gruppi di attributi che si ripetono
  3. Colonne indivisibili (attributi non composti)

  Prima forma normale soddisfatta per ogni relazione.

- **Seconda forma normale:**
  1. Nessuna dipendenza parziale (i valori dipendono dalle chiavi minime)

  Seconda forma normale soddisfatta in ogni relazione.

- **Terza forma normale:**
  1. Non vi sono dipendenze transitive tra le relazioni

  Terza forma normale soddisfatta in ogni relazione.

5. Progettazione fisica

Utenti e privilegi

Per aumentare la sicurezza del database ogni utente ha accesso solo all'esecuzione delle procedures di cui hanno bisogno.

Login

L'utente di Login è un utente non registrato nel sistema.
Si ha accesso all'esecuzione delle seguenti procedures:

- "login": legge dalle tabelle *user* e *utente*.
- "creaUtente": scrive sulla tabella *user* e *utente* per registrare un nuovo utente nel database
- "creaGestore": scrive sulla tabella *user* per registrare un nuovo gestore di servizio

Accesso in lettura e scrittura alle tabelle:

- user
- utente

Utente

Utente registrato nel sistema che non può pubblicare annunci.
Si ha accesso all'esecuzione delle seguenti procedures:

- "visualizzaContatti": legge dalla tabella *mezzocomunicazione* per ottenere la lista dei contatti posseduti;
- "aggiornaContattoPreferito": legge e scrive sulla tabella *mezzocomunicazione* per poter aggiornare il recapito preferito;
- "aggiornaInfoAnagrafiche": scrive sulla tabella *utente* per aggiornare le informazioni anagrafiche;
- "certificaUtente": scrive sulla tabella *utenteCertificato* per certificare un utente;
- "creaCommento": scrive sulla tabella *commento* per pubblicare un nuovo commento relativo ad un annuncio;
- "creaContatto": scrive sulla tabella *mezzocomunicazione* per inserire un nuovo recapito;
- "eliminaContatto": legge sulla tabella *mezzocomunicazione* il recapito per poi poterlo eliminare se esiste;

- “inviaMessaggio”: legge ed eventualmente scrive sulla tabella *conversazione* per creare una nuova conversazione se già non esiste. Scrive sulla tabella *messaggio* per inviare il messaggio;
- “segueAnnuncio”: scrive sulla tabella *segue* per inserire un utente che vuole seguire un annuncio;
- “visualizzaAnnunciSeguiti”: legge dalla tabella *annuncio* e *segue* per visualizzare tutti gli annunci che un utente segue;
- “visualizzaBachecaPubblica”: legge dalla view *bachecaPubblica* per poter visualizzare tutti gli annunci che non sono stati ancora venduti;
- “visualizzaChatInCorso”: legge dalle tabelle *conversazione* e *messaggio* per poter visualizzare le conversazioni che un utente ha in attivo e l’ultimo messaggio di ognuna di ognuna di esse;
- “visualizzaChatIntera”: legge dalle tabelle *conversazione* e *messaggio* per poter visualizzare tutti i messaggi di una determinata conversazione;
- “visualizzaCommenti”: legge dalla tabella *commento* per poter visualizzare tutti i commenti;
- “visualizzaInfoAnagrafiche”: legge dalla tabella *utente* per poter visualizzare tutte le informazioni anagrafiche;
- “visualizzaContattiVenditoreSeguiti”: legge dalle tabelle *segue, annuncio* e *mezzocomunicazione* per poter visualizzare i contatti del venditore di un annuncio seguito dall’utente loggato;
- “visualizzaCommentiSeguiti”: legge dalle tabelle annuncio, segue e commento per poter visualizzare i commenti degli annunci seguiti dall’utente loggato;
- “creaCommentoSeguiti”: legge dalle tabelle *annuncio* e *segue* per verificare se l’utente loggato segua un determinato annuncio per poi effettuare una scrittura su *commento* per inserire un commento al rispettivo annuncio seguito.
- “visualizzaContattiVenditore”: legge dalle tabelle *mezzocomunicazione* e *annuncio* per poter visualizzare i recapiti degli utenti che hanno annunci pubblicati non venduti.
- “inviaMessaggioSegui”: legge sulle tabelle *annuncio* e *segui* per verificare se l’utente loggato segua un determinato annuncio, per poi leggere ed eventualmente scrivere su *conversazione* per avviare una conversazione, se già non esiste, con il venditore che ha pubblicato l’annuncio a cui si è interessati. Infine, si scrive su *messaggio* per inviare il messaggio.
- “rispondiConversazione”: legge sulla tabella *conversazione* e scrive sul *messaggio* per inviare il messaggio.

Accesso in lettura e scrittura per le tabelle:

- mezzocomunicazione
- commento
- utente
- conversazione
- messaggio
- segue

Accesso in scrittura per le tabbelle:

- utenteCertificato

Accesso in lettura per le tabelle:

- annuncio

Utente Certificato

Utente registrato nel sistema che può pubblicare annunci.
Oltre ad avere accesso all' esecuzione delle procedures dell'*Utente,* ha accesso:

- "creaAnnuncio": legge la tabella *categoria* e scrive su *annuncio* per poter creare un annuncio specificandone l'appartenenza ad una determinata categoria;
- "eliminaAnnuncio": legge sulla tabella *annuncio* per poter effettuare l'eliminazione;
- "visualizzaMieiAnnunci": legge sulla tabella *annuncio* per poter visualizzare tutti gli annunci pubblicati e non venduti dall'utente loggato;
- "segnaVenduto": legge e scrive sulla tabella *annuncio* per poter segnare un oggetto come venduto;
- "visualizzaCategoria": legge dalle tabelle *categoria* e *sottocategoria* per poter visualizzare quali sono le categorie presenti e i rispettivi 'padri';
- "visualizzaAnnunci": legge dalla tabella *annuncio* per poter visualizzare tutti gli annunci pubblicati dall'utente loggato;

Accesso in lettura/scrittura per le tabelle:

- annuncio
- utente

- utenteCertificato
- segue
- commento
- conversazione
- messaggio
- mezzocomunicazione

Accesso in lettura per le tabelle:

- Categoria
- Sottocategoria

Gestore

Gestore del sistema.
Si ha accesso all'esecuzione delle seguenti procedures:

- "riscuotiPercentuale": scrive sulla tabella *annuncio* per poter riscuotere la percentuale di un oggetto venduto;
- "creaCategoria": scrive sulle tabelle *categoria* e *sottocategoria* per poter inserire una categoria ed eventualmente una categoria "figlia";
- "visualizzaCategoria": legge dalle tabelle *categoria* e *sottocategoria* per sapere quali sono le categorie presenti e i rispettivi 'padri';
- "visualizzaReport": legge dalla view *report* per poter vedere su quali annunci bisogna ancora riscuotere la percentuale;

Accesso in lettura/scrittura per le tabelle:

- annuncio
- categoria
- sottocategoria

Strutture di memorizzazione

Tabella <user>		
Attributo	**Tipo di dato**	**Attributi[2]**
Username	VARCHAR(50)	PK, NN
Password	CHAR(32)	NN
Ruolo	ENUM(“Utente”, “UtenteCertificato”, “Gestore”)	NN

Tabella <utente>		
Attributo	**Tipo di dato**	**Attributi[3]**
Username	VARCHAR(50)	PK, NN
Nome	VARCHAR(50)	NN
Cognome	VARCHAR(50)	NN
CodiceFiscale	CHAR(16)	NN
DataNascita	DATE	NN
IndirizzoResidenza	VARCHAR(50)	NN
IndirizzoFatturazione	VARCHAR(50)	

Tabella <utenteCertificato>		
Attributo	**Tipo di dato**	**Attributi[4]**
Username	VARCHAR(50)	PK, NN
IntestatarioCarta	VARCHAR(50)	NN
CodiceCarta	CHAR(16)	NN

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

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

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

CVC_CVV	CHAR(3)	NN
DataScadenza	DATE	NN

Tabella <annuncio>		
Attributo	**Tipo di dato**	**Attributi[5]**
idAnnuncio	INT	PK, NN,AI
Titolo	VARCHAR(50)	NN
Descrizione	VARCHAR(50)	NN
Categoria	VARCHAR(50)	NN
Prezzo	DECIMAL	NN
Foto	VARCHAR(50)	
StatoVendita	ENUM("Non Venduto", "Venduto")	NN
StatoRiscossione	ENUM("Non Riscosso", "Riscosso")	NN
utenteCertificato	VARCHAR(50)	NN

Tabella <categoria>		
Attributo	**Tipo di dato**	**Attributi[6]**
Nome	VARCHAR(50)	PK, NN

Tabella <sottocategoria>		
Attributo	**Tipo di dato**	**Attributi[7]**

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

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

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

Figlia	VARCHAR(50)	PK, NN
Genitore	VARCHAR(50)	PK,NN

Tabella <commento>		
Attributo	**Tipo di dato**	**Attributi**[8]
idCommento	INT	PK, NN, AI
idAnnuncioCommento	INT	PK,NN
Utente	VARCHAR(50)	NN
Contenuto	VARCHAR(100)	NN
Data_Ora	TIMESTAMP	NN

Tabella <conversazione>		
Attributo	**Tipo di dato**	**Attributi**[9]
idConversazione	INT	PK, NN,AI
Utente1	VARCHAR(50)	NN
Utente2	VARCHAR(50)	NN

Tabella <messaggio>		
Attributo	**Tipo di dato**	**Attributi**[10]
idMessaggio	INT	PK, NN,AI
idConversazioneMessaggio	INT	PK,NN
UtenteMittente	VARCHAR(50)	NN

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

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

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

Contenuto	VARCHAR(100)	NN
Data_Ora	TIMESTAMP	NN

Tabella <mezzocomunicazione>		
Attributo	**Tipo di dato**	**Attributi**[11]
Recapito	VARCHAR(50)	PK, NN
Utente	VARCHAR(50)	NN
Tipologia	ENUM("Fisso", "Cellulare", "Email")	NN
Preferito	TINYINT	NN

Tabella <segue>		
Attributo	**Tipo di dato**	**Attributi**[12]
idAnnuncioSegue	INT	PK, NN
Utente	VARCHAR(50)	PK,NN

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

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

Indici

Non sono stati utilizzati indici.

Trigger

COMMENTO_BEFORE_INSERT

Il trigger commento_BEFORE_INSERT è stato creato per evitare che un utente inserisca un commento su un annuncio venduto.

```
CREATE DEFINER = CURRENT_USER TRIGGER
`BachecaElettronicaAnnunci`.`commento_BEFORE_INSERT` BEFORE INSERT ON `commento`
FOR EACH ROW

BEGIN
if(SELECT StatoVendita
FROM annuncio
WHERE idAnnuncio = NEW.idAnnuncioCommento) = "Venduto"
then
SIGNAL SQLSTATE '45000' set MESSAGE_TEXT = "Annuncio non più attivo";
end if;

END
```

CONVERSAZIONE_BEFORE_INSERT

Il trigger conversazione_BEFORE_INSERT è stato creato per:

- Evitare che un utente cerchi di avviare una conversazione con se stesso;
- Evitare che un utente cerchi di avviare una conversazione con un utente che non abbia annunci non venduti.

```
CREATE DEFINER = CURRENT_USER TRIGGER
`BachecaElettronicaAnnunci`.`conversazione_BEFORE_INSERT` BEFORE INSERT ON
`conversazione` FOR EACH ROW

BEGIN

if((NEW.Utente1 = NEW.Utente2))
then
SIGNAL sqlstate '45000'SET MESSAGE_TEXT = 'Impossibile avviare la conversazione';
end if;

if not exists(select utentecertificato
                    from annuncio where annuncio.StatoVendita = 'Non Venduto'
      and annuncio.utenteCertificato = New.Utente2)
then
SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = "Impossibile avviare la conversazione.";
end if;
```

END

MEZZOCOMUNICAZIONE_BEFORE_INSERT

Il trigger mezzocomunicazione_BEFORE_INSERT è stato realizzato per far in modo che un utente venga avvisato se tenta di voler impostare un nuovo recapito come preferito, ma già ne esiste uno.

CREATE DEFINER = CURRENT_USER TRIGGER
`BachecaElettronicaAnnunci`.`mezzocomunicazione_BEFORE_INSERT` BEFORE INSERT ON `mezzocomunicazione` FOR EACH ROW

BEGIN

if(NEW.Preferito = 1 AND
(SELECT Preferito
FROM mezzocomunicazione
WHERE mezzocomunicazione.Utente = NEW.Utente AND mezzocomunicazione.Preferito = 1))

then

SIGNAL SQLSTATE '45000' set MESSAGE_TEXT = 'Ogni utente può avere un solo mezzo di comunicazione preferito';

END if;

END

SEGUE_BEFORE_INSERT

Il trigger segue_BEFORE_INSERT è stato creato per:

- Evitare che un utente segua due o più volte lo stesso annuncio;
- Evitare che un utente cerca di seguire un annuncio da lui stesso pubblicato;
- Evitare che un utente segua un annuncio già venduto.

CREATE DEFINER = CURRENT_USER TRIGGER
`BachecaElettronicaAnnunci`.`segue_BEFORE_INSERT` BEFORE INSERT ON `segue` FOR EACH ROW

BEGIN

if EXISTS(SELECT segue.Utente FROM segue WHERE segue.Utente = NEW.Utente AND segue.idAnnuncioSegue = NEW.idAnnuncioSegue)

then

SIGNAL SQLSTATE "45000" SET MESSAGE_TEXT = "L'annuncio è già stato seguito.";
END if;

if(SELECT idAnnuncio FROM annuncio WHERE utenteCertificato = NEW.Utente AND idAnnuncio = NEW.idAnnuncioSegue)
then

```
SIGNAL SQLSTATE '45000' set MESSAGE_TEXT = 'Non puoi seguire un annuncio da te creato';
END if;

if(SELECT StatoVendita FROM annuncio WHERE idAnnuncio = NEW.idAnnuncioSegue) =
"Venduto"
then
SIGNAL SQLSTATE "45000" SET MESSAGE_TEXT = "L'annuncio che vuoi seguire è stato
venduto.";
END if;

END
```

UTENTE_BEFORE_UPDATE

Il trigger Utente_BEFORE_UPDATE è stato creato per verificare se l'utente nell'aggiornare le proprie informazioni anagrafiche lasci qualche campo vuoto. In tal caso si provvede a non apportare alcuna modifica al rispettivo campo.

```
CREATE DEFINER = CURRENT_USER TRIGGER
`BachecaElettronicaAnnunci`.`Utente_BEFORE_UPDATE` BEFORE UPDATE ON `Utente` FOR
EACH ROW

BEGIN
if NEW.Nome = '' then
set NEW.Nome = old.Nome;
end if;

 if NEW.Cognome = '' then
set NEW.Cognome = old.Cognome;
end if;

 if NEW.CodiceFiscale = '' or length(NEW.CodiceFiscale) != 16 then
set NEW.CodiceFiscale = OLD.CodiceFiscale;
end if;

if NEW.DataNascita is null then
SET NEW.DataNascita = OLD.DataNascita;
END if;

if NEW.IndirizzoResidenza = '' then
set NEW.IndirizzoResidenza = OLD.IndirizzoResidenza;
end if;

if NEW.IndirizzoFatturazione = '' then
SET NEW.IndirizzoFatturazione = OLD.IndirizzoFatturazione;
end if;

END
```

ANNUNCIO_BEFORE_DELETE

Il trigger annuncio_BEFORE_DELETE è stato realizzato per evitare che possano essere eliminati annunci venduti, ma su cui i gestori di servizio devono ancora riscuotere la percentuale.

```
CREATE DEFINER = CURRENT_USER TRIGGER
`BachecaElettronicaAnnunci`.`annuncio_BEFORE_DELETE` BEFORE DELETE ON `annuncio`
FOR EACH ROW

BEGIN

if(OLD.StatoVendita = "Venduto" AND OLD.StatoRiscossione = "Non Riscosso")

then

SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = "Annuncio esistente nel sistema, ma non
eliminabile poiche i gestori devono riscuotere la percentuale di vendita.";

End if;

END
```

SOTTOCATEGORIA_BEFORE_INSERT

Il trigger Sottocategoria_BEFORE_INSERT è stato creato per evitare che si crei una categoria che sia figlia di se stessa.

```
CREATE DEFINER = CURRENT_USER TRIGGER
`BachecaElettronicaAnnunci`.`Sottocategoria_BEFORE_INSERT` BEFORE INSERT ON
`Sottocategoria` FOR EACH ROW

BEGIN

if(NEW.Figlia = NEW.Genitore)
then
SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = "La categoria 'Padre' e 'Figlia' non possono
essere le stesse";
END if;

END
```

Eventi

Nessun evento è stato implementato.

Viste

- La view *report* è stata creata per realizzare i report degli utenti in maniera tale che i gestori sanno da quali utenti devono ancora riscuotere la percentuale sull'oggetto venduto.

```
CREATE VIEW `report` AS
SELECT utentecertificato AS Venditore, IntestatarioCarta as `Intestatario Carta`,
codiceCarta AS `Codice Carta`, utentecertificato.CVC_CVV as `CVC\CVV`,
utentecertificato.DataScadenza as `Data Scadenza`, SUM(annuncio.Prezzo * 0.03) AS Debito
FROM annuncio JOIN utentecertificato ON annuncio.utenteCertificato =
utentecertificato.UsernameWHERE annuncio.StatoRiscossione = "Non Riscosso" AND
```

```
annuncio.StatoVendita = "Venduto"
group by annuncio.utenteCertificato;
```

- La view *bachecaPubblica* è stata realizzata per permettere agli utenti di poter visualizzare tutti gli annunci pubblicati e non venduti.

```
CREATE VIEW `bachecaPubblica` AS
select annuncio.idAnnuncio AS `Id Annuncio`,annuncio.utenteCertificato AS `Venditore`,
annuncio.Categoria AS `Categoria`,annuncio.Titolo AS `Titolo`,annuncio.Descrizione AS
`Descrizione`, annuncio.Prezzo AS `Prezzo`
from annuncio
where (annuncio.StatoVendita = 'Non Venduto')
```

Stored Procedures e transazioni

1) aggiornaContattoPreferito: usata dall'utente e utente certificato per poter aggiornare un contatto a preferito.

Prima di effettuare il vero e proprio aggiornamento, vengono effettuati due controlli:

1. Se il recapito inserito in input dall'utente loggato sia di suo possesso*;*

2. Se il recapito inserito in input dell'utente loggato sia già impostato come preferito;

Nel caso in cui nessuno dei due controlli viene soddisfatto si procede all'aggiornamento della preferenza del contatto. In particolare, verranno effettuati due update:

1. Setta a 0 un eventuale recapito dell'utente loggato già impostato come preferito;
2. Setta a 1 il recapito inserito in input

```
CREATE PROCEDURE `aggiornaContattoPreferito` (in contatto varchar(50), in utente varchar(50))

BEGIN

if not exists (select recapito
                from mezzocomunicazione
                where Recapito = contatto and mezzocomunicazione.Utente = utente)
        then
                signal sqlstate '45000'SET MESSAGE_TEXT = 'Il contatto inserito non è di tuo
possesso';
end if;

if  exists (select Recapito
                from mezzocomunicazione
```

```
            where (Utente = utente and Preferito = 1 AND recapito = contatto))
        then
            signal sqlstate '45000'SET MESSAGE_TEXT = 'Il contatto è già preferito';

ELSE
update mezzocomunicazione set Preferito = 0 WHERE (utente = utente and Preferito = 1);
update mezzocomunicazione set Preferito = 1 WHERE (recapito = contatto);
END if;
END
```

2) aggiornaInfoAnagrafiche: usata dall' utente e utente certificato per poter aggiornare le proprie informazioni anagrafiche. Ciò è stato reso possibile attraverso un update dei valori della tupla riferita all'utente loggato nella tabella *utente*.

```
CREATE  PROCEDURE `aggiornaInfoAnagrafiche`(
        IN var_nome VARCHAR(50),
        IN var_cognome VARCHAR(50),
        IN var_cf char(16),
        IN var_dataNascita DATE,
        IN var_indResidenza VARCHAR(50),
        IN var_indFatturazione VARCHAR(50),
        IN var_username VARCHAR(50)
)
BEGIN
UPDATE utente
SET
Nome = var_nome ,
Cognome = var_cognome,
CodiceFiscale = var_cf,
DataNascita= var_dataNascita,
IndirizzoResidenza = var_indResidenza,
IndirizzoFatturazione = var_indFatturazione
WHERE Username  = var_username;
```

```
END
```

3) certificaUtente: usata per permettere agli utenti di certificarsi e quindi di poter usare i privilegi dell'utente certificato. Ciò è stato reso possibile attraverso un insert nella tabella *utentecertificato* e un update del ruolo nella tabella *user*.

```
CREATE PROCEDURE `certificaUtente`(
    IN var_username VARCHAR(50),
    IN var_intestatarioCarta VARCHAR(50),
    IN var_codiceCarta CHAR(16),
    IN var_cvc_cvv CHAR(3),
    IN var_dataScadenza DATE
)
BEGIN
insert INTO utentecertificato (Username, IntestatarioCarta, CodiceCarta, CVC_CVV, DataScadenza) VALUES (var_username, var_intestatarioCarta, var_codiceCarta, var_cvc_cvv, var_dataScadenza);
UPDATE user set Ruolo = "UtenteCertificato" WHERE Username = var_username;
END
```

4) creaAnnuncio: usata dall'utente certificato per poter inserire un nuovo annuncio. Innanzitutto, viene verificato se la categoria inserita in input esiste nella tabella *categoria*. Nel caso la categoria esistesse, viene effettuato un insert nella tabella *annuncio* di tutti gli input, altrimenti viene avvisato l'utente certificato che la categoria in cui vuole aggiungere l'annuncio non esiste e quindi l'operazione non va a buon fine.

```
CREATE PROCEDURE `creaAnnuncio`(
    IN var_titolo VARCHAR(50),
    IN var_descrizione VARCHAR(100),
    IN var_prezzo DECIMAL(10,0),
    IN var_foto VARCHAR(50),
    IN var_categoria VARCHAR(50),
    IN var_utenteCertificato VARCHAR(50))
```

```
BEGIN

if var_categoria not in (select Nome from Categoria)
then
signal sqlstate '45000' set message_text = 'La categoria in cui vuoi inserire l'annuncio non esiste.';

else
INSERT INTO annuncio (Titolo, Descrizione, Prezzo, Foto, Categoria, UtenteCertificato) VALUES (var_titolo, var_descrizione, var_prezzo, var_foto, var_categoria, var_utentecertificato);
end if;

END
```

5) creaCategoria: usata dai gestori di servizio per poter creare delle nuove categorie. Innanzitutto, viene effettuato un insert nella tabella *categoria* del nome della categoria da aggiungere. Successivamente dopo aver verificato che l'input del secondo parametro non sia NULL, viene inserito nella tabella *sottocategoria* la coppia "figlio – padre" ovvero la sottocategoria e la rispettiva categoria 'padre'.

```
CREATE PROCEDURE `creaCategoria` (in var_nome varchar(50), in var_parent varchar(50))

BEGIN

INSERT INTO categoria(Nome) VALUES(var_nome);
if(var_parent is not null)
then
INSERT INTO sottocategoria(Figlia, Genitore) VALUES (var_nome, var_parent);
end if;

END
```

6) creaCommento: usata dall'utente e utente certificato per poter inserire un commento ad un annuncio presente nella bacheca pubblica. In particolare, se il numero intero preso in input corrisponde all'id di un annuncio presente nella tabella *annuncio,* si può procedere alla pubblicazione del commento, altrimenti l'utente verrà avvisato che non ha potuto svolgere la sua operazione.

```
CREATE  PROCEDURE `creaCommento`(
        IN var_annuncio INT,
```

```
        IN var_utente VARCHAR(50),
        IN var_contenuto VARCHAR(100)
)

BEGIN

declare exit handler for sqlexception
begin
        rollback;
        resignal;
end;

set transaction isolation level read committed;
start transaction;

if exists (SELECT idAnnuncio FROM annuncio WHERE idAnnuncio = var_annuncio)
then
INSERT INTO commento (idAnnuncioCommento,Utente, Contenuto, Data_Ora) VALUES (var_annuncio, var_utente, var_contenuto, CURRENT_TIMESTAMP());

ELSE
SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = "Impossibile pubblicare il commento.";
end if;
commit;

END
```

7) creaCommentoSeguiti: usata dall'utente e utente certificato per poter creare dei commenti agli annunci seguiti. Innanzitutto, attraverso la join tra le tabelle *annuncio* e *segui*, è stato verificato se il numero intero preso in input corrispondesse all'id di un annuncio seguito. Se il controllo venisse soddisfatto, si potrebbe procedere alla pubblicazione del commento, altrimenti l'utente verrà avvisato che non ha potuto svolgere la sua operazione.

```
CREATE PROCEDURE `creaCommentoSeguiti` (in var_idAnnuncio int, in var_username varchar(50),    IN var_contenuto VARCHAR(100))

BEGIN
```

```
if exists (select annuncio.idAnnuncio as Annuncio
from Segue join Annuncio on Segue.idAnnuncioSegue = Annuncio.idAnnuncio where idAnnuncio =
var_idAnnuncio)

then

INSERT INTO commento (idAnnuncioCommento,Utente, Contenuto, Data_Ora) VALUES
(var_idAnnuncio, var_username, var_contenuto, current_timestamp());

else
signal sqlstate '45000' set message_text = "Impossibile pubblicare il commento.";
end if;

END
```

8) creaContatto: usato dall' utente e utente certificato per poter creare un nuovo contatto. Ciò è stato reso possibile attraverso un insert nella tabella *mezzocomunicazione* dei dati presi in input.

```
CREATE PROCEDURE `creaContatto`(
        IN var_utente VARCHAR(50),
        IN var_recapito VARCHAR(50),
        IN var_tipologia ENUM('Fisso','Cellulare','Email'),
        IN var_preferito TINYINT
)

BEGIN

INSERT INTO mezzocomunicazione VALUES (var_utente, var_recapito,
var_tipologia,var_preferito);

END
```

9) creaGestore: permette di poter creare un nuovo gestore di servizio. Ciò è stato realizzato attraverso un semplice insert nella tabella *user*.

```
CREATE PROCEDURE `creaGestore` (in username varchar(50), in password varchar(50))

BEGIN

INSERT INTO user values(username,MD5(password), 'Gestore');
```

END

10) creaUtente: permette di poter creare un nuovo utente che potrà loggarsi e usare i servizi del sistema.
Il tutto è reso possibile attraverso un insert su *user* e uno su *utente*.

```
CREATE PROCEDURE `creaUtente`(
    IN username VARCHAR(50),
    IN password CHAR(32),
    IN nome VARCHAR(50),
    IN cognome VARCHAR(50),
    IN cf CHAR(16),
    IN dataNascita DATE,
    IN indResidenza VARCHAR(50),
    IN indFatturazione VARCHAR(50)
    )
BEGIN
INSERT INTO user values(username,MD5(password), 'Utente');
INSERT INTO utente VALUES(username,nome, cognome, cf, dataNascita, indResidenza, indFatturazione);
END
```

11) eliminaAnnuncio: usata dall'utente certificato per eliminare un annuncio pubblicato. Innanzitutto, si verifica se nella tabella *annuncio* esiste un annuncio con valore *idAnnuncio* uguale all'intero preso in input e valore *UtenteCertificato* uguale all' username dell'utente loggato. Se ciò è verificato, l'annuncio viene eliminato altrimenti l'utente loggato viene avvisato che l'annuncio non esiste.

```
CREATE  PROCEDURE `eliminaAnnuncio`(
    IN var_annuncio INT,
    IN var_username VARCHAR(50)
)
```

```
BEGIN

if not exists(SELECT idAnnuncio
from annuncio
where idAnnuncio = var_annuncio and UtenteCertificato = var_username )

then
SIGNAL sqlstate '45000' SET MESSAGE_TEXT = "Annuncio da eliminare non esiste.";
else
DELETE FROM annuncio WHERE idAnnuncio = var_annuncio AND utentecertificato =
var_username;
end if;
END
```

12) eliminaContatto: usato dall' utente e utente certificato per poter eliminare un contatto. Innanzitutto, si va a controllare se il recapito preso in input esiste nella tabella *mezzocomunicazione* ed appartiene all'utente loggato. Se ciò è verificato, il recapito viene eliminato altrimenti si avvisa l'utente che non esiste.

```
CREATE  PROCEDURE `eliminaContatto`(
        IN var_username VARCHAR(50),
        IN var_recapito VARCHAR(50)
)

BEGIN

declare c varchar(50);

SELECT Recapito from mezzocomunicazione where Recapito = var_recapito and Utente =
var_username INTO c;
if(c is null) then
signal sqlstate '45000' set message_text = "Il contatto non esiste";
end if;
delete from mezzocomunicazione where Recapito = var_recapito and Utente = var_username;

END
```

13) inviaMessaggio: usato dall'utente e utente certificato per poter inviare un messaggio ad un utente che ha un annuncio pubblicato (non venduto).
Innanzitutto, dopo aver controllato se l'utente che si vuole contattare abbia un annuncio pubblicato e non venduto, si verifica se esiste già una conversazione tra i due utenti.
Se la conversazione esiste, il messaggio verrà direttamente inviato, altrimenti dovrà essere creata la conversazione tra i due utenti attraverso un insert nella tabella *conversazione*. Fatto ciò, verrà recuperato l'id della conversazione appena creata per poter successivamente inviare il messaggio.

```
CREATE PROCEDURE `inviaMessaggio` (in var_utente1 varchar(50), in var_utente2 varchar(50), in contenuto varchar(100))

BEGIN

declare var_idConversazione int;
declare idConv int;

declare exit handler for sqlexception

begin
        rollback;
        resignal;
end;

set transaction isolation level read committed;
start transaction;


if exists(select utentecertificato
                from annuncio where annuncio.StatoVendita = 'Non Venduto'
    and annuncio.utenteCertificato = var_utente2)

then
        select idConversazione into idConv
        from Conversazione
        where (Utente1 = var_utente1 and Utente2 = var_utente2) or
        (Utente2 = var_utente1 and Utente1 = var_utente2);

if(idConv is null)
        then
```

```
        insert into Conversazione(Utente1,Utente2) values (var_utente1,var_utente2);
        Select last_insert_id() INTO var_idConversazione FROM conversazione LIMIT 1;
        insert into Messaggio(idConversazioneMessaggio,UtenteMittente,Contenuto,Data_Ora) values(var_idConversazione,var_utente1,contenuto,CURRENT_TIMESTAMP());

else
        insert into Messaggio(idConversazioneMessaggio,UtenteMittente,Contenuto,Data_Ora) values(idConv,var_utente1,contenuto,CURRENT_TIMESTAMP());
end if;

else
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = "L'utente che si vuole contattare non ha annunci pubblicati.";
end if;
commit;

END
```

14) inviaMessaggioSegui: usato dall'utente e utente certificato per poter inviare un messaggio ad un utente che ha un annuncio pubblicato che viene seguito.
Innanzitutto, dopo aver controllato attraverso la join tra le tabelle *annuncio* e *segui* che l'utente loggato stia per contattare un utente di cui segue un annuncio, si verifica se esiste già una conversazione tra i due utenti.
Se la conversazione esistesse, il messaggio verrà direttamente inviato, altrimenti dovrà essere creata la conversazione tra i due utenti attraverso un insert nella tabella *conversazione*. Fatto ciò, verrà recuperato l'id della conversazione inserita per poter successivamente inviare il messaggio. Nel caso l'utente loggato voglia contattare un utente di cui non segue alcun annuncio verrà avvisato che non è possibile svolgere l'operazione.

```
CREATE PROCEDURE `inviaMessaggioSegui` (in var_utente1 varchar(50), in var_utente2 varchar(50), in contenuto varchar(100))

BEGIN

declare var_idConversazione int;
declare idConv int;

declare exit handler for sqlexception
```

```
begin
        rollback;
        resignal;
end;

set transaction isolation level read committed;
start transaction;

if exists(select annuncio.idAnnuncio as Annuncio
                        from Segue join Annuncio on Segue.idAnnuncioSegue =
Annuncio.idAnnuncio
        WHERE annuncio.utenteCertificato = var_utente2 AND segue.Utente = var_utente1)
then
        select idConversazione into idConv
        from Conversazione
        where (Utente1 = var_utente1 and Utente2 = var_utente2) or
        (Utente2 = var_utente1 and Utente1 = var_utente2);

        if(idConv is null)
        then
        insert into Conversazione(Utente1,Utente2) values (var_utente1,var_utente2);
        Select last_insert_id() INTO var_idConversazione FROM conversazione LIMIT 1;
        insert into Messaggio(idConversazioneMessaggio,UtenteMittente,Contenuto,Data_Ora)
values(var_idConversazione,var_utente1,contenuto,CURRENT_TIMESTAMP());

else

        insert into Messaggio(idConversazioneMessaggio,UtenteMittente,Contenuto,Data_Ora)
values(idConv,var_utente1,contenuto,CURRENT_TIMESTAMP());
end if;

else
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Stai cercando di contattare un
utente di cui non segui alcun annuncio.';
end if;
commit;

END
```

15) login: usata per poter eseguire il login. Si controlla l'esistenza di username e password nella tabella *users* per poi restituire il ruolo al client.

```
CREATE PROCEDURE `login`(
        IN   var_username  VARCHAR(45),
        IN   var_password VARCHAR(45),
        OUT  var_role  INT
)
BEGIN
        declare var_user_role ENUM('Utente', 'UtenteCertificato', 'Gestore');
    select `ruolo` into var_user_role
         from `user`
                where `Username` = var_username  and `Password` = md5(var_password)
        -- See the corresponding enum in the client
                if var_user_role = 'Utente' then
                        set var_role =  1;
                elseif var_user_role = 'UtenteCertificato' then
                        set var_role = 2;
                elseif var_user_role = 'Gestore' then
                        set var_role = 3;
                else
                        set var_role = 4;
                end if;
END
```

16) riscuotiPercentuale: usata dai gestori di servizio per poter riscuotere le percentuali sugli oggetti venduti da ogni utente che ha pubblicato un annuncio.
Innanzitutto, si verifica se in *report* esiste un utente con username uguale a quello inserito in input. Se tale utente esistesse si andrebbe ad aggiornare lo stato della riscossione nella tabella *annuncio* degli oggetti la cui percentuale di guadagno è stata riscossa. Altrimenti il gestore sarà informato che o l'utente non ha alcun debito o non esiste.

```
CREATE PROCEDURE `riscuotiPercentuale` (in var_utente varchar(50))

BEGIN

        declare exit handler for sqlexception
   begin
      rollback;
      resignal;
   end;

   set transaction isolation level read committed;
   start transaction;

if exists (SELECT report.Venditore FROM report WHERE report.Venditore = var_utente)
then
update Annuncio
 set StatoRiscossione = 'Riscosso'
 where (StatoRiscossione = 'Non Riscosso' and UtenteCertificato = var_utente and StatoVendita =
'Venduto');
else

SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = "L'utente non ha alcun debito o non esiste";
end if;

COMMIT;

END
```

17) rispondiConversazione: usata dall'utente e utente certificato che vuole rispondere ad una conversazione già avviata. Innanzitutto, si va recuperare l'id della conversazione tra l'utente loggato e l'utente a cui si desidera rispondere e poi avviene l'invio del messaggio che andrà a comporre la conversazione di cui si è stato recuperato l'id.

```
CREATE PROCEDURE `rispondiConversazione` (in var_utente1 varchar(50), in var_utente2
varchar(50), in contenuto varchar(100))

BEGIN

DECLARE idConv int;
```

```
declare exit handler for sqlexception

begin
        rollback;
        resignal;
end;

set transaction isolation level read committed;
start transaction;

select idConversazione into idConv
from Conversazione
where (Utente1 = var_utente1 and Utente2 = var_utente2) or
(Utente2 = var_utente1 and Utente1 = var_utente2);

insert into Messaggio(idConversazioneMessaggio,UtenteMittente,Contenuto,Data_Ora)
values(idConv,var_utente1,contenuto,CURRENT_TIMESTAMP());
commit;

END
```

18) segnaVenduto: usata dall'utente certificato per poter segnare un oggetto come venduto. Ciò è stato reso possibile facendo prima una verifica sulla tabella *annuncio* dell'esistenza di un annuncio non venduto, con id uguale all'intero passato in input e con username dell'utente 'venditore' uguale all'username dell'utente loggato. Se il controllo non venisse soddisfatto si effettua l'update sulla tabella *annuncio* di 'StatoVendita' da 'Non Venduto' a 'Venduto' altrimenti viene comunicato all'utente che l'annuncio non esiste o è stato venduto.

```
CREATE PROCEDURE `segnaVenduto` (in var_username varchar(50), in idAnnuncio int)

BEGIN

if not exists(
select idAnnuncio
from annuncio
WHERE annuncio.idAnnuncio = idAnnuncio and annuncio.StatoVendita = "Non Venduto" and
annuncio.UtenteCertificato = var_username)
```

```
then
signal sqlstate '45000' set message_text = "L'annuncio non esiste o è già stato venduto";

end if;

update Annuncio
set annuncio.StatoVendita = "Venduto"
where annuncio.idAnnuncio= idAnnuncio and annuncio.utenteCertificato = var_username;

END
```

19) segueAnnuncio: usato dall'utente e utente certificato per poter seguire un annuncio con id pari all'intero preso in input.

```
CREATE  PROCEDURE `segueAnnuncio`(

        IN var_idAnnuncio INT,
        IN var_utente VARCHAR(50)

)

BEGIN
INSERT INTO segue VALUES(var_idAnnuncio, var_utente);

END
```

20) visualizzaAnnunciSeguiti: usata dagli utenti e utenti certificati per poter visualizzare tutti gli annunci che ha deciso di seguire. La visualizzazione di tutti gli annunci seguiti e relativi attributi è stata resa possibile dal join fra le tabelle *Segue* e *Annuncio*.

```
CREATE PROCEDURE `visualizzaAnnunciSeguiti` (in var_username varchar(50))

BEGIN

select annuncio.idAnnuncio as Annuncio, UtenteCertificato as `Venditore`,Titolo,Prezzo,

StatoVendita as `Stato Vendita`

from Segue join Annuncio on Segue.idAnnuncioSegue = Annuncio.idAnnuncio;

END
```

21) visualizzaBachecaPubblica: usata dall'utente e utente certificato per poter visualizzare tutti gli annunci messi in vendita.

```
CREATE PROCEDURE `visualizzaBachecaPubblica` ()

BEGIN

Select * from bachecaPubblica;

END
```

22) visualizzaCategoria: usata da gestore e utente certificato per poter visualizzare le categorie disponibili. Con l'ausilio di un left join fra *Categoria* e *Sottocategoria* è stato permesso di poter avere una tabella contenente tutte le categorie 'figlie' e relative categorie 'padre'.

```
CREATE PROCEDURE `visualizzaCategoria` ()

BEGIN

select Nome as Categoria, Genitore as `Categoria Padre`
from Categoria left join Sottocategoria on Nome = Figlia;

END
```

23) visualizzaChatInCorso: usata dall' utente e utente certificato per poter visualizzare quali sono le conversazioni attive e quale è l'ultimo messaggio scambiato in ognuna di esse.
La stored procedure si può dividere in due parti:
1. Viene preso l'idConversazione e l'username dell'utente con cui l'utente loggato ha scambiato almeno un messaggio;
2. Viene preso l'id della conversazione e il contenuto del relativo ultimo messaggio.
Infine per ottenere il risultato finale, è stato fatto il join tra queste due parti.

```
CREATE PROCEDURE `visualizzaChatInCorso` (in username varchar(50))

BEGIN

select Utente, Contenuto as `Ultimo messaggio` from(
select idConversazione, Utente2 as Utente from Conversazione where Utente1 = username
union
select idConversazione, Utente1 as Utente from Conversazione where Utente2 = username) as
```

```
utenteChat
join
(select idConv, Contenuto from
(select max(idMessaggio) as idMex, idConversazioneMessaggio as idConv from Messaggio group by idConv) as mexMax
join
(select Contenuto, idMessaggio from  Messaggio) as contenuto
on mexMax.idMex = contenuto.idMessaggio)as idConvCont on idConvCont.idConv= utenteChat.idConversazione;


END
```

24) visualizzaChatIntera: usata dall'utente e utente certificato per poter visualizzare i messaggi di un'intera conversazione. Ciò è stato reso possibile prendendo i mittenti e i rispettivi contenuti dei messaggi della conversazione che ha id corrispondente alla conversazione tra l'utente loggato e quello passato in input.

```
CREATE PROCEDURE `visualizzaChatIntera` (in var_utente1 varchar(50), in var_utente2 varchar(50))

BEGIN

select UtenteMittente AS Mittente, Contenuto
from Messaggio where idConversazioneMessaggio IN
                    (select idConversazione
                    from Conversazione
                    where (Utente1 = var_utente1 and Utente2 = var_utente2) or (Utente2 = var_utente1 and Utente1 = var_utente2));

END
```

25) visualizzaCommenti: usato dall' utente e utente certificato per poter visualizzare i commenti che hanno idAnnuncio uguale all' intero passato in input.

```
CREATE PROCEDURE `visualizzaCommenti`(IN `var_idAnnuncio` INT)

BEGIN
```

```
SELECT utente as Utente, Contenuto, Data_Ora as  `Data e Ora `
FROM commento
WHERE commento.idAnnuncioCommento = var_idAnnuncio;

END
```

26) visualizzaCommentiSeguiti: usata dall' utente e utente certificato per poter visualizzare i commenti degli annunci che segue.
La stored procedure si può dividere in due parti:
1. Attraverso la join tra le tabelle *annuncio* e *segue* vengono presi tutti gli annunci seguiti dall'utente loggato;
2. Attraverso la join tra le tabelle *annuncio* e *commento* vengono presi i commenti dell'annuncio che ha id uguale all'intero preso in input.
Infine per ottenere il risultato finale è stato fatto il join tra le due parti.

```
CREATE PROCEDURE `visualizzaCommentiSeguiti` (in var_idAnnuncio int, in var_username varchar(50))

BEGIN

SELECT utente as Utente, Contenuto from
(select annuncio.idAnnuncio as Annuncio
from Segue join Annuncio on Segue.idAnnuncioSegue = Annuncio.idAnnuncio where segue.utente = var_username) AS t1
join
(SELECT commento.idAnnuncioCommento, utente as Utente, Contenuto
FROM commento join annuncio on commento.idAnnuncioCommento = annuncio.idAnnuncio where annuncio.idAnnuncio = var_idAnnuncio) as t2

on t2.idAnnuncioCommento = t1.Annuncio;
END
```

27) visualizzaContatti: usato dall' utente e utente certificato per poter visualizzare i contatti posseduti.

```
CREATE PROCEDURE `visualizzaContatti` (in var_utente varchar(50))

BEGIN
```

```
SELECT  Recapito, Tipologia, convert(Preferito,CHAR) as Preferito

FROM mezzocomunicazione
WHERE Utente = var_utente;

END
```

28) visualizzaContattiVenditore: usato dall'utente e utente certificato loggato per poter visualizzare i contatti di un utente che ha almeno un annuncio pubblicato.
Attraverso la join tra le tabelle *annuncio* e *mezzocomunicazione* è stato reso possibile la selezione di recapiti di utenti con username uguale a quello preso in input e annunci pubblicati non venduti.

```
CREATE PROCEDURE `visualizzaContattiVenditore` (in var_utente varchar(50))

BEGIN

SELECT  distinct Recapito, Tipologia, convert(Preferito,CHAR) as Preferito
FROM mezzocomunicazione JOIN annuncio ON annuncio.UtenteCertificato =
mezzocomunicazione.Utente
WHERE utente = var_utente and annuncio.StatoVendita = "Non Venduto";

END
```

29) visualizzaContattiVenditoreSeguiti: usato dall'utente e utente certificato loggato per poter visualizzare i contatti di un venditore di cui segue l'annuncio. Si può dividere la procedure in 2 parti:
1.Sono stati recuperati tutti gli annunci seguiti;
2. Sono stati recuperati i recapiti degli utenti che hanno pubblicato almeno un annuncio.
Per ottenere il risultato finale è stato fatto il join tra due parti.

```
CREATE PROCEDURE `visualizzaContattiVenditoreSeguiti` (in var_utente varchar(50))

BEGIN

SELECT distinct Recapito, Tipologia, convert(Preferito,CHAR) as Preferito from
(select annuncio.idAnnuncio as Annuncio, UtenteCertificato as `Venditore`
from Segue join Annuncio on Segue.idAnnuncioSegue = Annuncio.idAnnuncio) AS t1
JOIN
(SELECT  Utente, Recapito, Tipologia, convert(Preferito,CHAR) as Preferito FROM
mezzocomunicazione JOIN annuncio ON annuncio.UtenteCertificato =
```

```
mezzocomunicazione.Utente) AS t2
ON t2.Utente = t1.Venditore
WHERE utente = var_utente;

END
```

30) visualizzaInfoAnagrafiche: usato da utente e utente certificato per poter visualizzare le proprie informazioni anagrafiche.

```
CREATE PROCEDURE `visualizzaInfoAnagrafiche` (in var_utente varchar(50))

BEGIN

select Nome, Cognome , CodiceFiscale as `Codice Fiscale`, DataNascita as `Data Nascita`, IndirizzoResidenza as `Indirizzo Residenza`,
IndirizzoFatturazione as `IndirizzoFatturazione`
from Utente
where Username = var_utente;

END
```

31) visualizzaMieiAnnunci: usato dall'utente certificato per poter visualizzare gli annunci pubblicati da lui stesso e non ancora venduti.

```
CREATE  PROCEDURE `visualizzaMieiAnnunci`(

        IN var_utente VARCHAR(50)
)

BEGIN

SELECT annuncio.idAnnuncio as Annuncio, annuncio.UtenteCertificato as Venditore,Titolo,Descrizione,convert(Prezzo,char) as Prezzo, Categoria, annuncio.StatoVendita as `Stato Vendita`
from annuncio
WHERE annuncio.UtenteCertificato = var_utente and annuncio.StatoVendita = "Non Venduto";

END
```

32) visualizzaReport: usata dal gestore per poter visualizzare quali sono gli utenti su cui il gestore deve ancora riscuote la percentuale sugli oggetti venduti.

```
CREATE PROCEDURE `visualizzaReport` ()

BEGIN

select * from report;
END
```

33) visualizzaAnnunci: usata dall'utente certificato per poter visualizzare tutti i suoi annunci.

```
CREATE PROCEDURE `visualizzaAnnunci` (in var_utente varchar(50))

BEGIN

SELECT annuncio.idAnnuncio as Annuncio, annuncio.UtenteCertificato as
Venditore,Titolo,Descrizione,convert(Prezzo,char) as Prezzo, categoria, annuncio.StatoVendita as
`Stato Vendita`, annuncio.StatoRiscossione as `Stato Riscossione`
from annuncio
WHERE annuncio.UtenteCertificato = var_utente;

END
```

Appendice: Implementazione

Codice SQL per instanziare il database

```
-- MySQL Script generated by MySQL Workbench

-- Mon Feb  7 14:42:10 2022

-- Model: New Model    Version: 1.0

-- 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 BachecaElettronicaAnnunci

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

DROP SCHEMA IF EXISTS `BachecaElettronicaAnnunci` ;


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

-- Schema BachecaElettronicaAnnunci

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

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

USE `BachecaElettronicaAnnunci` ;
```

```
-- -----------------------------------------------------
-- Table `BachecaElettronicaAnnunci`.`user`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `BachecaElettronicaAnnunci`.`user` ;

CREATE TABLE IF NOT EXISTS `BachecaElettronicaAnnunci`.`user` (
  `Username` VARCHAR(50) NOT NULL,
  `Password` CHAR(32) NOT NULL,
  `Ruolo` ENUM("Utente", "UtenteCertificato", "Gestore") NOT NULL,
  PRIMARY KEY (`Username`))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `BachecaElettronicaAnnunci`.`Utente`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `BachecaElettronicaAnnunci`.`Utente` ;

CREATE TABLE IF NOT EXISTS `BachecaElettronicaAnnunci`.`Utente` (
  `Username` VARCHAR(50) NOT NULL,
  `Nome` VARCHAR(50) NOT NULL,
  `Cognome` VARCHAR(50) NOT NULL,
  `CodiceFiscale` CHAR(16) NOT NULL,
```

```
  `DataNascita` DATE NOT NULL,
  `IndirizzoResidenza` VARCHAR(50) NOT NULL,
  `IndirizzoFatturazione` VARCHAR(50) NULL,
  PRIMARY KEY (`Username`))
ENGINE = InnoDB;



-- -----------------------------------------------------
-- Table `BachecaElettronicaAnnunci`.`utenteCertificato`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `BachecaElettronicaAnnunci`.`utenteCertificato` ;

CREATE TABLE IF NOT EXISTS `BachecaElettronicaAnnunci`.`utenteCertificato` (
  `Username` VARCHAR(50) NOT NULL,
  `IntestatarioCarta` VARCHAR(50) NOT NULL,
  `CodiceCarta` CHAR(16) NOT NULL,
  `CVC_CVV` CHAR(3) NOT NULL,
  `DataScadenza` DATE NOT NULL,
  INDEX `fk_utenteCertificato_Utente_idx` (`Username` ASC) VISIBLE,
  PRIMARY KEY (`Username`),
  CONSTRAINT `fk_utenteCertificato_Utente`
    FOREIGN KEY (`Username`)
    REFERENCES `BachecaElettronicaAnnunci`.`Utente` (`Username`)
    ON DELETE CASCADE
```

```
    ON UPDATE CASCADE)
ENGINE = InnoDB;



-- -----------------------------------------------------
-- Table `BachecaElettronicaAnnunci`.`categoria`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `BachecaElettronicaAnnunci`.`categoria` ;

CREATE TABLE IF NOT EXISTS `BachecaElettronicaAnnunci`.`categoria` (
  `Nome` VARCHAR(50) NOT NULL,
  PRIMARY KEY (`Nome`))
ENGINE = InnoDB;



-- -----------------------------------------------------
-- Table `BachecaElettronicaAnnunci`.`annuncio`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `BachecaElettronicaAnnunci`.`annuncio` ;

CREATE TABLE IF NOT EXISTS `BachecaElettronicaAnnunci`.`annuncio` (
  `idAnnuncio` INT NOT NULL AUTO_INCREMENT,
  `Titolo` VARCHAR(50) NOT NULL,
  `Descrizione` VARCHAR(100) NOT NULL,
```

```
  `Categoria` VARCHAR(50) NOT NULL,
  `Prezzo` DECIMAL NOT NULL,
  `Foto` VARCHAR(50) NULL,
  `StatoVendita` ENUM('Non Venduto', 'Venduto') NOT NULL,
  `StatoRiscossione` ENUM('Non Riscosso', 'Riscosso') NOT NULL,
  `utenteCertificato` VARCHAR(50) NOT NULL,
  PRIMARY KEY (`idAnnuncio`),
  INDEX `fk_Annuncio_utenteCertificato1_idx` (`utenteCertificato` ASC) VISIBLE,
  INDEX `fk_Annuncio_categoria1_idx` (`Categoria` ASC) VISIBLE,
  CONSTRAINT `fk_Annuncio_utenteCertificato1`
    FOREIGN KEY (`utenteCertificato`)
    REFERENCES `BachecaElettronicaAnnunci`.`utenteCertificato` (`Username`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Annuncio_categoria1`
    FOREIGN KEY (`Categoria`)
    REFERENCES `BachecaElettronicaAnnunci`.`categoria` (`Nome`)
    ON DELETE CASCADE
    ON UPDATE CASCADE)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `BachecaElettronicaAnnunci`.`commento`
```

```
-- -----------------------------------------------------
DROP TABLE IF EXISTS `BachecaElettronicaAnnunci`.`commento` ;


CREATE TABLE IF NOT EXISTS `BachecaElettronicaAnnunci`.`commento` (
  `idCommento` INT NOT NULL AUTO_INCREMENT,
  `idAnnuncioCommento` INT NOT NULL,
  `Utente` VARCHAR(50) NOT NULL,
  `Contenuto` VARCHAR(100) NOT NULL,
  `Data_Ora` TIMESTAMP NOT NULL,
  PRIMARY KEY (`idCommento`, `idAnnuncioCommento`),
  INDEX `fk_Commento_Annuncio1_idx` (`idAnnuncioCommento` ASC) VISIBLE,
  INDEX `fk_Commento_Utente1_idx` (`Utente` ASC) VISIBLE,
  CONSTRAINT `fk_Commento_Annuncio1`
    FOREIGN KEY (`idAnnuncioCommento`)
    REFERENCES `BachecaElettronicaAnnunci`.`annuncio` (`idAnnuncio`)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  CONSTRAINT `fk_Commento_Utente1`
    FOREIGN KEY (`Utente`)
    REFERENCES `BachecaElettronicaAnnunci`.`Utente` (`Username`)
    ON DELETE CASCADE
    ON UPDATE CASCADE)
ENGINE = InnoDB;
```

```
-- -----------------------------------------------------
-- Table `BachecaElettronicaAnnunci`.`conversazione`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `BachecaElettronicaAnnunci`.`conversazione` ;

CREATE TABLE IF NOT EXISTS `BachecaElettronicaAnnunci`.`conversazione` (
  `idConversazione` INT NOT NULL AUTO_INCREMENT,
  `Utente1` VARCHAR(50) NOT NULL,
  `Utente2` VARCHAR(50) NOT NULL,
  PRIMARY KEY (`idConversazione`),
  INDEX `fk_Conversazione_Utente1_idx` (`Utente1` ASC) VISIBLE,
  INDEX `fk_Conversazione_Utente2_idx` (`Utente2` ASC) VISIBLE,
  CONSTRAINT `fk_Conversazione_Utente1`
    FOREIGN KEY (`Utente1`)
    REFERENCES `BachecaElettronicaAnnunci`.`Utente` (`Username`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Conversazione_Utente2`
    FOREIGN KEY (`Utente2`)
    REFERENCES `BachecaElettronicaAnnunci`.`Utente` (`Username`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

```
-- -----------------------------------------------------
-- Table `BachecaElettronicaAnnunci`.`messaggio`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `BachecaElettronicaAnnunci`.`messaggio` ;

CREATE TABLE IF NOT EXISTS `BachecaElettronicaAnnunci`.`messaggio` (
  `idMessaggio` INT NOT NULL AUTO_INCREMENT,
  `idConversazioneMessaggio` INT NOT NULL,
  `UtenteMittente` VARCHAR(50) NOT NULL,
  `Contenuto` VARCHAR(100) NOT NULL,
  `Data_Ora` TIMESTAMP NOT NULL,
  PRIMARY KEY (`idMessaggio`, `idConversazioneMessaggio`),
  INDEX `fk_messaggio_Conversazione1_idx` (`idConversazioneMessaggio` ASC) VISIBLE,
  INDEX `fk_messaggio_Utente1_idx` (`UtenteMittente` ASC) VISIBLE,
  CONSTRAINT `fk_messaggio_Conversazione1`
    FOREIGN KEY (`idConversazioneMessaggio`)
    REFERENCES `BachecaElettronicaAnnunci`.`conversazione` (`idConversazione`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_messaggio_Utente1`
    FOREIGN KEY (`UtenteMittente`)
    REFERENCES `BachecaElettronicaAnnunci`.`Utente` (`Username`)
```

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



-- -----------------------------------------------------
-- Table `BachecaElettronicaAnnunci`.`mezzocomunicazione`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `BachecaElettronicaAnnunci`.`mezzocomunicazione` ;

CREATE TABLE IF NOT EXISTS `BachecaElettronicaAnnunci`.`mezzocomunicazione` (
  `Utente` VARCHAR(50) NOT NULL,
  `Recapito` VARCHAR(50) NOT NULL,
  `Tipologia` ENUM('Fisso', 'Cellulare', 'Email') NOT NULL,
  `Preferito` TINYINT NOT NULL,
  PRIMARY KEY (`Recapito`),
  INDEX `fk_mezzocomunicazione_Utente1_idx` (`Utente` ASC) VISIBLE,
  CONSTRAINT `fk_mezzocomunicazione_Utente1`
    FOREIGN KEY (`Utente`)
    REFERENCES `BachecaElettronicaAnnunci`.`Utente` (`Username`)
    ON DELETE CASCADE
    ON UPDATE CASCADE)
ENGINE = InnoDB;
```

```
-- -----------------------------------------------------
-- Table `BachecaElettronicaAnnunci`.`segue`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `BachecaElettronicaAnnunci`.`segue` ;

CREATE TABLE IF NOT EXISTS `BachecaElettronicaAnnunci`.`segue` (
  `idAnnuncioSegue` INT NOT NULL,
  `Utente` VARCHAR(50) NOT NULL,
  INDEX `fk_segue_Annuncio1_idx` (`idAnnuncioSegue` ASC) VISIBLE,
  INDEX `fk_segue_Utente1_idx` (`Utente` ASC) VISIBLE,
  PRIMARY KEY (`idAnnuncioSegue`, `Utente`),
  CONSTRAINT `fk_segue_Annuncio1`
    FOREIGN KEY (`idAnnuncioSegue`)
    REFERENCES `BachecaElettronicaAnnunci`.`annuncio` (`idAnnuncio`)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  CONSTRAINT `fk_segue_Utente1`
    FOREIGN KEY (`Utente`)
    REFERENCES `BachecaElettronicaAnnunci`.`Utente` (`Username`)
    ON DELETE CASCADE
    ON UPDATE CASCADE)
ENGINE = InnoDB;
```

```
-- -----------------------------------------------------
-- Table `BachecaElettronicaAnnunci`.`Sottocategoria`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `BachecaElettronicaAnnunci`.`Sottocategoria` ;

CREATE TABLE IF NOT EXISTS `BachecaElettronicaAnnunci`.`Sottocategoria` (
  `Figlia` VARCHAR(50) NOT NULL,
  `Genitore` VARCHAR(50) NOT NULL,
  PRIMARY KEY (`Figlia`, `Genitore`),
  INDEX `fk_categoria_has_categoria_categoria2_idx` (`Genitore` ASC) VISIBLE,
  INDEX `fk_categoria_has_categoria_categoria1_idx` (`Figlia` ASC) VISIBLE,
  CONSTRAINT `fk_categoria_has_categoria_categoria1`
    FOREIGN KEY (`Figlia`)
    REFERENCES `BachecaElettronicaAnnunci`.`categoria` (`Nome`)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  CONSTRAINT `fk_categoria_has_categoria_categoria2`
    FOREIGN KEY (`Genitore`)
    REFERENCES `BachecaElettronicaAnnunci`.`categoria` (`Nome`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

```
SET SQL_MODE=@OLD_SQL_MODE;

SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS;

SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS;
```

Codice del Front-End

---

main.c

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

#include "defines.h"

typedef enum {
	UTENTE = 1,
	UTENTECERTIFICATO,
	GESTORE,
	FAILED_LOGIN
} role_t;



struct configuration conf;
static MYSQL* conn;


static role_t attempt_login(MYSQL* conn, char* username, char* password) {
	MYSQL_STMT* login_procedure;
	MYSQL_BIND param[3];
	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;
	}

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

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

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

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

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

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

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

	param[0].buffer_type = MYSQL_TYPE_LONG;
	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;
	}

	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;
}



static int attempt_registration(MYSQL* conn) {
```

```
MYSQL_STMT* registration_procedure;
MYSQL_BIND param[8];

char codFisc[17];
char nome[51];
char cognome[51];
char nascita[11];
char residenza[51];
char fatturazione[51];
MYSQL_TIME* date;

bool usernamenull;
bool passwordnull;
bool nomenull;
bool cognomenull;
bool cfnull;
bool datanull;
bool indResidenzanull;


utente:
printf("\nUsername: ");
getInput(50, conf.username);
printf("Password: ");
getInput(50, conf.password);
printf("Nome: ");
getInput(50, nome);
printf("Cognome: ");
getInput(50, cognome);
printf("Codice fiscale: ");
getInput(16, codFisc);
printf("Data compleanno (gg-mm-aaaa): ");
getInput(10, nascita);
printf("Indirizzo residenza: ");
getInput(50, residenza);
printf("Indirizzo fatturazione: ");
getInput(50, fatturazione);

if (strlen(codFisc) != 16) {
        printf("\nIl cf deve avere 16 caratteri.\n");
        goto utente;
}

date = malloc(sizeof(MYSQL_TIME*));
date->time_type = MYSQL_TIMESTAMP_DATE;

switch (getBirthDate(nascita, date)) {
case EXIT_SUCCESS:
        break;
case EXIT_FAILURE:
```

```
		goto utente;
	}

	if (conf.username[0] == '\0')
		usernamenull = true;
	else
		usernamenull = false;
	if (conf.password[0] == '\0')
		passwordnull = true;
	else
		passwordnull = false;
	if (nome[0] == '\0')
		nomenull = true;
	else
		nomenull = false;
	if (cognome[0] == '\0')
		cognomenull = true;
	else
		cognomenull = false;
	if (codFisc[0] == '\0' )
		cfnull = true;
	else
		cfnull = false;
	if (nascita[0] == '\0')
		datanull = true;
	else {
		datanull = false;
		}
	if (residenza[0] == '\0')
		indResidenzanull = true;
	else
		indResidenzanull = false;


	switch (validateParams()) {
	case EXIT_SUCCESS:
		break;
	case EXIT_FAILURE:
		goto utente;
	}

	if (!setup_prepared_stmt(&registration_procedure, "CALL creaUtente(?,?,?,?,?,?,?,?)", conn))
{
		print_stmt_error(registration_procedure, "Unable to initialize registration
statement\n");
		goto err2;
	}

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

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

	param[1].buffer_type = MYSQL_TYPE_VAR_STRING;
	param[1].buffer = conf.password;
	param[1].buffer_length = strlen(conf.password);
	param[1].is_null = &passwordnull;

	param[2].buffer_type = MYSQL_TYPE_VAR_STRING;
	param[2].buffer = nome;
	param[2].buffer_length = strlen(nome);
	param[2].is_null = &nomenull;

	param[3].buffer_type = MYSQL_TYPE_VAR_STRING;
	param[3].buffer = cognome;
	param[3].buffer_length = strlen(cognome);
	param[3].is_null = &cognomenull;

	param[4].buffer_type = MYSQL_TYPE_VAR_STRING;
	param[4].buffer = codFisc;
	param[4].is_null = &cfnull;

	param[5].buffer_type = MYSQL_TYPE_DATE;
	param[5].buffer = date;
	param[5].buffer_length = sizeof(date);
	param[5].is_null = &datanull;

	param[6].buffer_type = MYSQL_TYPE_VAR_STRING;
	param[6].buffer = residenza;
	param[6].buffer_length = strlen(residenza);
	param[6].is_null = &indResidenzanull;

	param[7].buffer_type = MYSQL_TYPE_VAR_STRING;
	param[7].buffer = fatturazione;
	param[7].buffer_length = strlen(fatturazione);

	if (mysql_stmt_bind_param(registration_procedure, param) != 0) {
		print_stmt_error(registration_procedure, "Could not bind parameters for
registration");
		goto err;
	}

	if (mysql_stmt_execute(registration_procedure) != 0) {
		print_stmt_error(registration_procedure, "Could not execute registration procedure");
		goto err;
	}

	mysql_stmt_close(registration_procedure);
```

```
	return EXIT_SUCCESS;
err:
	mysql_stmt_close(registration_procedure);
err2:
	return EXIT_FAILURE;
}


static int attempt_registration_gestore(MYSQL* conn) {

	MYSQL_STMT* registration_procedure;
	MYSQL_BIND param[2];

	bool usernamenull;
	bool passwordnull;

utente:
	printf("\nUsername: ");
	getInput(50, conf.username);
	printf("Password: ");
	getInput(50, conf.password);


	if (conf.username[0] == '\0')
		usernamenull = true;
	else
		usernamenull = false;
	if (conf.password[0] == '\0')
		passwordnull = true;
	else
		passwordnull = false;


	switch (validateParams()) {
	case EXIT_SUCCESS:
		break;
	case EXIT_FAILURE:
		goto utente;
	}


	if (!setup_prepared_stmt(&registration_procedure, "CALL creaGestore(?,?)", conn)) {
		print_stmt_error(registration_procedure, "Unable to initialize registration
statement\n");
		goto err2;
	}

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

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

```
	param[0].buffer = conf.username;
	param[0].buffer_length = strlen(conf.username);
	param[0].is_null = &usernamenull;

	param[1].buffer_type = MYSQL_TYPE_VAR_STRING;
	param[1].buffer = conf.password;
	param[1].buffer_length = strlen(conf.password);
	param[1].is_null = &passwordnull;


	if (mysql_stmt_bind_param(registration_procedure, param) != 0) {
		print_stmt_error(registration_procedure, "Could not bind parameters for
registration");
		goto err;
	}

	if (mysql_stmt_execute(registration_procedure) != 0) {
		print_stmt_error(registration_procedure, "Could not execute registration procedure");
		goto err;
	}

	mysql_stmt_close(registration_procedure);
	return EXIT_SUCCESS;
err:
	mysql_stmt_close(registration_procedure);
err2:
	return EXIT_FAILURE;
}



int main(void) {

	char choicesAccess[4] = { '1','2','3','4'};
	char resAccess;
	role_t role;

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

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

	if (mysql_real_connect(conn, conf.host, conf.db_username, conf.db_password, conf.database,
conf.port, NULL, CLIENT_MULTI_STATEMENTS | CLIENT_MULTI_RESULTS) == NULL) {
```

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

accessPage:
        printf("\033[2J\033[H");
        printf("Benvenuto a BachecaAnnunci!\n");
        printf("\nCosa vuoi fare?\n");
        printf("1) Accedi\n");
        printf("2) Registra Utente\n");
        printf("3) Registra Gestore\n");
        printf("4) Esci\n");
        resAccess = multiChoice("\nSeleziona un'opzione", choicesAccess, 4);
        switch (resAccess) {
        case '1':
            goto loginPage;
        case '2':
            goto registerPageUtente;
        case '3':
            goto registerPageGestore;
        case '4':
            goto exitPage;
        }

loginPage:
        printf("\nUsername: ");
        getInput(45, conf.username);
        printf("Password: ");
        getInput(45, conf.password);

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

        switch (role) {
        case UTENTE:
            run_as_utente(conn);
            goto accessPage;

        case UTENTECERTIFICATO:
            run_as_utenteCertificato(conn);
            goto accessPage;

        case GESTORE:
            run_as_gestore(conn);
            goto accessPage;

        case FAILED_LOGIN:
            fprintf(stderr, "Username o Password sono sbagliati\n");
            getchar();
            goto accessPage;
```

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

registerPageUtente:

	switch (attempt_registration(conn)) {
	case EXIT_FAILURE:
		printf("\nRiprova a registrarti.\n");
		getchar();
		goto accessPage;
	case EXIT_SUCCESS:
		printf("\nRegistrazione avvenuta con sucesso, effettua il login per iniziare a usare i
nostri servizi.\n");
		getchar();
		goto accessPage;
	}

registerPageGestore:
	switch (attempt_registration_gestore(conn)) {
	case EXIT_FAILURE:
		printf("\nRiprova a registrarti.\n");
		getchar();
		goto accessPage;
	case EXIT_SUCCESS:
		printf("\nRegistrazione avvenuta con sucesso, effettua il login.\n");
		getchar();
		goto accessPage;
	}

exitPage:
	printf("\033[2J\033[H");
	printf("Arrivederci!\n");
	mysql_close(conn);
	return EXIT_SUCCESS;
}
```

---

utente.c

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

```
void aggiorna_utente_certificato(MYSQL* conn) {

	MYSQL_STMT* prepared_stmt;
	MYSQL_BIND param[5];

	char cartaCredito[17];
	char intestatario[50];
	char cvc_cvv[4];
	char data[11];
	bool cartanull;
	bool intestatarionull;
	bool cvc_cvvnull;
	bool datanull;
	MYSQL_TIME* date;

	printf("\nInserisci i seguenti dati per autenticare l'utente.\n");
utente:
	printf("\nNumero carta di credito: ");
	getInput(17, cartaCredito);
	printf("Intestatario carta: ");
	getInput(50, intestatario);
	printf("CVC/CVV: ");
	getInput(3, cvc_cvv);
	printf("Data scadenza carta (mm-aa): ");
	getInput(10, data);

	if (strlen(cartaCredito) != 16) {
		printf("\nCarta di credito deve avere 16 caratteri.\n");
		goto utente;
	}

	if (strlen(cvc_cvv) != 3) {
		printf("\nCVC_CVV deve avere 3 caratteri.\n");
		goto utente;
	}

	switch (validateParams()) {
	case EXIT_SUCCESS:
		break;
	case EXIT_FAILURE:
		goto utente;
	}

	date = malloc(sizeof(MYSQL_TIME*));
	date->time_type = MYSQL_TIMESTAMP_DATE;


	switch (getExpirationDate(data, date)) {
	case EXIT_SUCCESS:
```

```
            break;
        case EXIT_FAILURE:
            goto utente;
        }

        if (cartaCredito[0] == '\0')
            cartanull = true;
        else
            cartanull = false;
        if (intestatario[0] == '\0')
            intestatarionull = true;
        else
            intestatarionull = false;
        if (cvc_cvv[0] == '\0')
            cvc_cvvnull = true;
        else
            cvc_cvvnull = false;
        if (data[0] == '\0')
            datanull = true;
        else
            datanull = false;

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

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

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

        param[1].buffer_type = MYSQL_TYPE_VAR_STRING;
        param[1].buffer = intestatario;
        param[1].buffer_length = strlen(intestatario);
        param[1].is_null = &intestatarionull;

        param[2].buffer_type = MYSQL_TYPE_VAR_STRING;
        param[2].buffer = cartaCredito;
        param[2].buffer_length = strlen(cartaCredito);
        param[2].is_null = &cartanull;

        param[3].buffer_type = MYSQL_TYPE_VAR_STRING;
        param[3].buffer = cvc_cvv;
        param[3].buffer_length = strlen(cvc_cvv);
        param[3].is_null = &cvc_cvvnull;

        param[4].buffer_type = MYSQL_TYPE_DATE;
        param[4].buffer = date;
```

```
	param[4].buffer_length = sizeof(date);
	param[4].is_null = &datanull;

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

	if (mysql_stmt_execute(prepared_stmt) != 0) {
		print_stmt_error(prepared_stmt, "An error occurred while upgrading the user.");
	}
	else {
		printf("Utente certificato correttamente...\n");
	}
	getchar();
	mysql_stmt_close(prepared_stmt);
	return main();
}



void utentePage(MYSQL* conn) {

	char options[7] = { '1','2', '3','4','5','6','7'};
	char op;

	while (true) {
	start:
		printf("\033[2J\033[H");
		printf("Hai effetuato l'accesso come Utente\n");
		printf("\nCosa vuoi fare?\n");
		printf("1) Visualizza bacheca pubblica\n");
		printf("2) Visualizza annunci seguiti\n");
		printf("3) Visualizza chats\n");
		printf("4) Informazioni anagrafiche\n");
		printf("5) Contatti\n");
		printf("6) Autentica utente\n");
		printf("7) Esci\n");

		op = multiChoice("\nSeleziona un'opzione", options, 7);

		switch (op) {
		case '1':
			visualizza_bacheca_pubblica(conn);
			goto start;
		case '2':
			visualizza_annunci_seguiti(conn);
			goto start;
		case '3':
			visualizza_chat(conn);
```

```
                goto start;
            case '4':
                gestisci_informazioni_personali(conn);
                goto start;
            case'5':
                gestisci_contatti(conn);
                goto start;
            case '6':
                aggiorna_utente_certificato(conn);
                goto start;
            case '7':
                return;
            default:
                fprintf(stderr, "Invalid condition at %s:%d\n", __FILE__, __LINE__);
                abort();
            }
            getchar();
        }
}


void run_as_utente(MYSQL* conn) {

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

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

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

        utentePage(conn);

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

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

---

utenteCertificato.c

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

#include "defines.h"

static void creaAnnuncio(MYSQL* conn) {

   MYSQL_STMT* prepared_stmt;
   MYSQL_BIND param[6];

   char titolo[51];
   char descrizione[101];
   char prezzo[9];
   char categoria[51];
   char foto[51];
   bool titolonull;
   bool descrizionenull;
   bool prezzonull;
   bool categorianull;


   if (!setup_prepared_stmt(&prepared_stmt, "call visualizzaCategoria()", conn)) {
      finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize categories list statement\n",
false);
   }

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

   dump_result_set(conn, prepared_stmt, "\n***Lista categoria***");
   mysql_stmt_close(prepared_stmt);

   start:
   printf("\nTitolo: ");
   getInput(50, titolo);
   printf("Descrizione: ");
   getInput(100, descrizione);
   printf("Categoria: ");
   getInput(50, categoria);
   printf("Foto: ");
   getInput(50, foto);
   printf("Prezzo: ");
   getInput(9, prezzo);
```

```
    if (titolo[0] == '\0')
        titolonull = true;
    else
        titolonull = false;
    if (descrizione[0] == '\0')
        descrizionenull = true;
    else
        descrizionenull = false;
    if (prezzo[0] == '\0')
        prezzonull = true;
    else
        prezzonull = false;
    if (categoria[0] == '\0')
        categorianull = true;
    else
        categorianull = false;

    switch (validateParams()) {
    case EXIT_SUCCESS:
        break;
    case EXIT_FAILURE:
        goto start;
    }

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

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

    param[0].buffer_type = MYSQL_TYPE_VAR_STRING;
    param[0].buffer = titolo;
    param[0].buffer_length = strlen(titolo);
    param[0].is_null = &titolonull;

    param[1].buffer_type = MYSQL_TYPE_VAR_STRING;
    param[1].buffer = descrizione;
    param[1].buffer_length = strlen(descrizione);
    param[1].is_null = &descrizionenull;

    param[2].buffer_type = MYSQL_TYPE_VAR_STRING;
    param[2].buffer = prezzo;
    param[2].buffer_length = strlen(prezzo);
    param[2].is_null = &prezzonull;

    param[3].buffer_type = MYSQL_TYPE_VAR_STRING;
    param[3].buffer = foto;
```

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

    param[4].buffer_type = MYSQL_TYPE_VAR_STRING;
    param[4].buffer = categoria;
    param[4].buffer_length = strlen(categoria);
    param[4].is_null = &categorianull;

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

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

    if (mysql_stmt_execute(prepared_stmt) != 0) {
        print_stmt_error(prepared_stmt, "An error occurred while creating the annuncement");
    }
    else {
        printf("Annuncio creato correttamente...\n");
    }
    getchar();
    mysql_stmt_close(prepared_stmt);
}


void visualizza_miei_annunci(MYSQL* conn) {

    MYSQL_STMT* prepared_stmt;
    MYSQL_BIND param;

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

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

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

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

    if (mysql_stmt_execute(prepared_stmt) != 0) {
        finish_with_stmt_error(conn, prepared_stmt, "Could not retrieve the list of your ads\n", true);
    }
```

```
    dump_result_set(conn, prepared_stmt, "\n***Lista degli annunci***");
    mysql_stmt_close(prepared_stmt);
}



void visualizza_annunci(MYSQL* conn) {

    MYSQL_STMT* prepared_stmt;
    MYSQL_BIND param;

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

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

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

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

    if (mysql_stmt_execute(prepared_stmt) != 0) {
        finish_with_stmt_error(conn, prepared_stmt, "Could not retrieve the list of your ads\n", true);
    }

    dump_result_set(conn, prepared_stmt, "\n***Lista degli annunci***");
    mysql_stmt_close(prepared_stmt);
}

void utenteCertificatoPage(MYSQL* conn) {

    char options[9] = { '1','2','3','4','5','6','7','8','9'};
    char op;

    while (true) {
    start:
        printf("\033[2J\033[H");
        printf("Hai effetuato l'accesso come Utente\n");
        printf("\nCosa vuoi fare?\n");
        printf("1) Visualizza bacheca pubblica\n");
        printf("2) Visualizza annunci seguiti\n");
        printf("3) Visualizza chats\n");
        printf("4) Informazioni anagrafiche\n");
        printf("5) Contatti\n");
        printf("6) Crea annuncio\n");
        printf("7) Segna annuncio come venduto\n");
```

```
printf("8) Rimuovi annuncio\n");
printf("9) Esci\n");

op = multiChoice("\nSeleziona un' opzione", options, 9);

switch (op) {
case '1':
   visualizza_bacheca_pubblica(conn);
   goto start;

case '2':
   visualizza_annunci_seguiti(conn);
   goto start;

case '3':
   visualizza_chat(conn);
   goto start;

case '4':
   gestisci_informazioni_personali(conn);
   goto start;
case '5':
   gestisci_contatti(conn);
   goto start;
case '6':
  {
  creaAnnuncio(conn);
  goto start;
  }
case '7':
{
   visualizza_miei_annunci(conn);
   segna_venduto(conn);
   goto start;
}
case '8':
{
   visualizza_annunci(conn);
   rimuovi_annuncio(conn);
   goto start;
}
case '9':
   return;
   return;
   default:
   fprintf(stderr, "Invalid condition at %s:%d\n", __FILE__, __LINE__);
   abort();
    }
}
getchar();
```

```
    }


void run_as_utenteCertificato(MYSQL* conn) {
    printf("Switching to administrative role...\n");

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

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

    utenteCertificatoPage(conn);

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

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


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

    int articolo;
    char articoloC[8];

    printf("\nInserisci l'id dell'annuncio da rimuovere: ");
    getInput(8, articoloC);

    articolo = atoi(articoloC);

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

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

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

```
    param[0].buffer = &articolo;
    param[0].buffer_length = sizeof(articolo);

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

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

    if (mysql_stmt_execute(prepared_stmt) != 0) {
        print_stmt_error(prepared_stmt, "An error occurred while removing the announcement.");
    }
    else {
        printf("Annuncio rimosso correttamente...\n");
    }
    getchar();
    mysql_stmt_close(prepared_stmt);
}



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

    char idAd[8];
    int idAnnuncio;

    printf("\nInserisci id annuncio: ");
    getInput(8, idAd);

    idAnnuncio = atoi(idAd);

    if (!setup_prepared_stmt(&prepared_stmt, "call segnaVenduto(?, ?)", conn)) {
        finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize mark as sold statement\n",
false);
    }

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

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

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

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

    if (mysql_stmt_execute(prepared_stmt) != 0) {
        print_stmt_error(prepared_stmt, "An error occurred while marking the annuncement as sold.");
    }
    else {
        printf("Annuncio venduto correttamente...\n");
    }
    getchar();
    mysql_stmt_close(prepared_stmt);
}
```

---

utils_utente_utenteCertificato.c

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

void visualizza_bacheca_pubblica(MYSQL* conn) {

    MYSQL_STMT* prepared_stmt;
    char op;
    char options[5] = { '1','2','3','4','5' };

    if (!setup_prepared_stmt(&prepared_stmt, "call visualizzaBachecaPubblica()", conn)) {
        finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize public notice board
statement\n", false);
    }

    if (mysql_stmt_execute(prepared_stmt) != 0) {
        finish_with_stmt_error(conn, prepared_stmt, "Could not retrieve the public notice board\n",
```

```
true);
    }

    dump_result_set(conn, prepared_stmt, "\n***Bacheca pubblica degli Annunci***");
    mysql_stmt_close(prepared_stmt);

        printf("\nCosa vuoi fare?\n");
        printf("1) Segui annuncio\n");
        printf("2) Visualizza contatti del Venditore\n");
        printf("3) Visualizza commenti\n");
        printf("4) Invia messaggio al venditore\n");
        printf("5) Indietro");

        op = multiChoice("\n\nSeleziona un' opzione", options, 5);

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

        case '2':
            visualizza_contatto_venditore(conn);
            break;

        case '3':
            visualizza_commenti(conn);
            break;

        case'4':
            contatta_venditore(conn);
            break;

        case'5':
            return;
        }
    }

void segui_annuncio(MYSQL* conn) {
    MYSQL_STMT* prepared_stmt;
```

```
    MYSQL_BIND param[2];
    char annuncio[8];
    int annuncio1;

    printf("\nInserisci id dell' annuncio da seguire: ");
    getInput(8, annuncio);
    annuncio1 = atoi(annuncio);

    if (!setup_prepared_stmt(&prepared_stmt, "call segueAnnuncio(?,?)", conn)) {
        finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize follow annuncement statement\n", false);
    }
    memset(param, 0, sizeof(param));
    param[0].buffer_type = MYSQL_TYPE_LONG;
    param[0].buffer = &annuncio1;

    param[0].buffer_length = sizeof(annuncio);
    param[1].buffer_type = MYSQL_TYPE_VAR_STRING;
    param[1].buffer = conf.username;
    param[1].buffer_length = strlen(conf.username);

    if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
        finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters for follow annuncement\n", true);
    }
    if (mysql_stmt_execute(prepared_stmt) != 0) {
        print_stmt_error(prepared_stmt, "An error occurred while follow annuncement.");
    }
    else {
        printf("Annuncio seguito correttamente...\n");
    }
    getchar();
    mysql_stmt_close(prepared_stmt);
}

void visualizza_contatto_venditore(MYSQL* conn) {
    MYSQL_STMT* prepared_stmt;
```

```
    MYSQL_BIND param[1];
    char venditore[51];

    printf("\nInserisci il nome del venditore per conoscere i suoi contatti: ");
    getInput(50, venditore);
    if (!setup_prepared_stmt(&prepared_stmt, "call visualizzaContattiVenditore(?)", conn)) {
        finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize contacts board statement\n", false);
    }

    memset(param, 0, sizeof(param));
    param[0].buffer_type = MYSQL_TYPE_VAR_STRING;
    param[0].buffer = venditore;
    param[0].buffer_length = strlen(venditore);


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

    }

    if (mysql_stmt_execute(prepared_stmt) != 0) {
        finish_with_stmt_error(conn, prepared_stmt, "Could not retrieve the contacts board\n", true);
    }
    dump_result_set(conn, prepared_stmt, "\n***Lista dei contatti***");
    printf("\nPress to continue...");
    getchar();
    mysql_stmt_close(prepared_stmt);
}


void visualizza_commenti(MYSQL* conn) {
    MYSQL_STMT* prepared_stmt;
    MYSQL_BIND param[1];
    char options[2] = { '1','2' };
    char op;
```

```
    char annuncio[8];
    int annuncio1;

    printf("\nInserisci l'id dell'annuncio per visualizzare i suoi commenti: ");
    getInput(8, annuncio);
    annuncio1 = atoi(annuncio);

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

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



    param[0].buffer_type = MYSQL_TYPE_LONG;

    param[0].buffer = &annuncio1;
    param[0].buffer_length = sizeof(annuncio);

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

    if (mysql_stmt_execute(prepared_stmt) != 0) {
        finish_with_stmt_error(conn, prepared_stmt, "Could not retrieve annuncement's comments\n",
true);
    }

    dump_result_set(conn, prepared_stmt, "\n***Commenti dell' annuncio***");
    mysql_stmt_close(prepared_stmt);

    printf("\n1)Pubblica un commento\n2)Indietro\n");

    op = multiChoice("\nSeleziona un' opzione", options, 2);

    switch (op) {
    case '1':
```

```
        pubblica_commento(conn, annuncio1);
        break;

    case '2':
        return;
    }
}

void pubblica_commento(MYSQL* conn, int ad) {
    MYSQL_STMT* prepared_stmt;
    MYSQL_BIND param[3];

    char commento[101];
    bool commentonull;

    printf("\nContenuto del commento: ");
    getInput(100, commento);

    if (commento[0] == '\0')
        commentonull = true;
    else
        commentonull = false;

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

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

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

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

    param[2].buffer_type = MYSQL_TYPE_VAR_STRING;
    param[2].buffer = commento;
```

```
    param[2].buffer_length = strlen(commento);
    param[2].is_null = &commentonull;

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

    if (mysql_stmt_execute(prepared_stmt) != 0) {
        print_stmt_error(prepared_stmt, "An error occurred while posting the comment.");
    }
    else {
        printf("Commento correttamente pubblicato...\n");
    }
    getchar();

    mysql_stmt_close(prepared_stmt);
}

void contatta_venditore(MYSQL* conn) {
    MYSQL_STMT* prepared_stmt;
    MYSQL_BIND param[3];
    char content[101];
    char utente2[51];
    bool contentnull;

    printf("\nInserisci il nome dell'utente da contattare: ");
    getInput(50, utente2);
    printf("\nScrivi messaggio: ");
    getInput(100, content);


    if (content[0] == '\0')
        contentnull = true;
    else
        contentnull = false;
```

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

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

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

    param[1].buffer_type = MYSQL_TYPE_VAR_STRING;
    param[1].buffer = utente2;
    param[1].buffer_length = strlen(utente2);

    param[2].buffer_type = MYSQL_TYPE_VAR_STRING;
    param[2].buffer = content;
    param[2].buffer_length = strlen(content);
    param[2].is_null = &contentnull;

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

    if (mysql_stmt_execute(prepared_stmt) != 0) {
        print_stmt_error(prepared_stmt, "An error occurred while sending message.");
    }

    else {
        printf("Messaggio inviato correttamente...\n");
    }
    getchar();
    mysql_stmt_close(prepared_stmt);
}


void visualizza_annunci_seguiti(MYSQL* conn) {
```

```
MYSQL_BIND param[1];
char options[4] = { '1','2','3','4' };
char op;
MYSQL_STMT* prepared_stmt;

if (!setup_prepared_stmt(&prepared_stmt, "call visualizzaAnnunciSeguiti(?)", conn)) {
    finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize  follow annuncement statement\n", false);
}
memset(param, 0, sizeof(param));

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


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

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

dump_result_set(conn, prepared_stmt, "\n***Annunci da te seguiti***");
mysql_stmt_close(prepared_stmt);

    printf("\nCosa vuoi fare?\n");
    printf("1) Visualizza contatti del Venditore\n");
    printf("2) Visualizza commenti\n");
    printf("3) Invia messaggio al venditore\n");
    printf("4) Indietro\n");

    op = multiChoice("Seleziona un' opzione", options, 4);
```

```
        switch (op) {
        case '1':
            visualizza_contatto_venditore_seguiti(conn);
            break;
        case '2':
            visualizza_commenti_seguiti(conn);
            break;
        case '3':
            contatta_venditore_seguiti(conn);
            break;
        case '4':
            return;
        }
}

void visualizza_contatto_venditore_seguiti(MYSQL* conn) {

    MYSQL_STMT* prepared_stmt;
    MYSQL_BIND param[1];
    char venditore[51];

    printf("\nInserisci il nome del venditore per conoscere i suoi contatti: ");
    getInput(50, venditore);

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

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

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

    if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
        finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters for contacts list\n",
```

```
true);
    }

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

    dump_result_set(conn, prepared_stmt, "\n***Lista dei contatti***");
    printf("\nPress to continue...");
    getchar();
    mysql_stmt_close(prepared_stmt);
}

void visualizza_commenti_seguiti(MYSQL* conn) {

    MYSQL_STMT* prepared_stmt;
    MYSQL_BIND param[2];
    char options[2] = { '1','2' };
    char op;
    char annuncio[8];
    int annuncio1;

    printf("\nInserisci l'id dell'annuncio per visualizzare i suoi commenti: ");
    getInput(8, annuncio);
    annuncio1 = atoi(annuncio);

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

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

    param[0].buffer_type = MYSQL_TYPE_LONG;
    param[0].buffer = &annuncio1;
    param[0].buffer_length = sizeof(annuncio);
```

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

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

    if (mysql_stmt_execute(prepared_stmt) != 0) {
        finish_with_stmt_error(conn, prepared_stmt, "Could not retrieve annuncement's comments\n", true);
    }

    dump_result_set(conn, prepared_stmt, "\n***Commenti dell' annuncio***");
    mysql_stmt_close(prepared_stmt);

    printf("\n1)Pubblica un commento\n2)Indietro\n");

    op = multiChoice("\nSeleziona un' opzione", options, 2);

    switch (op) {
    case '1':
        pubblica_commento_seguiti(conn, annuncio1);
        break;
    case '2':
        return;
    }
}

void pubblica_commento_seguiti(MYSQL* conn, int ad) {

    MYSQL_STMT* prepared_stmt;
    MYSQL_BIND param[3];
    char commento[101];
    bool commentonull;

    printf("\nContenuto del commento: ");
    getInput(100, commento);
```

```
   if (commento[0] == '\0')
      commentonull = true;
   else
      commentonull = false;

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

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

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



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

   param[2].buffer_type = MYSQL_TYPE_VAR_STRING;
   param[2].buffer = commento;
   param[2].buffer_length = strlen(commento);
   param[2].is_null = &commentonull;

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

   if (mysql_stmt_execute(prepared_stmt) != 0)       {
      print_stmt_error(prepared_stmt, "An error occurred while posting the comment.");
  }
   else {
      printf("Commento correttamente pubblicato...\n");
   }
```

```
    getchar();
    mysql_stmt_close(prepared_stmt);
}

void contatta_venditore_seguiti(MYSQL* conn) {

    MYSQL_STMT* prepared_stmt;
    MYSQL_BIND param[3];
    char content[101];
    char utente2[51];
    bool contentnull;

    printf("\nInserisci il nome dell'utente da contattare: ");
    getInput(50, utente2);
    printf("\nScrivi messaggio: ");
    getInput(100, content);


    if (content[0] == '\0')
        contentnull = true;
    else
        contentnull = false;

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

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

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

    param[1].buffer_type = MYSQL_TYPE_VAR_STRING;
    param[1].buffer = utente2;
    param[1].buffer_length = strlen(utente2);
```

```
    param[2].buffer_type = MYSQL_TYPE_VAR_STRING;
    param[2].buffer = content;
    param[2].buffer_length = strlen(content);
    param[2].is_null = &contentnull;

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

    if (mysql_stmt_execute(prepared_stmt) != 0) {
        print_stmt_error(prepared_stmt, "An error occurred while sending message.");
    }
    else {
        printf("Messaggio inviato correttamente...\n");
    }
    getchar();
    mysql_stmt_close(prepared_stmt);
}

void visualizza_chat(MYSQL* conn) {

    MYSQL_STMT* prepared_stmt;
    MYSQL_BIND param[2];

    char options[2] = { '1','2' };
    char op;
    int conversazione1 = 0;

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

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

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

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

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

    dump_result_set(conn, prepared_stmt, "\n***Lista delle conversazioni***");
    mysql_stmt_close(prepared_stmt);

    printf("\nCosa vuoi fare?\n");
    printf("1) Visualizza conversazione\n");
    printf("2) Indietro");

    op = multiChoice("\n\nSeleziona un' opzione", options, 2);

    switch (op) {
    case '1':
        start_mex(conn);
        break;
    case '2':
        return;
    }
}

void start_mex(MYSQL* conn) {

    MYSQL_STMT* prepared_stmt;
    MYSQL_BIND param[2];

    char options[2] = { 'y','n' };
    char username[51];
    char op;
    bool nomenull;
```

```
   printf("\nInserisci il nome dell'utente per visualizzare l'intera conversazione: ");
   getInput(50, username);

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

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

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

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

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

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



   dump_result_set(conn, prepared_stmt, "\n***La conversazione***");
   mysql_stmt_close(prepared_stmt);

   op = multiChoice("\n\nVuoi rispondere alla conversazione?", options, 2);

   switch (op) {
   case 'y':
      rispondi_conversazione(conn, username);
      break;
   case 'n':
      return;
```

```
    }
}

void rispondi_conversazione(MYSQL* conn, char* user) {

    MYSQL_STMT* prepared_stmt;
    MYSQL_BIND param[3];
    char content[101];
    bool contentnull;

    printf("\nScrivi messaggio: ");
    getInput(100, content);

    if (content[0] == '\0')
        contentnull = true;
    else
        contentnull = false;

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

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

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

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

    param[2].buffer_type = MYSQL_TYPE_VAR_STRING;
    param[2].buffer = content;
    param[2].buffer_length = strlen(content);
    param[2].is_null = &contentnull;

    if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
        finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters for sending
```

```
message\n", true);
    }

    if (mysql_stmt_execute(prepared_stmt) != 0) {
        print_stmt_error(prepared_stmt, "An error occurred while sending message.");
    }
    else {
        printf("Messaggio inviato correttamente...\n");
    }
    getchar();
    mysql_stmt_close(prepared_stmt);
}

void gestisci_informazioni_personali(MYSQL* conn) {

    char options[3] = { '1','2','3' };
    char op;

    while (true) {
        printf("\033[2J\033[H");
        printf("Cosa vuoi fare?\n");
        printf("1) Visualizza informazioni anagrafiche\n");
        printf("2) Modifica informazioni anagrafiche\n");
        printf("3) Indietro\n");

        op = multiChoice("\nSeleziona un' opzione", options, 3);

        switch (op) {
        case'1':
            visualizza_info_anagrafiche(conn);
            break;
        case '2':
            visualizza_info_anagrafiche(conn);
            modifica_info_anagrafiche(conn);
            break;
        case '3':
            return;
```

```
            getchar();
        }
    }
}

void visualizza_info_anagrafiche(MYSQL* conn) {

    MYSQL_STMT* prepared_stmt;
    MYSQL_BIND param[1];

    if (!setup_prepared_stmt(&prepared_stmt, "call visualizzaInfoAnagrafiche(?)", conn)) {
        finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize show personal information statement\n", false);
    }


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

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

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

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

    dump_result_set(conn, prepared_stmt, "\n***Informazioni anagrafiche***");
    getchar();
    mysql_stmt_close(prepared_stmt);
}

void modifica_info_anagrafiche(MYSQL* conn) {
```

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

    char codFisc[17];
    char nome[51];
    char cognome[51];
   char nascita[11];
    char residenza[51];
    char fatturazione[51];

    MYSQL_TIME* date = NULL;
    char datanull;

modifica:
    printf("\nNuovo nome: ");
    getInput(50, nome);
    printf("Nuovo cognome: ");
    getInput(50, cognome);
    printf("Nuovo codice fiscale: ");
    getInput(17, codFisc);
    printf("Nuova data di nascita (gg-mm-aaaa): ");

    getInput(10, nascita);
    printf("Nuovo indirizzo di residenza: ");
    getInput(50, residenza);
    printf("Nuovo indirizzo di fatturazione: ");
    getInput(50, fatturazione);

    switch (validateParams()) {
    case EXIT_SUCCESS:
        break;
    case EXIT_FAILURE:
        goto modifica;
    }

    if (nascita[0] == '\0')
        datanull = true;
```

```
else {
    datanull = false;
    date = malloc(sizeof(MYSQL_TIME*));
    date->time_type = MYSQL_TIMESTAMP_DATE;

    switch (getBirthDate(nascita, date)) {
    case EXIT_SUCCESS:
        break;
    case EXIT_FAILURE:
        goto modifica;
    }
}

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

    finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize changing personal information statement\n", false);

}


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

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

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

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

param[3].buffer_type = MYSQL_TYPE_DATE;
param[3].buffer = date;
```

```
    param[3].buffer_length = sizeof(date);
    param[3].is_null = &datanull;

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

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

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

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

    }

    if (mysql_stmt_execute(prepared_stmt) != 0) {
        print_stmt_error(prepared_stmt, "An error occurred while changing the informations.");

    }
    else {
        printf("Informazioni personali modificate correttamente...\n");
    }
    mysql_stmt_close(prepared_stmt);
}


void gestisci_contatti(MYSQL* conn) {

    char options[5] = { '1','2','3','4','5' };
    char op;

    while (true) {
        printf("\033[2J\033[H");
```

```
printf("Cosa vuoi fare?\n");
printf("1) Visualizza contatti\n");
printf("2) Crea contatto\n");
printf("3) Aggiorna contatto a preferito\n");
printf("4) Cancella contatto\n");
printf("5) Indietro\n");

op = multiChoice("\nSeleziona un' opzione", options, 5);

switch (op) {
case '1':
   visualizza_contatti(conn);
   break;
case '2':
{
   visualizza_contatti(conn);
   aggiungi_contatto(conn);
}
break;
case '3':
{
   visualizza_contatti(conn);
   aggiorna_preferito(conn);
}
break;
case '4':
{
   visualizza_contatti(conn);
   elimina_contatto(conn);
}
case '5':
   return;
}
getchar();
```

```
    }
}

void visualizza_contatti(MYSQL* conn) {

    MYSQL_STMT* prepared_stmt;
    MYSQL_BIND param[1];

    if (!setup_prepared_stmt(&prepared_stmt, "call visualizzaContatti(?)", conn)) {
        finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize show contacts list statement\n", false);

    }

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

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

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

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

    dump_result_set(conn, prepared_stmt, "\n***Lista contatti***");
    mysql_stmt_close(prepared_stmt);

}

void aggiungi_contatto(MYSQL* conn) {

    MYSQL_STMT* prepared_stmt;
    MYSQL_BIND param[4];

    char options[3] = { '1','2','3' };
    char optionsf[2] = { 'y','n' };
    char op;
```

```
char contatto[51];
bool contattonull;
char tipo[10];
char preferito;

printf("\n\nInserisci le informazioni del tuo contatto\n");
printf("\nRecapito: ");
getInput(51, contatto);
if (contatto[0] == '\0')
   contattonull = true;
else
   contattonull = false;

if (strcmp(contatto, "") == 0)   return;

printf("\nChe tipologia di contatto stai aggiungendo?\n1)Telefono fisso\n2)Cellulare\n3)Email\n");

op = multiChoice("Seleziona un' opzione", options, 3);

switch (op) {
case '1':
   strcpy(tipo, "Fisso");
   break;
case '2':
   strcpy(tipo, "Cellulare");
   break;
case '3':
   strcpy(tipo, "Email");
   break;
}

op = multiChoice("\nVuoi aggiungere il contatto tra i preferiti?", optionsf, 2);

switch (op) {
case 'y':
   preferito = 1;
   break;
case 'n':
```

```
        preferito = 0;
        break;
    }

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

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

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

    param[1].buffer_type = MYSQL_TYPE_VAR_STRING;
    param[1].buffer = contatto;
    param[1].buffer_length = strlen(contatto);
    param[1].is_null = &contattonull;

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

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

    if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
        finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters for contact insertion\n", true);
    }
    if (mysql_stmt_execute(prepared_stmt) != 0) {
        print_stmt_error(prepared_stmt, "An error occurred while adding the contact.");
    }
```

```
    else {
        printf("Contatto correttamente aggiunto...\n");
    }
    mysql_stmt_close(prepared_stmt);
}


void elimina_contatto(MYSQL* conn) {

    MYSQL_STMT* prepared_stmt;
    MYSQL_BIND param[2];
    char toDelete[8];

    printf("\nScrivi il contatto da eliminare: ");
    getInput(8, toDelete);
    if (!setup_prepared_stmt(&prepared_stmt, "call eliminaContatto(?, ?)", conn)) {
        finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize delete contacts statement\n",
false);
    }

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

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

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

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

    if (mysql_stmt_execute(prepared_stmt) != 0) {
        print_stmt_error(prepared_stmt, "An error occurred while  delete contacts.");
    }
```

```
    else {
        printf("Contatto eliminato correttamente...\n");
    }
    getchar();
    mysql_stmt_close(prepared_stmt);
}

void aggiorna_preferito(MYSQL* conn) {

    MYSQL_STMT* prepared_stmt;
    MYSQL_BIND param[2];
   char newFav[51];

  aggiorna:

    printf("\nInserisci il contatto da rendere preferito: ");
    getInput(50, newFav);

    switch (validateParams()) {
    case EXIT_SUCCESS:
        break;
    case EXIT_FAILURE:
        goto aggiorna;
    }

    if (!setup_prepared_stmt(&prepared_stmt, "call aggiornaContattoPreferito(?, ?)", conn)) {
        finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize update favourite contacts statement\n", false);
    }

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

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

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

```
   if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
      finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters for update favourite contacts\n", true);
   }
   if (mysql_stmt_execute(prepared_stmt) != 0) {
      print_stmt_error(prepared_stmt, "An error occurred while updating the favourite contact.");
   }
   else {
      printf("Contatto aggiornato correttamente a preferito...\n");
   }
   mysql_stmt_close(prepared_stmt);
}
```

---

gestore.c

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



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


	if (!setup_prepared_stmt(&prepared_stmt, "call visualizzaCategoria()", conn)) {
		finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize categories list statement\n", false);
	}

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

	dump_result_set(conn, prepared_stmt, "\n***Lista categoria***");
	mysql_stmt_close(prepared_stmt);
	getchar();
```

```
}


static void riscuoti_percentuale(MYSQL* conn) {
	MYSQL_STMT* prepared_stmt;
	MYSQL_BIND param;

	char username[51];

	if (!setup_prepared_stmt(&prepared_stmt, "call visualizzaReport()", conn)) {
		finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize categories list
statement\n", false);
	}

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

	dump_result_set(conn, prepared_stmt, "\n***Report***");
	mysql_stmt_close(prepared_stmt);

	printf("\nUsername dell'utente per riscuotere la percentuale: ");
	getInput(50, username, false);

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

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

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

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

	if (mysql_stmt_execute(prepared_stmt) != 0) {
		print_stmt_error(prepared_stmt, "An error occurred while collecting percentage.");
	}
	else {
		printf("Riscossione avvenuta con successo...\n");
	}
	getchar();
	mysql_stmt_close(prepared_stmt);
}
```

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

	char nome[51];
	char categoriapadre[51];
	bool categoriapadrenull;
	bool categorianull;

   visualizzaCategoria(conn);

	again:
	printf("\nNome della categoria: ");
	getInput(50, nome, false);

	printf("Nome categoria padre: ");
	getInput(50, categoriapadre, false);

	if (nome[0] == '\0')
		categorianull = true;
	else
		categorianull = false;

	if (categoriapadre[0] == '\0')
		categoriapadrenull = true;
	else
		categoriapadrenull = false;

	switch (validateParams()) {
	case EXIT_SUCCESS:
		break;
	case EXIT_FAILURE:
		goto again;
	}

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

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

	param[0].buffer_type = MYSQL_TYPE_VAR_STRING;
	param[0].buffer = nome;
	param[0].buffer_length = strlen(nome);
	param[0].is_null = &categorianull;

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

```
	param[1].buffer = categoriapadre;
	param[1].buffer_length = strlen(categoriapadre);
	param[1].is_null = &categoriapadrenull;

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

	if (mysql_stmt_execute(prepared_stmt) != 0) {
		print_stmt_error(prepared_stmt, "\nAn error occurred while registering category.");
	}
	else {
		printf("Categoria creata correttamente...\n");
	}
	getchar();
	mysql_stmt_close(prepared_stmt);
}


void gestorePage(MYSQL* conn)
{
	char options[4] = { '1','2', '3','4'};
	char op;
	while (true) {
	start:
		printf("\033[2J\033[H");
		printf("Hai effetuato l'accesso come Gestore\n\n");
		printf("Cosa vuoi fare?\n");
		printf("1) Visualizza categoria\n");
		printf("2) Crea categoria\n");
		printf("3) Riscuoti percentuale\n");
		printf("4) Esci\n");

		op = multiChoice("Seleziona un'opzione", options, 4);

		switch (op) {
		case'1':
			visualizzaCategoria(conn);
			goto start;
		case '2':
			crea_categoria(conn);
			goto start;
		case '3':
			riscuoti_percentuale(conn);
			goto start;
		case '4':
			return;

		default:
```

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


void run_as_gestore(MYSQL* conn) {
        printf("Switching to administrative role...\n");

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

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

        gestorePage(conn);

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

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

---

inout.c

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

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

        // Acquisisce da tastiera al più lung - 1 caratteri
```

```
	for (i = 0; i < lung; i++) {
		fread(&c, sizeof(char), 1, stdin);
		if (c == '\n') {
			stringa[i] = '\0';
			break;
		}
		else
			stringa[i] = c;
	}

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

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

	return stringa;
}

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

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

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

		char c;
		getInput(1, &c);

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

```
}

// Accepts the inputted parameters
int validateParams() {
	char choices[2] = { 's', 'n' };
	char res;
	res = multiChoice("\nSei sicuro di voler confermare questi parametri?", choices, 2);
	switch (res) {
	case 's':
		return EXIT_SUCCESS;
	case 'n':
		return EXIT_FAILURE;
	}
}

// Write the birth date from a string in the MYSQL_TIME struct
int getBirthDate(char* dateText, MYSQL_TIME* date) {

	char* end;
	unsigned int d[3];

	// Extracts the day, month and year
	for (int i = 0; i < 3; i++) {
		d[i] = strtoul(dateText, &end, 10);
		dateText = end + 1;
		if (d[i] == 0)
			return EXIT_FAILURE;
	}

	// Check for out of bounds values
	if (d[0] >= 32 || d[0] == 0)
		return EXIT_FAILURE;
	if (d[1] >= 13 || d[1] == 0)
		return EXIT_FAILURE;
	if (d[2] >= 10000 || d[2] == 0)
		return EXIT_FAILURE;

	date->day = d[0];
	date->month = d[1];
	date->year = d[2];

	return EXIT_SUCCESS;
}

// Write the expiration date of a credit card from a string in the MYSQL_TIME struct
int getExpirationDate(char* dateText, MYSQL_TIME* date) {

	char* end;
	unsigned int d[2];
```

```
	// Extracts the day, month and year
	for (int i = 0; i < 2; i++) {
		d[i] = strtoul(dateText, &end, 10);
		dateText = end + 1;
		if (d[i] == 0)
			return EXIT_FAILURE;
	}

	if (d[0] >= 13 || d[0] == 0)
		return EXIT_FAILURE;
	if (d[1] >= 10000 || d[1] == 0)
		return EXIT_FAILURE;

	date->day = 31;
	date->month = d[0];
	if (d[1] >= 2000)
		date->year = d[1];
	else
		date->year = d[1] + 2000;

	// Sets the maximum number of days based on the month
	switch (d[0]) {
	case 1:
	case 3:
	case 5:
	case 7:
	case 8:
	case 10:
	case 12:
		date->day = 31;
	case 4:
	case 6:
	case 9:
	case 11:
		date->day = 31;
	case 2:
		if (date->year % 4 == 0)
			date->day = 29;
		else
			date->day = 28;
	}

	return EXIT_SUCCESS;
}
```

---

parse.c

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

#define BUFF_SIZE 4096

char* strndup(const char* s, size_t n) {
	char* p;
	size_t n1;

	for (n1 = 0; n1 < n && s[n1] != '\0'; n1++)
		continue;
	p = malloc(n + 1);
	if (p != NULL) {
		memcpy(p, s, n1);
		p[n1] = '\0';
	}
	return p;
}

// The final config struct will point into this

static char config[BUFF_SIZE];
/**
 * JSON type identifier. Basic types are:
 *	o Object
 *	o Array
 *	o String
 *	o Other primitive: number, boolean (true/false) or null
 */
```

```
typedef enum {
	JSMN_UNDEFINED = 0,
	JSMN_OBJECT = 1,
	JSMN_ARRAY = 2,
	JSMN_STRING = 3,
	JSMN_PRIMITIVE = 4
} jsmntype_t;

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

/**
 * JSON token description.
 * type		type (object, array, string etc.)
 * start start position in JSON data string
 * end		end position in JSON data string
 */

typedef struct {
	jsmntype_t type;
	int start;
	int end;
	int size;
#ifdef JSMN_PARENT_LINKS
	int parent;
#endif
} jsmntok_t;

/**
```

```
 * JSON parser. Contains an array of token blocks available. Also stores
 * the string being parsed now and current position in that string
 */

typedef struct {
	unsigned int pos; /* offset in the JSON string */
	unsigned int toknext; /* next token to allocate */
	int toksuper; /* superior token node, e.g parent object or array */
} jsmn_parser;

/**
 * Allocates a fresh unused token from the token pool.
 */

static jsmntok_t* jsmn_alloc_token(jsmn_parser* parser, jsmntok_t* tokens, size_t num_tokens) {
	jsmntok_t* tok;
	if (parser->toknext >= num_tokens) {
		return NULL;
	}
	tok = &tokens[parser->toknext++];
	tok->start = tok->end = -1;
	tok->size = 0;
#ifdef JSMN_PARENT_LINKS
	tok->parent = -1;
#endif
	return tok;
}

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

```
	token->size = 0;
}

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

	start = parser->pos;


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

#ifdef JSMN_STRICT
	/* In strict mode primitive must be followed by a comma/object/array */
	parser->pos = start;
	return JSMN_ERROR_PART;
#endif
```

```
found:
	if (tokens == NULL) {
		parser->pos--;
		return 0;
	}

	token = jsmn_alloc_token(parser, tokens, num_tokens);
	if (token == NULL) {
		parser->pos = start;
		return JSMN_ERROR_NOMEM;
	}
	jsmn_fill_token(token, JSMN_PRIMITIVE, start, parser->pos);
#ifdef JSMN_PARENT_LINKS

	token->parent = parser->toksuper;
#endif
	parser->pos--;
	return 0;
}

/**
 * Fills next token with JSON string.
 */

static int jsmn_parse_string(jsmn_parser* parser, const char* js,
	size_t len, jsmntok_t* tokens, size_t num_tokens) {
	jsmntok_t* token;

	int start = parser->pos;
	parser->pos++;

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

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

```
                return 0;
            }

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

        }

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

                                (js[parser->pos] >= 65 && js[parser->pos] <= 70) || /*
A-F */
```

```
                                        (js[parser->pos] >= 97 && js[parser->pos] <= 102))) {
/* a-f */
                                        parser->pos = start;
                                        return JSMN_ERROR_INVAL;
                                }
                                        parser->pos++;
                                }
                                parser->pos--;
                                break;
                                /* Unexpected symbol */
                        default:
                                parser->pos = start;
                                return JSMN_ERROR_INVAL;
                        }
                }
        }
        parser->pos = start;
        return JSMN_ERROR_PART;
}
/**
 * Parse JSON string and fill tokens.
 */
static int jsmn_parse(jsmn_parser* parser, const char* js, size_t len, jsmntok_t* tokens, unsigned int
num_tokens) {
        int r;
        int i;
        jsmntok_t* token;
        int count = parser->toknext;
```

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

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

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

#ifdef JSMN_PARENT_LINKS
			if (parser->toknext < 1) {
				return JSMN_ERROR_INVAL;
			}
			token = &tokens[parser->toknext - 1];
```

```
	for (;;) {
		if (token->start != -1 && token->end == -1) {
			if (token->type != type) {
				return JSMN_ERROR_INVAL;
			}
			token->end = parser->pos + 1;
			parser->toksuper = token->parent;
			break;
		}
		if (token->parent == -1) {
			if (token->type != type || parser->toksuper == -1) {
				return JSMN_ERROR_INVAL;
			}
			break;
		}
		token = &tokens[token->parent];
	}
#else
	for (i = parser->toknext - 1; i >= 0; i--) {
		token = &tokens[i];
		if (token->start != -1 && token->end == -1) {
			if (token->type != type) {
				return JSMN_ERROR_INVAL;
			}
			parser->toksuper = -1;
			token->end = parser->pos + 1;
			break;
		}
	}
	/* Error if unmatched closing bracket */
	if (i == -1) return JSMN_ERROR_INVAL;
	for (; i >= 0; i--) {
```

```
                        token = &tokens[i];
                        if (token->start != -1 && token->end == -1) {
                                parser->toksuper = i;
                                break;
                        }
                }
#endif
                break;
        case '\"':
                r = jsmn_parse_string(parser, js, len, tokens, num_tokens);
                if (r < 0) return r;
                count++;
                if (parser->toksuper != -1 && tokens != NULL)
                        tokens[parser->toksuper].size++;
                break;
        case '\t': case '\r': case '\n': case ' ':
                break;
        case ':':
                parser->toksuper = parser->toknext - 1;
                break;
        case ',':
                if (tokens != NULL && parser->toksuper != -1 &&
                        tokens[parser->toksuper].type != JSMN_ARRAY &&
        tokens[parser->toksuper].type != JSMN_OBJECT) {
#ifdef JSMN_PARENT_LINKS
                        parser->toksuper = tokens[parser->toksuper].parent;
#else
                        for (i = parser->toknext - 1; i >= 0; i--) {
                                if (tokens[i].type == JSMN_ARRAY || tokens[i].type ==
JSMN_OBJECT) {
                                        if (tokens[i].start != -1 && tokens[i].end == -1) {
                                                parser->toksuper = i;
```

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

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

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


        return count;

}

/**
 * Creates a new parser based over a given  buffer with an array of tokens
 * available.
 */

static void jsmn_init(jsmn_parser* parser) {
        parser->pos = 0;
        parser->toknext = 0;
        parser->toksuper = -1;
}

static int jsoneq(const char* json, jsmntok_t* tok, const char* s)

{
```

```
        if (tok->type == JSMN_STRING
                && (int)strlen(s) == tok->end - tok->start
                && strncmp(json + tok->start, s, tok->end - tok->start) == 0) {
                return 0;
        }
        return -1;
}

static size_t load_file(char* filename)
{
        FILE* f = fopen(filename, "rb");
        if (f == NULL) {
                fprintf(stderr, "Unable to open file %s\n", filename);
                exit(1);
        }
        fseek(f, 0, SEEK_END);
        size_t fsize = ftell(f);
        fseek(f, 0, SEEK_SET);  //same as rewind(f);
        if (fsize >= BUFF_SIZE) {
                fprintf(stderr, "Configuration file too large\n");
                abort();
        }
        fread(config, fsize, 1, f);
        fclose(f);
        config[fsize] = 0;
        return fsize;
}

int parse_config(char* path, struct configuration* conf)
```

```
{
        int i;
        int r;
        jsmn_parser p;
        jsmntok_t t[128]; /* We expect no more than 128 tokens */

        load_file(path);

        jsmn_init(&p);
        r = jsmn_parse(&p, config, strlen(config), t, sizeof(t) / sizeof(t[0]));

        if (r < 0) {
                printf("Failed to parse JSON: %d\n", r);
                return 0;
        }

        /* Assume the top-level element is an object */

        if (r < 1 || t[0].type != JSMN_OBJECT) {
                printf("Object expected\n");
                return 0;
        }

        /* Loop over all keys of the root object */

        for (i = 1; i < r; i++) {
                if (jsoneq(config, &t[i], "host") == 0) {
                        /* We may use strndup() to fetch string value */
                        conf->host = strndup(config + t[i + 1].start, t[i + 1].end - t[i + 1].start);
                    i++;
                }

                else if (jsoneq(config, &t[i], "username") == 0) {
                        conf->db_username = strndup(config + t[i + 1].start, t[i + 1].end - t[i +
1].start);
                        i++;
                }
                else if (jsoneq(config, &t[i], "password") == 0) {
```

```
                conf->db_password = strndup(config + t[i + 1].start, t[i + 1].end - t[i + 1].start);
                i++;
            }
            else if (jsoneq(config, &t[i], "port") == 0) {
                conf->port = strtol(config + t[i + 1].start, NULL, 10);
                i++;
            }
            else if (jsoneq(config, &t[i], "database") == 0) {
                conf->database = strndup(config + t[i + 1].start, t[i + 1].end - t[i + 1].start);
                i++;
            }
            else {
                printf("Unexpected key: %.*s\n", t[i].end - t[i].start, config + t[i].start);
            }
        }
        return 1;
}
```

---

utils.c

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

#include "defines.h"

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

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

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

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

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

	mysql_stmt_attr_set(*stmt, STMT_ATTR_UPDATE_MAX_LENGTH, &update_length);

	return true;
}

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

void finish_with_stmt_error(MYSQL* conn, MYSQL_STMT* stmt, char* message, bool
close_stmt)
{
	print_stmt_error(stmt, message);
	if (close_stmt)		mysql_stmt_close(stmt);
	mysql_close(conn);
	exit(EXIT_FAILURE);
```

```
}

static void print_dashes(MYSQL_RES* res_set)
{
	MYSQL_FIELD* field;
	unsigned int i, j;

	mysql_field_seek(res_set, 0);
	putchar('+');
	for (i = 0; i < mysql_num_fields(res_set); i++) {
		field = mysql_fetch_field(res_set);
		for (j = 0; j < field->max_length + 2; j++)
			putchar('-');
		putchar('+');
	}
	putchar('\n');
}

static void dump_result_set_header(MYSQL_RES* res_set)
{
	MYSQL_FIELD* field;
	unsigned long col_len;
	unsigned int i;

	/* determine column display widths -- requires result set to be */
	/* generated with mysql_store_result(), not mysql_use_result() */

	mysql_field_seek(res_set, 0);

	for (i = 0; i < mysql_num_fields(res_set); i++) {
		field = mysql_fetch_field(res_set);
		col_len = strlen(field->name);

		if (col_len < field->max_length)
			col_len = field->max_length;
		if (col_len < 4 && !IS_NOT_NULL(field->flags))
			col_len = 4; /* 4 = length of the word "NULL" */
		field->max_length = col_len; /* reset column info */
	}

	print_dashes(res_set);
	putchar('|');
	mysql_field_seek(res_set, 0);
	for (i = 0; i < mysql_num_fields(res_set); i++) {
		field = mysql_fetch_field(res_set);
		printf(" %-*s |", (int)field->max_length, field->name);
	}
	putchar('\n');

	print_dashes(res_set);
```

```
}

void dump_result_set(MYSQL* conn, MYSQL_STMT* stmt, char* title)
{
	int status = 0;
	do {
		int i;
		int num_fields;	/* number of columns in result */
		MYSQL_FIELD* fields;  /* for result set metadata */
		MYSQL_BIND* rs_bind;  /* for output buffers */
		MYSQL_RES* rs_metadata;
		MYSQL_TIME* date;
		size_t attr_size;

		/* Prefetch the whole result set. This in conjunction with
		 * STMT_ATTR_UPDATE_MAX_LENGTH set in `setup_prepared_stmt`
		 * updates the result set metadata which are fetched in this
		 * function, to allow to compute the actual max length of
		 * the columns.
		 */
		if (mysql_stmt_store_result(stmt)) {
			fprintf(stderr, " mysql_stmt_execute(), 1 failed\n");
			fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
			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:
			case MYSQL_TYPE_LONGLONG:
				attr_size = sizeof(int);
				break;
			default:
				attr_size = fields[i].max_length;
				break;
			}

			// Setup the binding for the current parameter
			rs_bind[i].buffer_type = fields[i].type;
			rs_bind[i].buffer = malloc(attr_size + 1);
			rs_bind[i].buffer_length = attr_size + 1;
			rs_bind[i].is_null = malloc(sizeof(bool*));

			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 (*(bool*)rs_bind[i].is_null) {
			printf(" %-*s |", (int)fields[i].max_length, "NULL");
			continue;
		}

		switch (rs_bind[i].buffer_type) {

		case MYSQL_TYPE_DATETIME:
		case MYSQL_TYPE_TIMESTAMP:
			date = (MYSQL_TIME*)rs_bind[i].buffer;
			printf(" %02d-%02d-%d %02d:%02d:%02d.%-*d |",
date->day, date->month, date->year, date->hour, date->minute, date->second,
(int)fields[i].max_length - 20, (date->second_part) / 1000);
			break;

		case MYSQL_TYPE_DATE:
			date = (MYSQL_TIME*)rs_bind[i].buffer;
			printf(" %02d-%02d-%-*d |", date->day, date->month,
(int)fields[i].max_length, date->year);
			break;

		case MYSQL_TYPE_TIME:
			date = (MYSQL_TIME*)rs_bind[i].buffer;
			printf(" %02d:%02d:%02d.%-*d |", date->hour, date-
>minute, date->second, (int)fields[i].max_length - 9, (date->second_part) / 1000);
			break;

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

		case MYSQL_TYPE_FLOAT:
		case MYSQL_TYPE_DOUBLE:
			printf(" %-.02f |", *(float*)rs_bind[i].buffer);
			break;
```

```
                                        case MYSQL_TYPE_LONG:
                                        case MYSQL_TYPE_SHORT:
                                                printf(" %-*d |", (int)fields[i].max_length + 3,
*(int*)rs_bind[i].buffer);
                                                break;
                                        case MYSQL_TYPE_TINY:
                                                printf(" %-*hhd |", (int)fields[i].max_length + 2,
*(int*)rs_bind[i].buffer);
                                                break;

                                        default:
                                                printf("ERROR: Unhandled type (%d)\n",
rs_bind[i].buffer_type);
                                                abort();
                                        }
                                }
                                putchar('\n');
                                print_dashes(rs_metadata);
                        }

                        mysql_free_result(rs_metadata); /* free metadata */

                        free(rs_bind);
                }

                status = mysql_stmt_next_result(stmt);
                if (status > 0)
                        finish_with_stmt_error(conn, stmt, "An error occurred when fetching a new
result\n", true);
        } while (status == 0);
}
```

---

defines.h

```
#pragma once

#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;

/*---------------------------------UTENTE---------------------------------*/
extern void run_as_utente(MYSQL* conn);
extern void utentePage(MYSQL* conn);
extern void aggiorna_utente_certificato(MYSQL* conn);

/*----------------------UTILS UTENTE / UTENTE CERTIFICATO---------------*/
extern void visualizza_bacheca_pubblica(MYSQL* conn);
extern void segui_annuncio(MYSQL* conn);
extern void visualizza_contatto_venditore(MYSQL* conn);
extern void visualizza_commenti(MYSQL* conn);
extern void pubblica_commento(MYSQL* conn, int ad);
extern void contatta_venditore(MYSQL* conn);
extern void visualizza_annunci_seguiti(MYSQL* conn);
extern void visualizza_contatto_venditore_seguiti(MYSQL* conn);
extern void visualizza_commenti_seguiti(MYSQL* conn);
extern void pubblica_commento_seguiti(MYSQL* conn, int ad);
extern void contatta_venditore_seguiti(MYSQL* conn);
extern void visualizza_chat(MYSQL* conn);
extern void start_mex(MYSQL* conn);
extern void rispondi_conversazione(MYSQL* conn, char* user);
extern void gestisci_informazioni_personali(MYSQL* conn);
extern void visualizza_info_anagrafiche(MYSQL* conn);
extern void modifica_info_anagrafiche(MYSQL* conn);
extern void gestisci_contatti(MYSQL* conn);
extern void visualizza_contatti(MYSQL* conn);
extern void aggiungi_contatto(MYSQL* conn);
extern void elimina_contatto(MYSQL* conn);
extern void aggiorna_preferito(MYSQL* conn);

/*--------------------UTENTE CERTIFICATO----------------------*/
extern void creaAnnuncio(MYSQL* conn);
extern void rimuovi_annuncio(MYSQL* conn);
extern void visualizza_miei_annunci(MYSQL* conn);
extern void utenteCertificatoPage(MYSQL* conn);
extern void run_as_utenteCertificato(MYSQL* conn);
extern void segna_venduto(MYSQL* conn);
extern void visualizza_annunci(MYSQL* conn);

/*--------------------GESTORE-------------------------------*/
extern void visualizzaCategoria(MYSQL* conn);
extern void riscuoti_percentuale(MYSQL* conn);
extern void run_as_gestore(MYSQL* conn);
```

```
extern void gestorePage(MYSQL* conn);
extern void crea_categoria(MYSQL* conn);

/*----------------------INOUT---------------------------------*/
extern char* getInput(unsigned int lung, char* stringa);
extern char multiChoice(char* domanda, char choices[], int num);
extern int validateParams();
extern int getBirthDate(char* dateText, MYSQL_TIME* date);
extern int getExpirationDate(char* dateText, MYSQL_TIME* date);

/*-------------------PARSE----------------------*/
extern char* strndup(const char* s, size_t n);
extern int parse_config(char* path, struct configuration* conf);

/*---------------------------UTILS---------------------------*/
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);
```