Basi di Dati e Conoscenza

Progetto A.A. 2020/2021

BACHECA ELETTRONICA DI ANNUNCI

0267412

Anuar Elio Magliari

Indice

L’assegnazione della tesina può essere effettuata online, visitando il sito https://www.pellegrini.tk/progetti/ ed inserendo i propri dati. Per qualsiasi problema, contattare il docente via email all’indirizzo a.pellegrini@ing.uniroma2.it

NOTA:

In caso di necessità c’è un backup del progetto nel mio account github https://github.com/Monello1299/Basi-di-Dati

1. Descrizione del Minimondo

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

1. Identificazione dei termini ambigui e correzioni possibili

Linea	Termine	Nuovo termine	Motivo correzione
2	Utente del sistema	Utente	Un utente che interagisce con il sistema è anche utente di sistema
4	Domanda	Commento	Una domanda è identificato come un commento
10	Gestore del sistema	Amministratore	Il termine “sistema” è ridondante
10	Gerarchia di categorie	Categoria	Il termine “gerarchia” è ridondante, in quanto una suddivisione degli annunci in categorie è implicitamente una gerarchia
24	Mostrare	Vedere	Un utente non può mostrare un annuncio, bensì è il sistema a farlo
28	Mostrare	Vedere	Un utente non può mostrare lo storico delle conversazioni, bensì è il sistema a farlo

Specifica disambiguata

Si vuole realizzare un sistema informativo per la gestione di una bacheca elettronica di annunci. Tale bacheca permette agli utenti 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 commenti (in maniera pubblica) sull’oggetto.
Un utente 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.
Gli amministratori possono creare delle 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, 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
- Vedere gli annunci che ha aggiunto tra i preferiti, 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 aggiunti dagli utenti, portando l'indicazione del suo stato)
- Inviare messaggi agli altri utenti e vedere lo storico delle sue conversazioni, anche con la possibilità di rispondere ad una conversazione specifica
- Inserire commenti agli annunci ancora attivi.

Gli amministratori prendono una percentuale su ciascun oggetto indicato come venduto. Per questo motivo, essi possono generare un report indicante per ciascun utente 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.

3. Glossario dei Termini

Realizzare un dizionario dei termini, compilando la tabella qui sotto, a partire dalle specifiche precedentemente disambiguate

5

Termine	Descrizione	Sinonimi	Collegamenti
Utente	Persona che si interfaccia con il database		
Annuncio	Un inserzione nel sistema riguardante un oggetto	Oggetto	Utente, Categoria
Categoria	Permette di distingue la natura dell'annuncio da altri tipi di annunci		Annuncio
Messaggio	Tipo di comunicazione indiretta tra utenti		Conversazione
Report	Fattura riguardante gli annunci classificati come venduti per un utente		Utente
Conversazione	Interazione tramite messaggi tra utenti		Utente, Messaggio

Storico Conversazione	Traccia tutte le conversazioni dell'utente		Utente, Conversazione
Informazione anagrafica	Contiene tutti i dati identificativi necessari dell'utente		Utente
Carta di Credito	Contiene tutti i dati relativi alla carta di credito		Utente

4.

5. Raggruppamento dei requisiti in insiemi omogenei

Annuncio
Un utente, per creare un annuncio, seleziona una categoria e scrive una descrizione dell'oggetto. Eventualmente, può decidere di caricare una foto dell'oggetto. Un annuncio inserito in bacheca classificato come venduto non viene più visualizzato nella bacheca pubblica. Un utente può "seguire" gli annunci.
Utente
Un utente 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. Per creare un annuncio, un utente deve necessariamente aver inserito i dati della sua carta di credito. Un utente, 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. Il report riporta anche le informazioni sulla carta di credito dell'utente.
Categoria
Gli amministratori possono creare delle categorie per gli annunci. Un utente, per creare un annuncio, seleziona una categoria.
Messaggio
Un utente, 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. In generale, un utente può inviare messaggi agli altri utenti.
Report
Gli amministratori possono generare un report indicante per ciascun utente quanti annunci sono stati contrassegnati come venduti. Il report riporta anche le informazioni sulla carta di credito dell'utente.
Conversazioni
In generale, un utente può inviare messaggi agli altri utenti e mostrare lo storico delle sue

conversazioni, anche con la possibilità di rispondere ad una conversazione specifica.

Storico Conversazione

In generale un utente può: Inviare messaggi agli altri utenti e vedere lo storico delle sue conversazioni, anche con la possibilità di rispondere ad una conversazione specifica.

Informazione anagrafica

Un utente 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.

Carta di credito

Un utente si registra inserendo i dati relativi alla sua carta di credito. I dati della carta di credito non sono obbligatori.

Per creare un annuncio, un utente deve necessariamente aver inserito i dati della sua carta di credito.

3. Progettazione concettuale

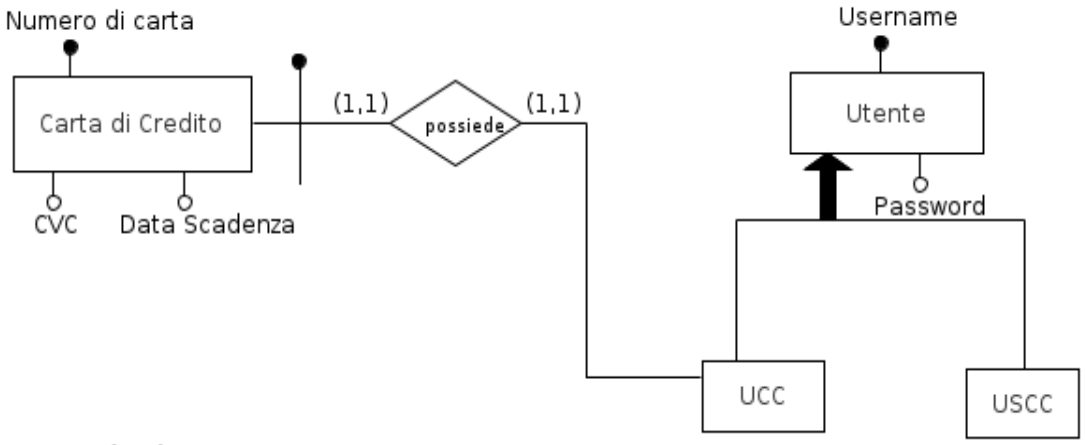
6. Costruzione dello schema E-R

La progettazione del modello ER si è basata su un approccio bottom up, realizzando dapprima tutti i concetti chiave della specifica, approfondendoli con i suoi attributi, e in seguito collegare tutti i concetti per ottenere un unico modello. In seguito sono riportati i passi seguiti durante la progettazione.

- **Utente**

La specifica definisce chiaramente due tipi di utenti che utilizzano il sistema. Un utente UCC (Utente con Carta di Credito) ed un utente USCC (Utente Senza Carta di Credito). Entrambi condividono la caratteristica di possedere un username ed una password, ma l'utente UCC oltre a questo possiede anche una carta di credito, dove l'entità "Carta di Credito" riporta tutte le sue informazioni che dipendono unicamente dall'utente.

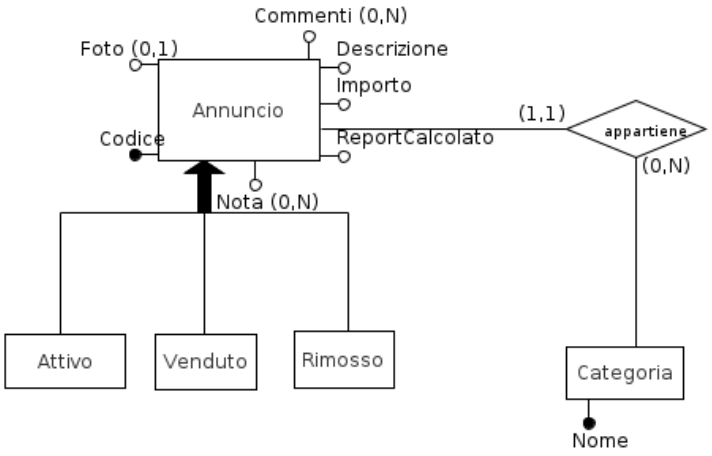
Nota: Da qui in poi per semplicità entrambi gli utenti verranno indicati tramite i loro acronimi



- **Annuncio**

Altro elemento fondamentale della specifica è l'entità "Annuncio". Rappresenta il concetto chiave del sistema. In essa possiamo includere una foto rappresentativa dell'oggetto messo in vendita (come specificato, il suo inserimento è facoltativo), un importo (molto importante per l'entità Report che vedremo in seguito), un codice identificativo unico nel sistema, commenti pubblici inerenti all'oggetto in vendita e note pubbliche inerenti all'oggetto in vendita. Inoltre, nel momento

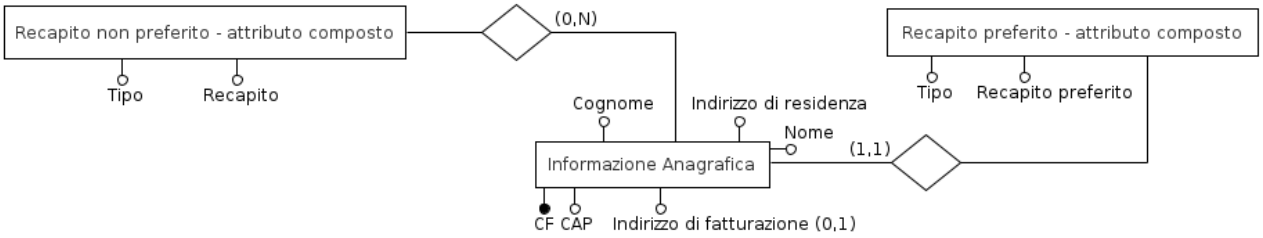
dell'inserimento di un nuovo annuncio, l'utente UCC specificherà una e una sola categoria esistente nel database, inserita precedentemente da un amministratore.



Inoltre si è deciso di differenziare l'annuncio in tre tipi per diversificare il proprio stato attuale e l'attributo "ReportCalcolato" servirà al sistema per sapere se quell'annuncio è già stato calcolato.

- **Informazione Anagrafica**

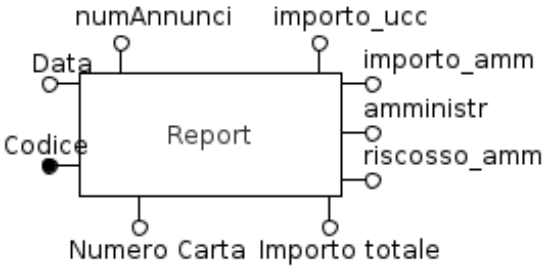
L'entità "Informazione Anagrafica" riporta tutte le informazioni richieste dell'utente (che sia UCC o USCC). I recapiti possono essere di due tipi: "Preferito" e "Non Preferito".
Per un recapito non preferito è possibile inserire più di un contatto.



Nota: A causa di alcune limitazioni del tool usato per costruire il modello, i due recapiti sono due attributi composti, come riportato nell'immagine.

- **Report**

L'entità "Report" è una sorta di fattura riguardante un annuncio indicato come venduto. Riporta tutte le informazioni necessarie per il suo uso.

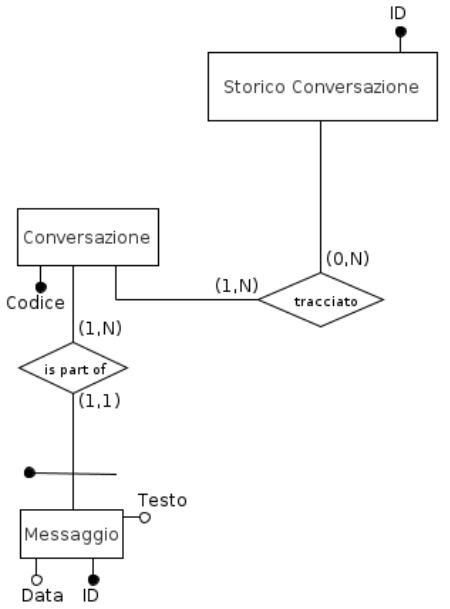


Gli attributi sono stati scelti con i seguenti criteri:

- **Codice**
  - Identificativo unico nel sistema.
- **Data**
  - Riporta la data di creazione del report.
- **Numero Carta**
  - Riporta il numero della carta di credito dell'utente UCC, necessario per la riscossione del report.
- **Importo totale**
  - Riporta l'importo totale senza l'applicazione della percentuale per gli amministratori.
- **numAnnunci**
  - Numero degli annunci a cui il report si riferisce.
- **importo_ucc**
  - Importo che sarà destinato all'utente UCC, dopo l'applicazione della percentuale.
- **importo_amm**
  - Importo che sarà destinato agli amministratori, dopo l'applicazione della percentuale.
- **amministr**
  - Username dell'amministratore che si è occupato della generazione del report
- **riscosso_amm**
  - Si è pensato di aggiungere questo attributo per indicare nel sistema che la percentuale dell'amministratore è stato riscosso, essendo loro i gestori del sistema, questo dato può essere utile per mantenere un ordine nel database.

- **Conversazione**

  Questo riporta un po' tutto il mini modello riguardante la gestione della comunicazione tra utenti. Per sviluppare questa parte della specifica si è voluto creare un entità "Conversazione" contenente tutte le conversazioni tra utenti con ognuno un codice univoco per essere identificati nel sistema. Ovviamente una conversazione contiene messaggi, quindi si è deciso di usare il pattern "is part of" per indicare che una conversazione è costituita da messaggi. Ogni messaggio ha un suo codice identificativo e dipende unicamente dall'entità in relazione.

  Inoltre nel momento della creazione di una conversazione essa verrà tracciata nello storico dei due utenti che stanno comunicando.

Nota: A causa di alcune limitazioni del tool usato per costruire il modello, la cardinalità (1, N) con la relazione tracciato è (2, 2). In quanto quella conversazione è tracciata da due e solo due utenti.

Integrazione finale

Infine si è unito ogni "mini" modello visti precedentemente con le seguenti relazioni.

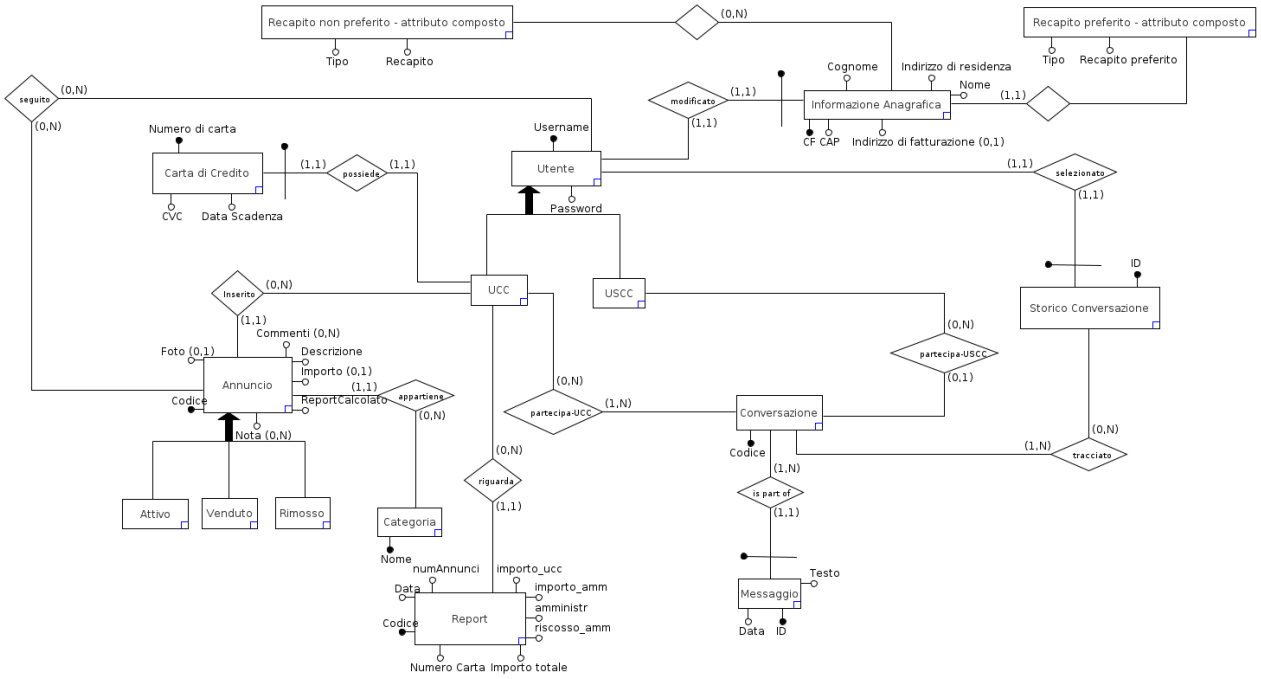
1. **Inserito**
   - Tiene traccia di ogni utente UCC (l'unico che può pubblicare un annuncio) che ha inserito il suo annuncio
2. **Seguito**
   - Tiene traccia degli annunci seguiti dagli utenti
3. **Modificato**
   - Tiene traccia delle informazioni anagrafiche di ogni utente
4. **Riguarda**
   - Correla un report ad un utente
5. **Selezionato**
   - Tiene traccia del possesso di uno storico conversazione di ogni utente
6. **Partecipa-UCC**
   - Tiene traccia della partecipazione dell'utente UCC a determinate conversazioni

7. **Partecipa-USCC**
    - Tiene traccia della partecipazione, in questo caso, dell'utente USCC a determinate conversazioni

Nota: A causa di alcune limitazioni del tool usato per costruire il modello, la cardinalità (1, N) tra "Conversazione" e "Partecipa-UCC" è (1, 2), in quanto le uniche conversazione previste sono:

- *UCC - UCC*
- *UCC – USCC*

Quindi non sono previste comunicazioni USCC – USCC vista la sua poca importanza a far comunicare due utenti che non hanno lo scopo di offrire un bene nel sistema. Quindi ciò avrebbe portato un aumento del traffico di informazioni poco "utili", di conseguenza un degrado delle prestazioni.



7. Regole aziendali

8. Regole di vincolo

- Un oggetto venduto, o rimosso, inserito in bacheca non deve essere visualizzato nella bacheca pubblica;
- Ogni utente deve essere informato delle modifiche svolte agli annunci da lui seguiti, compresi inserimenti, o eliminazioni, di note e commenti;
- Il sistema deve calcolare una percentuale pari al 3% sul report generato;
- Almeno un amministratore deve aver creato una categoria;
- Gli utenti possono vedere, seguire e commentare solo annunci attivi;

Regole di derivazione

- Il numero degli oggetti venduti si ottiene contando il numero degli annunci “venduti”;
- L’importo totale si ottiene sommando tutti gli importi degli annunci venduti;
- Le informazioni della carta di credito nel report si ottiene dall’utente UCC;

9. Dizionario dei dati

Entità	Descrizione	Attributi	Identificatori
Utente	Rappresenta una persona nel sistema	Username, Password	Username
UCC	Utente con Carta di Credito		
USCC	Utente Senza Carta di Credito		
Annuncio	Rappresenta un oggetto nel sistema	Codice, Nota, Commenti, Descrizione, Importo, Foto, ReportCalcolato	Codice
Attivo	Annuncio indicato come attivo, quindi ancora disponibile ad essere venduto		
Venduto	Annuncio indicato come venduto		
Rimosso	Annuncio indicato come rimosso		
Categoria	Rappresenta il tipo di annuncio che si può avere nel sistema	Nome	Nome
Carta di Credito	Rappresenta la carta di credito posseduta da un utente UCC	Numero di carta, CVC, Data Scadenza	Numero di carta
Report	Rappresenta una sorta di fattura per l’utente riguardante i suoi annunci venduti	Codice, Data, numAnnunci, importo_ucc,	Codice

		importo_amm, amministr, riscosso_amm, Numero Carta, Importo totale	
Informazione Anagrafica	Contiene i dati personali degli utenti	CF, Nome, Cognome, Indirizzo di residenza, Indirizzo di fatturazione, CAP	
Storico Conversazione	Rappresenta il registro delle conversazioni dell'utente	ID	ID
Messaggio	Rappresenta l'informazione che si inviano gli utenti	ID, Data, Testo	ID
Conversazione	Rappresenta la comunicazione tra gli utenti	Codice	Codice

4. Progettazione logica

10. Volume dei dati

Concetto nello schema	Tipo[1]	Volume atteso
Utente	E	**40.000**
Utente con carta di credito (UCC)	E	**15.000**
Utente senza carta di credito (USCC)	E	**25.000**
Carta di credito	E	**15.000** *(ogni utente UCC ha una carta di credito)*
Annuncio	E	**30.000** *(ogni UCC ha almeno 2 annunci)*
Attivo	E	**10.000**
Venduto	E	**15.000**
Rimosso	E	**5.000**
Report	E	**15.000** *(almeno ogni utente UCC ha venduto qualcosa)*
Conversazione	E	**200.000** *(ogni utente ha avuto almeno circa 5 conversazioni)*
Storico Conversazione	E	**400.000** *(il doppio delle conversazioni)*
Messaggio	E	**2.000.000** *(ogni conversazione ha almeno 10 messaggi)*
Informazione Anagrafica	E	**40.000**
Categoria	E	**20**
Recapito Non Preferito	E	**40.000** *(ogni utente ha almeno un recapito non preferito)*
Possiede	R	**15.000** *(ogni utente UCC ha una carta di credito)*
Selezionato	R	**240.000**
Tracciato	R	**400.000** *(ogni volta che una conversazione viene creata, viene tracciata nello storico dei due utenti)*
Seguito	R	**120.000** *(in media ogni utente mette 3 annunci tra i preferiti)*
Inserito	R	**15.000** *(almeno 1 annuncio è stato pubblicato da un utente UCC)*
Appartiene	R	**30.000** *(ogni annuncio ha una e una sola relazione con categoria)*
Modificato	R	**40.000**
Partecipa-UCC	R	**178.000**
Riguarda	R	**15.000** *(ogni UCC ha venduto almeno una cosa)*
Is part of	R	**2.000.000** *(una relazione per ogni conversazione)*
Partecipa-USCC	R	**196.000**

1 Indicare con E le entità, con R le relazioni

11. Tavola delle operazioni

Cod.	Descrizione	Frequenza attesa
001	Registrazione utente alla bacheca	**500 al giorno**
002	Modifica informazioni anagrafica dell'utente	**153 al mese**
003	Inserimento dei dati relativi alla carta di credito	**187 al giorno** *(circa il 38% dei nuovi utenti decide di essere anche venditore)*
004	Inserimento di una nuova categoria	**3 al mese**
005	Inserimento di un nuovo annuncio	**559 ogni 4 giorni**
006	Caricamento di una foto dell'oggetto	**102 ogni 4 giorni** *(operazione opzionale)*
007	Visualizzazione di un annuncio da parte di un utente	**25.340 al giorno**
008	Inserimento o Rimozione di un commento ad un annuncio	**1002 al giorno** *(circa il 40% degli utenti che visualizza un annuncio inserisce un nuovo commento)*
009	Invio di un nuovo messaggio da un utente USCC	**125.000 al giorno**
010	Invio di un nuovo messaggio da un UCC	**220.000 al giorno**
011	Visualizzazione di messaggio da parte di utente USCC	**140.000 al giorno**
012	Visualizzazione messaggio da parte di utente UCC	**240.000 al giorno**
013	Seguire un annuncio	**7020 al giorno**
014	Inserimento o Rimozione nota ad un annuncio da parte di un UCC *(come scritto nelle specifiche, il sistema notifica l'evento a tutti gli utenti che stanno seguendo l'annuncio)*	**178 al giorno**
015	Rimozione di un annuncio	**94 ogni 4 giorni**
016	Vendita di un oggetto	**270 ogni 4 giorni**
017	Visualizzazione di tutti gli annunci seguiti	**784 al giorno**
018	Visualizzazione dello storico delle proprie conversazioni da parte di un USCC	**98.000 al giorno**
019	Visualizzazione dello storico delle proprie conversazioni da parte di un UCC	**172.000 al giorno**
020	Generazione di un nuovo report	**143 ogni 4 giorni** *(circa il 53% degli oggetti inseriti vengono venduti)*
021	Selezione di una conversazione dallo storico delle conversazioni da parte di un UCC	**205.000 al giorno**
022	Selezione di una conversazione dallo storico delle conversazioni da parte di un USCC	**116.000 al giorno**
023	Visualizzazione commento di un annuncio	**14.000 al giorno**
024	Visualizzazione di una nota di un annuncio	**10.000 al giorno**
025	Visualizzazione di una categoria	**559 ogni 4 giorni** *(ogni volta che si inserisce un annuncio si consultano le categorie esistenti)*

026	Inserimento o Rimozione di un recapito non preferito	**1000 al giorno**
027	Visualizzazione delle informazioni anagrafiche	**2500 al giorno**
028	Riscossione report di un Amministratore	**572 al giorno**
029	Visualizzazione Report	**1200 al giorno**
030	Visualizzazione delle proprie informazioni utente UCC	**2500 al giorno**
031	Visualizzazione delle proprie informazioni utente USCC	**1500 al giorno**

12.*Nota:*

- *Si è scelto di dare la possibilità all'amministratore di poter segnalare nel database la riscossione della sua percentuale nel report generato, solo per scopi gestionali.*
- *Molte operazioni sono state inventate per rendere più ampio il sistema, ma comunque dedotte e pertinenti alla specifica assegnata*

13.Costo delle operazioni

Operazione 001 con costo 7			
Concetto	**Costrutto**	**Accessi**	**Tipo**
Utente	E	1	S
USCC	E	1	S
Informazioni Anagrafica	E	1	S
Modificato	R	1	S
Operazione 002 con costo 5			
Concetto	**Costrutto**	**Accessi**	**Tipo**
Utente	E	1	L
Informazioni Anagrafica	E	1	S
Modificato	R	1	S
Operazione 003 con costo 6			
Concetto	**Costrutto**	**Accessi**	**Tipo**
Utente	E	1	L
UCC	E	1	L
Carta di Credito	E	1	S

Possiede	R	1	S
Operazione 004 con costo 2			
Concetto	**Costrutto**	**Accessi**	**Tipo**
Categoria	E	1	S
Operazione 005 con costo 8			
Concetto	**Costrutto**	**Accessi**	**Tipo**
UCC	E	1	L
Annuncio	E	1	S
Categoria	E	1	L
Inserito	R	1	S
Appartiene	R	1	S
Operazione 006 con costo 5			
Concetto	**Costrutto**	**Accessi**	**Tipo**
UCC	E	1	L
Annuncio	E	1	S
Inserito	R	1	S
Operazione 007 con costo 6			
Concetto	**Costrutto**	**Accessi**	**Tipo**
Utente	E	1	L
Attivo	E	1	L
Annuncio	E	1	L
UCC	E	1	L
Seguito	R	1	L
Inserito	R	1	L
Operazione 008 con costo 5			
Concetto	**Costrutto**	**Accessi**	**Tipo**
Utente	E	1	L
Annuncio	E	1	S
Seguito	R	1	S
Operazione 009 con costo 7			
Concetto	**Costrutto**	**Accessi**	**Tipo**
USCC	E	1	L
Partecipa-USCC	R	1	L

Conversazione	E	1	L
Messaggio	E	1	S
Is part of	R	1	S
Operazione 010 con costo 7			
Concetto	**Costrutto**	**Accessi**	**Tipo**
UCC	E	1	L
Partecipa-UCC	R	1	L
Conversazione	E	1	L
Is part of	R	1	S
Messaggio	E	1	S
Operazione 011 con costo 6			
Concetto	**Costrutto**	**Accessi**	**Tipo**
Utente	E	1	L
USCC	E	1	L
Partecipa-USCC	R	1	L
Conversazione	E	1	L
Messaggio	E	1	L
Is part of	R	1	L
Operazione 012 con costo 5			
Concetto	**Costrutto**	**Accessi**	**Tipo**
UCC	E	1	L
Partecipa-UCC	R	1	L
Conversazione	E	1	L
Is part of	R	1	L
Messaggio	E	1	L
Operazione 013 con costo 5			
Concetto	**Costrutto**	**Accessi**	**Tipo**
Utente	E	1	L
Seguito	R	1	S
Attivo	E	1	L
Annuncio	E	1	L
Operazione 014 con costo 32			
Concetto	**Costrutto**	**Accessi**	**Tipo**
Annuncio	E	1	S

Inserito	R	1	S
Attivo	E	1	L
UCC	E	1	L
Seguito	R	14	L
Utente	E	14	L
Operazione 015 con costo 7			
Concetto	**Costrutto**	**Accessi**	**Tipo**
Annuncio	E	1	S
UCC	E	1	L
Rimosso	E	1	S
Inserito	R	1	S
Operazione 016 con costo 7			
Concetto	**Costrutto**	**Accessi**	**Tipo**
UCC	E	1	L
Annuncio	E	1	S
Venduto	E	1	S
Inserito	R	1	S
Operazione 017 con costo 43			
Concetto	**Costrutto**	**Accessi**	**Tipo**
Utente	E	1	L
Annuncio	E	14	L
Attivo	E	5	L
Venduto	E	7	L
Rimosso	E	2	L
Seguito	R	14	L
Operazione 018 con costo 4			
Concetto	**Costrutto**	**Accessi**	**Tipo**
USCC	E	1	L
Utente	E	1	L
Selezionato	R	1	L
Storico Conversazione	E	1	L
Operazione 019 con costo 4			
Concetto	**Costrutto**	**Accessi**	**Tipo**
UCC	E	1	L

Utente	E	1	L
Selezionato	R	1	L
Storico Conversazione	E	1	L
Operazione 020 con costo 8			
Concetto	**Costrutto**	**Accessi**	**Tipo**
Report	E	1	S
Riguarda	R	1	S
UCC	E	1	L
Inserito	R	1	L
Annuncio	E	1	L
Venduto	E	1	L
Operazione 021 con costo 14			
Concetto	**Costrutto**	**Accessi**	**Tipo**
UCC	E	1	L
Utente	E	1	L
Selezionato	R	1	L
Storico Conversazione	E	1	L
Tracciato	E	5	L
Conversazione	E	5	L
Operazione 022 con costo 14			
Concetto	**Costrutto**	**Accessi**	**Tipo**
USCC	E	1	L
Utente	E	1	L
Selezionato	R	1	L
Storico Conversazione	E	1	L
Tracciato	E	5	L
Conversazione	E	5	L
Operazione 023 con costo 3			
Concetto	**Costrutto**	**Accessi**	**Tipo**
Utente	E	1	L
Annuncio	E	1	L
Seguito	R	1	L
Operazione 024 con costo 3			
Concetto	**Costrutto**	**Accessi**	**Tipo**
Utente	E	1	L

Annuncio	E	1	L
Seguito	R	1	L
Operazione 025 con costo 20			
Concetto	**Costrutto**	**Accessi**	**Tipo**
Categoria	E	20	L
Operazione 026 con costo 5			
Concetto	**Costrutto**	**Accessi**	**Tipo**
Utente	E	1	L
Informazioni Anagrafica	E	1	S
Modificato	R	1	S
Operazione 027 con costo 3			
Concetto	**Costrutto**	**Accessi**	**Tipo**
Utente	E	1	L
Informazioni Anagrafica	E	1	L
Modificato	R	1	L
Operazione 028 con costo 2			
Concetto	**Costrutto**	**Accessi**	**Tipo**
Report	E	1	S
Operazione 029 con costo 1			
Concetto	**Costrutto**	**Accessi**	**Tipo**
Report	E	1	L
Operazione 030 con costo 4			
Concetto	**Costrutto**	**Accessi**	**Tipo**
Utente	E	1	L
UCC	E	1	L
Carta di Credito	E	1	L
Possiede	R	1	L
Operazione 031 con costo 2			
Concetto	**Costrutto**	**Accessi**	**Tipo**
Utente	E	1	L
USCC	E	1	L

- 001 costo 7, Al giorno 500 Costi al giorno 3500

- 002 costo 5, Al mese 153 Costi al mese 765
- 003 costo 6, Al giorno 187 Costi al giorno 1122
- 004 costo 2, Al mese 3 Costi al mese 6
- 005 costo 8, Ogni 4 giorni 559 Costi ogni 4 giorni 4472
- 006 costo 5, Ogni 4 giorni 102 Costi ogni 4 giorni 510
- 007 costo 6, Al giorno 25.340 Costi al giorno 152.040
- 008 costo 5, Al giorno 1002 Costi al giorno 5010
- 009 costo 7, Al giorno 125.000 Costi al giorno 875.000
- 010 costo 7, Al giorno 220.000 Costi al giorno 1.540.000
- 011 costo 6, Al giorno 140.000 Costi al giorno 840.000
- 012 costo 5, Al giorno 240.000 Costi al giorno 1.200.000
- 013 costo 5, Al giorno 7020 Costi al giorno 35.100
- 014 costo 32, Al giorno 178 Costi al giorno 5696
- 015 costo 7, Ogni 4 giorni 94 Costi ogni 4 giorni 658
- 016 costo 7, Ogni 4 giorni 270 Costi ogni 4 giorni 1890
- 017 costo 43, Al giorno 784 Costi al giorno 33.712
- 018 costo 4, Al giorno 98.000 Costi al giorno 392.000
- 019 costo 4, Al giorno 172.000 Costi al giorno 688.000
- 020 costo 8, Ogni 4 giorni 143 Costi ogni 4 giorni 1144
- 021 costo 14, Ogni giorno 205.000 Costi ogni giorno 2.870.000
- 022 costo 14, Ogni giorno 116.000 Costi ogni giorno 1.624.000
- 023 costo 3, Ogni giorno 14.000 Costi ogni giorno 42.000
- 024 costo 3, Ogni giorno 10.000 Costi ogni giorno 30.000
- 025 costo 20, Ogni 4 giorni 559 Costi ogni 4 giorni 11.180
- 026 costo 5, Ogni 4 giorni 559 Costi ogni giorno 5.000
- 027 costo 3, Ogni 4 giorni 559 Costi ogni giorno 7.500
- 028 costo 2, Ogni 4 giorni 559 Costi ogni giorno 1144
- 029 costo 1, Ogni 4 giorni 559 Costi ogni giorno 1200
- 030 costo 4, Ogni 4 giorni 559 Costi ogni giorno 10.000
- 031 costo 2, Ogni 4 giorni 559 Costi ogni giorno 3.000

14.

15.Ristrutturazione dello schema E-R

Analisi delle ridondanze

- Visualizzazione di un annuncio da parte di un utente senza dato ridondante (operazione 007)
  152.040 accessi
  - Visualizzazione di un annuncio da parte di un utente con dato ridondante
    101.360 accessi (attributo: foreign key to UCC)

Operazione 007 con costo 4			
Concetto	**Costrutto**	**Accessi**	**Tipo**
Utente	E	1	L
Attivo	E	1	L
Annuncio	E	1	L
Seguito	R	1	L

Si è scelto di inserire un attributo ridondante “UCC_Username” che sarà una foreign key ad UCC, ovvero si attuerà una traduzione non ottima in “Annuncio”.

- Inserimento dei dati relativi alla carta di credito senza dato ridondante (operazione 003)
  1122 accessi
  - Inserimento dei dati relativi alla carta di credito con dati ridondanti
    561 accessi (attributi: Codice carta, Data di scadenza e CVC)
    Essendo l’operazione 001 molto frequente, solo circa il 40% degli utenti diventano venditori, ma se si riflettesse in spazio di memoria e in accessi risparmiati in un lungo periodo questo porterebbe un notevole vantaggio di prestazioni. Considerando un risparmio del circa 50% in termini di prestazioni.

Operazione 003 con costo 3			
Concetto	**Costrutto**	**Accessi**	**Tipo**
Utente	E	1	L
UCC	E	1	S

- Selezione di una conversazione dallo storico delle conversazioni senza dato ridondante (operazione 021 ed operazione 022)
  2.870.000 accessi, 1.624.000 accessi
  - Selezione di una conversazione dallo storico delle conversazioni con dato ridondante
  820.000 accessi, 464.000 accessi (attributo multivalore: ConversazioniCodice)
  L'aggiunta dell'attributo comporta un grande vantaggio in termini di tempo

Operazione 021 con costo 4			
Concetto	**Costrutto**	**Accessi**	**Tipo**
UCC	E	1	L
Utente	E	1	L
Selezionato	R	1	L
Storico Conversazione	E	1	L
Operazione 022 con costo 4			
Concetto	**Costrutto**	**Accessi**	**Tipo**
USCC	E	1	L
Utente	E	1	L
Selezionato	R	1	L
Storico Conversazione	E	1	L

Eliminazione delle generalizzazioni

La seconda ristrutturazione consiste nel diversificare i concetti di "Annuncio" e "Utente" andando a eliminare le generalizzazioni. Si è deciso così di:

1. Eliminare le entità figlie di "Annuncio" in quanto le operazioni più frequenti sono eseguite sull'entità padre;
2. Accorpare l'entità "Utente" nelle figlie in quanto i concetti chiavi e le operazioni più importanti vengono effettuate sulle entità figlie;

Eliminazione degli attributi multivalori

- Annuncio:
  - Commento;
  - Nota;

Sono stati eliminati questi due attributi multivalori e si è deciso quindi di creare due entità con due relazioni di dipendenza verso “Annuncio”.

- Storico Conversazione
  - ConversazioniCodice

È stato eliminato l’attributo multivalore e si è deciso quindi di creare un entità con una relazione di dipendenza verso “Storico Conversazione”.

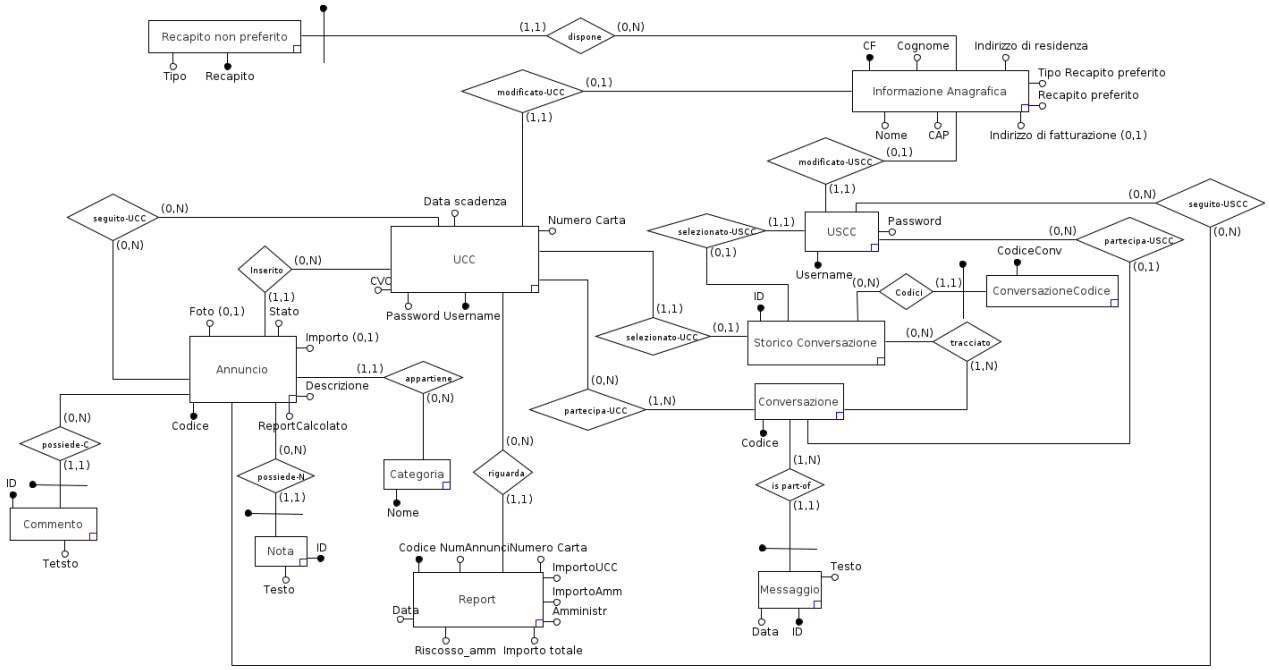
Scelta identificatori primari

Entità	Identificatore primario
UCC	Username
USCC	Username
Annuncio	Codice
Informazione Anagrafica	CF
Conversazione	Codice
Messaggio	ID
Categoria	Nome
Recapito non preferito	Recapito
Nota	ID
Commento	ID
Storico Conversazione	ID
ConversazioneCodice	CodiceConv
Report	Codice

16. Trasformazione di attributi e identificatori

- Gli attributi Numero Carta, Data Scadenza, CVC in UCC sono riferiti alla carta di credito.

Modello Ristrutturato



Nota: A causa di alcune limitazioni del tool usato per costruire il modello, la cardinalità (1, N) tra "Conversazione" e "Partecipa-UCC" è (1, 2), in quanto le uniche conversazione previste sono:

- *UCC - UCC*
- *UCC – USCC*

Quindi non sono previste comunicazioni USCC – USCC vista la sua poca importanza a far comunicare due utenti che non hanno lo scopo di offrire un bene nel sistema. Quindi ciò avrebbe portato un aumento del traffico di informazioni poco "utili", di conseguenza un degrado delle prestazioni.

17.

18. Traduzione di entità e associazioni

- Annuncio(Codice, Descrizione, Importo*, Stato, Foto*, ReportCalcolato, *Categoria_Nome*, *UCC_Username*);
- Nota(ID, *Annuncio_Codice*, Testo);
- Commento(*ID*, *Annuncio_Codice*, Testo);
- Categoria(Nome);

- Seguito-UCC(*Annuncio_Codice*, *UCC_Username*);
- Seguito-USCC(*Annuncio_Codice*, *USCC_Username*);
- UCC(Username, Password, Numero Carta, Data scadenza, CVC, *ID_StoricoConversazione*, *CF_Anagrafico*);
- Report(*Codice*, ImportoTotale, NumAnnunci, Numero Carta, Data, Riscosso_Amm, Importo_UCC, Importo_Amm, Amministr, *UCC_Username*);
- USCC(Username, Password, *ID_StoricoConversazione*, *CF_Anagrafico*);
- Tracciato(*Codice_Conversazione*, *ID_StoricoConversazione*);
- ConversazioneCodice(CodiceConv, *ID_StoricoConversazione*);
- StoricoCoversazione(ID);
- Conversazione(Codice, *UCC_Username_1*, *UCC_Username_2**, *USCC_Username**);
- Messaggio(ID, *Codice_Conversazione*, Data, Testo);
- Informazione Anagrafica(CF, Cognome, Nome, Indirizzo di residenza, CAP, Indirizzo di fatturazione*, Tipo Recapito preferito, Recapito preferito);
- Recapito non preferito(Recapito, *Informazione Anagrafica_CF*, Tipo);

Vincoli referenziali

ConversazioneCodice(ID_StoricoConversazione) ⊆ StoricoConversazione(ID);

Tracciato(ID_StoricoConversazione) ⊆ StoricoConversazione(ID);

Tracciato(Codice_Conversazione) ⊆ Conversazione(Codice);

Conversazione(*UCC_Username_1*) ⊆ UCC(Username);

Conversazione(*UCC_Username_2*) ⊆ UCC(Username);

Conversazione(*USCC_Username*) ⊆ USCC(Username);

USCC(ID_StoricoConversazione) ⊆ StoricoConversazione(ID);

USCC(*CF_Anagrafico*) ⊆ Informazione Anagrafica(CF);

UCC(ID_StoricoConversazione) ⊆ StoricoConversazione(ID);

UCC(*CF_Anagrafico*) ⊆ Informazione Anagrafica(CF);

Annuncio(Categoria_Nome) ⊆ Categoria(Nome);

Annuncio(*UCC_Username*) ⊆ UCC(Username);

Nota(Annuncio_Codice) ⊆ Annuncio(Codice);

Commento(Annuncio_Codice) ⊆ Annuncio(Codice);

Seguito-UCC(Annuncio_Codice) ⊆ Annuncio(Codice);

Seguito-UCC(UCC_Username) ⊆ UCC(Username);

Seguito-USCC(Annuncio_Codice) ⊆ Annuncio(Codice);

Seguito-USCC(USCC_Username) ⊆ USCC(Username);

Report(UCC_Username) ⊆ UCC(Username);

Messaggio(Codice_Conversazione) ⊆ Conversazione(Codice);

Recapito non preferito(Informazione Anagrafica_CF) ⊆ Informazione Anagrafica(CF);

Nota: i valori scritti in corsivo sono vincoli referenziali

19.Normalizzazione del modello relazionale

Tutte le tabelle sono in 3NF.

5. Progettazione fisica

20.Utenti

Descrivere, all'interno dell'applicazione, quali utenti sono stati previsti con quali privilegi di accesso su quali tabelle, giustificando le scelte progettuali.

- Utente con carta di credito (UCC);
- Utente senza carta di credito (USCC);
- Amministratore;
- Login

Privilegi

- UCC (dispone anche di tutti i privilegi di USCC):
  - Inserire un nuovo Annuncio
  - Rimuovere il proprio Annuncio
  - Indicare come Venduto il suo annuncio
  - Modificare le proprie informazioni Anagrafiche
  - Aggiungere o Rimuovere una foto ad un suo Annuncio
  - Aggiungere o Rimuovere una nota ad un suo Annuncio
  - Inviare un messaggio
  - Visualizzare gli Annunci seguiti
  - Visualizzare i suoi Report

- USCC:
  - Visualizzare un Annuncio
  - Visualizzare una Categoria
  - Visualizzare le proprie informazioni
  - Modificare le proprie informazioni Anagrafiche
  - Aggiungere o Rimuovere un commento in un annuncio
  - Visualizzare un commento
  - Visualizzare una nota

- Aggiungere o Rimuovere un recapito non preferito
- Visualizzare tutti i suoi recapiti
- Inviare un messaggio
- Visualizzare un messaggio
- Visualizzare il suo Storico
- Seguire un Annuncio
- Visualizzare gli Annunci seguiti
- Visualizzare le sue notifiche

- Amministratore:
  - Inserimento nuova Categoria
  - Visualizzare una categoria
  - Visualizzare un Annuncio
  - Generare un report
  - Riscuotere un report
  - Visualizzare un report
- Login:
  - Login
  - Registrare un nuovo utente UCC
  - Registrare un nuovo utente USCC

21.Strutture di memorizzazione

Tabella Categoria		
Attributo	**Tipo di dato**	**Attributi**[2]
Nome	VARCHAR(20)	PK, NN, UQ

Tabella UCC		
Attributo	**Tipo di dato**	**Attributi**
Username	VARCHAR(45)	PK, NN, UQ
Password	VARCHAR(45)	NN
NumeroCarta	VARCHAR(16)	NN
DataScadenza	DATE	NN
CVC	INT	NN
StoricoConversazione_ID	INT	NN
CF_Anagrafico	VARCHAR(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.

Tabella Annuncio		
Attributo	**Tipo di dato**	**Attributi**
Codice	INT	PK, NN, UQ, AI
Stato	ENUM('Attivo', 'Venduto', 'Rimosso')	NN
Descrizione	VARCHAR(100)	NN
Importo	INT	NN
Foto	VARCHAR(9)	
Report_calcolato	TINYINT	NN
UCC_Username	VARCHAR(45)	NN
Categoria_Nome	VARCHAR(20)	NN

Tabella Amministratore		
Attributo	**Tipo di dato**	**Attributi**
Username	VARCHAR(45)	PK, NN, UQ
Password	VARCHAR(45)	NN

Tabella Nota		
Attributo	**Tipo di dato**	**Attributi**
ID	INT	PK, NN, AI
Testo	VARCHAR(45)	NN
Annuncio_Codice	INT	PK. NN. UQ

Tabella Seguito-USCC		
Attributo	**Tipo di dato**	**Attributi**
Annuncio_Codice	INT	PK, NN
USCC_Username	VARCHAR(45)	PK, NN

5

Tabella Conversazione		
Attributo	**Tipo di dato**	**Attributi**
Codice	INT	PK, NN, UQ, AI
UCC_Username_1	VARCHAR(45)	NN
UCC_Username_2	VARCHAR(45)	
USCC_Username	VARCHAR(45)	

Tabella StoricoConversazione		
Attributo	**Tipo di dato**	**Attributi**
ID	INT	PK, NN, UQ, AI

Tabella Seguito-UCC		
Attributo	**Tipo di dato**	**Attributi**
Annuncio_Codice	INT	PK, NN
UCC_Username	VARCHAR(45)	PK, NN

Tabella USCC		
Attributo	**Tipo di dato**	**Attributi**
Username	VARCHAR(45)	PK, NN, UQ
Password	VARCHAR(45)	NN
StoricoConversazione_ID	INT	NN
CF_Anagrafico	VARCHAR(16)	NN

Tabella Messaggio		
Attributo	**Tipo di dato**	**Attributi**
ID	INT	PK, NN, AI
Data	DATETIME	NN
Conversazione_Codice	INT	PK, NN
Testo	VARCHAR(100)	NN

Tabella InformazioneAnagrafica		
Attributo	**Tipo di dato**	**Attributi**
CF	VARCHAR(16)	PK, NN, UQ
Cognome	VARCHAR(20)	NN
Nome	VARCHAR(20)	NN
IndirizzoDiResidenza	VARCHAR(20)	NN
CAP	INT	NN
IndirizzoDiFatturazione	VARCHAR(20)	
TipoRecapitoPreferito	ENUM('email', 'cellulare', 'social', 'sms')	NN
RecapitoPreferito	VARCHAR(40)	NN

5

Tabella Notifica		
Attributo	**Tipo di dato**	**Attributi**
Codice_Notifica	INT	PK, NN, UQ, AI
Codice	INT	NN
Tipo_Notifica	VARCHAR(45)	NN
Username_Utente	VARCHAR(45)	NN
Data	DATETIME	NN
Messaggio	VARCHAR(45)	NN

Tabella RecapitoNonPreferito		
Attributo	**Tipo di dato**	**Attributi**
Recapito	VARCHAR(45)	PK, NN
Tipo	ENUM(‘email’, ‘cellulare’, ‘social’, ‘sms’)	NN
InformazioneAnagrafica_CF	VARCHAR(16)	PK, NN

Tabella Commento		
Attributo	**Tipo di dato**	**Attributi**
ID	INT	PK, NN, AI
Testo	VARCHAR(45)	NN
Annuncio_Codice	INT	PK, NN, UQ

Tabella Report		
Attributo	**Tipo di dato**	**Attributi**
Codice	INT	PK, NN, UQ, AI
UCC_Username	VARCHAR(45)	NN
ImportoTotale	INT	NN
NumeroAnnunci	INT	NN
NumeroCarta	VARCHAR(16)	NN
Data	DATE	NN
Amministratore_Username	VARCHAR(45)	NN
Riscosso_Amministratore	TINYINT	NN
Importo_Amministratore	INT	NN
Importo_UCC	INT	NN

Tabella ConversazioneCodice		
Attributo	**Tipo di dato**	**Attributi**
CodiceConv	INT	PK, NN
StoricoConversazione_ID	INT	PK, NN

Tabella Tracciato		
Attributo	**Tipo di dato**	**Attributi**
Conversazione_Codice	INT	PK, NN
StoricoConversazione_ID	INT	PK, NN

22.Indici

Tabella UCC	
Indice PRIMARY	**Tipo[3]:**
Username	PRIMARY
Indice Username_UNIQUE	**Tipo:**
Username	UNIQUE
Indice fk_UCC_StoricoConversazioni1_idx	**Tipo:**
StoricoConversazione_ID	INDEX
Indice fk_UCC_InformazioneAnagrafica1_idx	**Tipo:**
CF_Anagrafico	INDEX

5

Tabella Nota	
Indice PRIMARY	**Tipo:**
ID	PRIMARY

3

Indice fk_Annuncio_Codice _UNIQUE	**Tipo:**
Annuncio_Codice	UNIQUE
Indice fk_Nota_Annuncio1_idx	**Tipo:**
Annuncio_Codice	INDEX

Tabella Annuncio	
Indice PRIMARY	**Tipo:**
Codice	PRIMARY
Indice Codice_UNIQUE	**Tipo:**
Codice	UNIQUE
Indice fk_Annuncio_UCC_idx	**Tipo:**
UCC_Username	INDEX
Indice fk_Annuncio_Categoria 1_idx	**Tipo:**
Categoria_Nome	INDEX

Tabella Seguito-USCC	
Indice PRIMARY	**Tipo:**
Annuncio_Codice	PRIMARY
Indice PRIMARY	**Tipo:**
USCC_Username	PRIMARY
Indice fk_Seguito-USCC_Annuncio1_idx	**Tipo:**
Annuncio_Codice	INDEX
Indice fk_Seguito-USCC_USCC1_idx	**Tipo:**
USCC_Username	INDEX

5

Tabella Seguito-UCC	
Indice PRIMARY	**Tipo:**
Annuncio_Codice	PRIMARY
Indice PRIMARY	**Tipo:**
UCC_Username	PRIMARY
Indice fk_Seguito-UCC_Annuncio1_idx	**Tipo:**
Annuncio_Codice	INDEX
Indice fk_Seguito-UCC_UCC1_idx	**Tipo:**
UCC_Username	INDEX

Tabella Conversazione	
Indice PRIMARY	**Tipo:**
Codice	PRIMARY
Indice Codice_UNIQUE	**Tipo:**
Codice	UNIQUE
Indice fk_Conversazione_UCC1_idx	**Tipo:**

UCC_Username_1	INDEX
Indice fk_Conversazione_USCC1_idx	**Tipo:**
USCC_Username	INDEX
Indice fk_Conversazione_UCC2_idx	**Tipo:**
UCC_Username_2	INDEX

Tabella USCC	
Indice PRIMARY	**Tipo:**
Username	PRIMARY
Indice Username_UNIQUE	**Tipo:**
Username	UNIQUE
Indice fk_USCC_StoricoConversazione1_idx	**Tipo:**
StoricoConversazione_ID	INDEX
Indice fk_USCC_InformazioneAnagrafica1_idx	**Tipo:**
CF_Anagrafico	INDEX

Tabella Messaggio	
Indice PRIMARY	**Tipo:**
ID	PRIMARY
Indice PRIMARY	**Tipo:**
Conversazione_Codice	PRIMARY
Indice fk_Messaggio_Conversazione1_idx	**Tipo:**
Conversazione_Codice	INDEX

Tabella InformazioneAnagrafica	
Indice PRIMARY	**Tipo:**
CF	PRIMARY
Indice CF_UNIQUE	**Tipo:**
CF	UNIQUE

Tabella Notifica	
Indice PRIMARY	**Tipo:**
Codice_Notifica	PRIMARY
Indice CF_UNIQUE	**Tipo:**
Codice_Notifica	UNIQUE

5

Tabella RecapitoNonPreferito	
Indice PRIMARY	**Tipo:**
Recapito	PRIMARY
Indice PRIMARY	**Tipo:**
InformazioneAnagrafica_CF	PRIMARY
Indice fk_RecapitoNonPreferito_InformazioneAnagrafica1_idx	**Tipo:**
InformazioneAnagrafica_CF	INDEX

Tabella Amministratore	
Indice PRIMARY	**Tipo:**
Username	PRIMARY
Indice Username_UNIQUE	**Tipo:**
Username	UNIQUE

Tabella Commento	
Indice PRIMARY	**Tipo:**
ID	PRIMARY
Indice PRIMARY	**Tipo:**
Annuncio_Codice	PRIMARY
Indice Annuncio_Codice_UNIQUE	**Tipo:**
Annuncio_Codice	UNIQUE
Indice fk_Commento_Annuncio1_idx	**Tipo:**
Annuncio_Codice	INDEX

Tabella Report	
Indice PRIMARY	**Tipo:**
Codice	PRIMARY
Indice Codice _UNIQUE	**Tipo:**
Codice	UNIQUE
Indice fk_Report_UCC1_idx	**Tipo:**
UCC_Username	INDEX

Tabella ConversazioneCodice	
Indice PRIMARY	**Tipo:**
CodiceConv	PRIMARY
Indice PRIMARY	**Tipo:**
StoricoConversazione_ID	PRIMARY
Indice fk_ConversazioneCodice_StoricoConversazione1_idx	**Tipo:**
StoricoConversazione_ID	INDEX

Tabella Tracciato	
Indice PRIMARY	**Tipo:**
Conversazione_Codice	PRIMARY
Indice PRIMARY	**Tipo:**
StoricoConversazione_ID	PRIMARY

Indice fk_Tracciato_StoricoConversazione1_idx	**Tipo:**
StoricoConversazione_ID	INDEX
Indice fk_Tracciato_Conversazione1_idx	**Tipo:**
Conversazione_Codice	INDEX

Tabella Categoria	
Indice PRIMARY	**Tipo:**
Nome	PRIMARY
Indice Nome_UNIQUE	**Tipo:**
Nome	UNIQUE

Tabella StoricoConversazione	
Indice PRIMARY	**Tipo:**
ID	PRIMARY
Indice ID_UNIQUE	**Tipo:**
ID	UNIQUE

NOTA:

1. **In caso di necessità c'è un backup del progetto nel mio account github**
2. **https://github.com/Monello1299/Basi-di-Dati**
3. **Per una lettura migliore del codice utilizzare un editor di testo**

Trigger

-- Avviso Nota Inserita ----------

L'idea del trigger è di soddisfare la regola aziendale di avvisare ogni utente che segue l'annuncio di una nuova nota. Viene eseguito un ciclo in modo da permettere di inserire per ogni utente il suo messaggio e sarà poi alla stored procedure la responsabilità di gestire i dati. Ho applicato una sorta di pattern GRASP in modo da avere che ogni utente ha il suo messaggio e non un unico messaggio dove è condiviso per tutti, quindi da avere un low coupling tra gli utenti (ogni row contentente il messaggio dipende solo dall'utente indirizzato)

```
DELIMITER //
DROP TRIGGER IF EXISTS BachecaElettronicadb.warn_new_note ;
CREATE DEFINER = CURRENT_USER TRIGGER BachecaElettronicadb.warn_new_note
AFTER INSERT ON BachecaElettronicadb.Nota FOR EACH ROW
BEGIN
    declare username VARCHAR(45);
    declare loop_username INT DEFAULT 0;
```

```
declare cur_username_ucc_user CURSOR FOR SELECT seguiti.UCC_Username FROM BachecaElettronicadb.Nota AS nota JOIN BachecaElettronicadb.`Seguito-UCC` AS seguiti ON nota.Annuncio_Codice = seguiti.Annuncio_Codice;
declare cur_username_uscc_user CURSOR FOR SELECT seguiti.USCC_Username FROM BachecaElettronicadb.Nota AS nota JOIN BachecaElettronicadb.`Seguito-USCC` AS seguiti ON nota.Annuncio_Codice = seguiti.Annuncio_Codice;
declare CONTINUE HANDLER FOR NOT FOUND SET loop_username = 1;


-- Seleziona tutti gli utenti che seguono l'annuncio e inserisce una nuova entry in modo da notificare l'evento
OPEN cur_username_ucc_user;
insert_new_ucc_warning:  LOOP
                                                FETCH cur_username_ucc_user INTO username;

                                            IF loop_username = 1 THEN
                                                LEAVE insert_new_ucc_warning;
                                            end IF;

                                                INSERT INTO BachecaElettronicadb.Notifica(Codice, Tipo_Notifica, Username_Utente, Data, Messaggio)
                                                VALUES (NEW.ID, 'Nota Annuncio', username, now(), 'Nuovo Inserimento');

end LOOP insert_new_ucc_warning;
CLOSE cur_username_ucc_user;

SET loop_username = 0;
SET username = NULL;

OPEN cur_username_uscc_user;
insert_new_uscc_warning:  LOOP
```

```
                FETCH cur_username_uscc_user INTO username;

                IF loop_username = 1 THEN
                    LEAVE insert_new_uscc_warning;
                end IF;

                INSERT INTO BachecaElettronicadb.Notifica(Codice, Tipo_Notifica, Username_Utente, Data, Messaggio)
                VALUES (NEW.ID, 'Nota Annuncio', username, now(), 'Nuovo Inserimento');

        end LOOP insert_new_uscc_warning;
        CLOSE cur_username_uscc_user;


END//
DELIMITER ;
-- -----------------------------------------------------------------

-- Avviso Nota Eliminata ----------------------------------------------

DELIMITER //
DROP TRIGGER IF EXISTS BachecaElettronicadb.warn_note_deleted ;
CREATE DEFINER = CURRENT_USER TRIGGER BachecaElettronicadb.warn_note_deleted AFTER DELETE ON BachecaElettronicadb.Nota FOR EACH ROW
BEGIN
        declare username VARCHAR(45);
        declare loop_username INT DEFAULT 0;

        declare cur_username_ucc_user CURSOR FOR SELECT seguiti.UCC_Username FROM BachecaElettronicadb.Nota AS nota JOIN BachecaElettronicadb.`Seguito-UCC` AS seguiti ON nota.Annuncio_Codice = seguiti.Annuncio_Codice;
```

```
declare cur_username_uscc_user CURSOR FOR SELECT seguiti.USCC_Username FROM BachecaElettronicadb.Nota AS nota JOIN BachecaElettronicadb.`Seguito-USCC` AS seguiti ON nota.Annuncio_Codice = seguiti.Annuncio_Codice;
declare CONTINUE HANDLER FOR NOT FOUND SET loop_username = 1;


/*NOTA
 * L'idea del trigger è di soddisfare la regola aziendale di avvisare ogni utente
 * che segue l'annuncio di una nota eliminata. Viene eseguito un ciclo in modo da permettere
 * di inserire per ogni utente il suo messaggio e sarà poi alla stored procedure la responsabilità
 * di gestire i dati. Ho applicato una sorta di pattern GRASP in modo da avere che ogni utente
 * ha il suo messaggio e non un unico messaggio dove è condiviso per tutti, quindi da avere un low
 * coupling tra gli utenti (ogni row contentente il messaggio dipende solo dall'utente indirizzato)
 */

-- Seleziona tutti gli utenti che seguono l'annuncio e inserisce una nuova entry in modo da notificare l'evento
OPEN cur_username_ucc_user;
insert_new_ucc_warning:   LOOP
                                                FETCH cur_username_ucc_user INTO username;

                                            IF loop_username = 1 THEN
                                                LEAVE insert_new_ucc_warning;
                                            end IF;

                                                INSERT INTO BachecaElettronicadb.Notifica(Codice, Tipo_Notifica, Username_Utente, Data, Messaggio)
                                                VALUES (OLD.ID, 'Nota Annuncio', username, now(), 'Nota Eliminata');
```

```
        end LOOP insert_new_ucc_warning;
        CLOSE cur_username_ucc_user;

        SET loop_username = 0;
        SET username = NULL;

        OPEN cur_username_uscc_user;
        insert_new_uscc_warning:  LOOP
                                                FETCH cur_username_uscc_user INTO
username;

                                                IF loop_username = 1 THEN
                                                    LEAVE insert_new_uscc_warning;
                                                end IF;

                                                INSERT INTO
BachecaElettronicadb.Notifica(Codice, Tipo_Notifica, Username_Utente, Data, Messaggio)
                                                VALUES (OLD.ID, 'Nota Annuncio',
username, now(), 'Nota Eliminata');

        end LOOP insert_new_uscc_warning;
        CLOSE cur_username_uscc_user;


END//
DELIMITER ;
-- ----------------------------------------------------------------

-- Avviso Commento Inserito ----------------------------------------

DELIMITER //
DROP TRIGGER IF EXISTS BachecaElettronicadb.warn_new_comment ;
CREATE DEFINER = CURRENT_USER TRIGGER BachecaElettronicadb.warn_new_comment
AFTER INSERT ON BachecaElettronicadb.Commento FOR EACH ROW
```

```
BEGIN
    declare username VARCHAR(45);
    declare loop_username INT DEFAULT 0;

    declare cur_username_ucc_user CURSOR FOR SELECT seguiti.UCC_Username FROM BachecaElettronicadb.Commento AS commento JOIN BachecaElettronicadb.`Seguito-UCC` AS seguiti ON commento.Annuncio_Codice = seguiti.Annuncio_Codice;
    declare cur_username_uscc_user CURSOR FOR SELECT seguiti.USCC_Username FROM BachecaElettronicadb.Commento AS commento JOIN BachecaElettronicadb.`Seguito-USCC` AS seguiti ON commento.Annuncio_Codice = seguiti.Annuncio_Codice;
    declare CONTINUE HANDLER FOR NOT FOUND SET loop_username = 1;


    /*NOTA
     * L'idea del trigger è di soddisfare la regola aziendale di avvisare ogni utente
     * che segue l'annuncio di un nuovo commento. Viene eseguito un ciclo in modo da permettere
     * di inserire per ogni utente il suo messaggio e sarà poi alla stored procedure la responsabilità
     * di gestire i dati. Ho applicato una sorta di pattern GRASP in modo da avere che ogni utente
     * ha il suo messaggio e non un unico messaggio dove è condiviso per tutti, quindi da avere un low
     * coupling tra gli utenti (ogni row contentente il messaggio dipende solo dall'utente indirizzato)
     */

    -- Seleziona tutti gli utenti che seguono l'annuncio e inserisce una nuova entry in modo da notificare l'evento
    OPEN cur_username_ucc_user;
    insert_new_ucc_warning:  LOOP
                                    FETCH cur_username_ucc_user INTO username;

                                    IF loop_username = 1 THEN
                                        LEAVE insert_new_ucc_warning;
```

```
                                    end IF;

                                    INSERT INTO BachecaElettronicadb.Notifica(Codice,
Tipo_Notifica, Username_Utente, Data, Messaggio)
                                        VALUES (NEW.ID, 'Commento Annuncio',
username, now(), 'Nuovo Commento');

        end LOOP insert_new_ucc_warning;
        CLOSE cur_username_ucc_user;

        SET loop_username = 0;
        SET username = NULL;

        OPEN cur_username_uscc_user;
        insert_new_uscc_warning: LOOP
                                    FETCH cur_username_uscc_user INTO username;

                                    IF loop_username = 1 THEN
                                        LEAVE insert_new_uscc_warning;
                                    end IF;

                                    INSERT INTO BachecaElettronicadb.Notifica(Codice,
Tipo_Notifica, Username_Utente, Data, Messaggio)
                                        VALUES (NEW.ID, 'Commento Annuncio',
username, now(), 'Nuovo Commento');

        end LOOP insert_new_uscc_warning;
        CLOSE cur_username_uscc_user;


END//
DELIMITER ;
-- ----------------------------------------------------------------
```

-- Avviso Commento Inserito ----------

```
DELIMITER //
DROP TRIGGER IF EXISTS BachecaElettronicadb.warn_comment_deleted ;
CREATE DEFINER = CURRENT_USER TRIGGER BachecaElettronicadb.warn_comment_deleted AFTER DELETE ON BachecaElettronicadb.Commento FOR EACH ROW
BEGIN
	declare username VARCHAR(45);
	declare loop_username INT DEFAULT 0;

	declare cur_username_ucc_user CURSOR FOR SELECT seguiti.UCC_Username FROM BachecaElettronicadb.Commento AS commento JOIN BachecaElettronicadb.`Seguito-UCC` AS seguiti ON commento.Annuncio_Codice = seguiti.Annuncio_Codice;
	declare cur_username_uscc_user CURSOR FOR SELECT seguiti.USCC_Username FROM BachecaElettronicadb.Commento AS commento JOIN BachecaElettronicadb.`Seguito-USCC` AS seguiti ON commento.Annuncio_Codice = seguiti.Annuncio_Codice;
	declare CONTINUE HANDLER FOR NOT FOUND SET loop_username = 1;


	/*NOTA
	 * L'idea del trigger è di soddisfare la regola aziendale di avvisare ogni utente
	 * che segue l'annuncio che è stato eliminato il commento. Viene eseguito un ciclo in modo da permettere
	 * di inserire per ogni utente il suo messaggio e sarà poi alla stored procedure la responsabilità
	 * di gestire i dati. Ho applicato una sorta di pattern GRASP in modo da avere che ogni utente
	 * ha il suo messaggio e non un unico messaggio dove è condiviso per tutti, quindi da avere un low
	 * coupling tra gli utenti (ogni row contentente il messaggio dipende solo dall'utente indirizzato)
	 */
```

```
-- Seleziona tutti gli utenti che seguono l'annuncio e inserisce una nuova entry in modo da notificare l'evento
OPEN cur_username_ucc_user;
insert_new_ucc_warning:  LOOP
                                        FETCH cur_username_ucc_user INTO username;

                                        IF loop_username = 1 THEN
                                                LEAVE insert_new_ucc_warning;
                                        end IF;

                                        INSERT INTO BachecaElettronicadb.Notifica(Codice, Tipo_Notifica, Username_Utente, Data, Messaggio)
                                        VALUES (OLD.ID, 'Commento Annuncio', username, now(), 'Commento Eliminato');

end LOOP insert_new_ucc_warning;
CLOSE cur_username_ucc_user;

SET loop_username = 0;
SET username = NULL;

OPEN cur_username_uscc_user;
insert_new_uscc_warning: LOOP
                                        FETCH cur_username_uscc_user INTO username;

                                        IF loop_username = 1 THEN
                                                LEAVE insert_new_uscc_warning;
                                        end IF;

                                        INSERT INTO BachecaElettronicadb.Notifica(Codice, Tipo_Notifica, Username_Utente, Data, Messaggio)
                                        VALUES (OLD.ID, 'Commento Annuncio', username, now(), 'Commento Eliminato');
```

```
        end LOOP insert_new_uscc_warning;
        CLOSE cur_username_uscc_user;


END//
DELIMITER ;
-- -----------------------------------------------------------------

-- Avviso Annuncio Modificato ---------------------------------------

DELIMITER //
DROP TRIGGER IF EXISTS BachecaElettronicadb.warn_ad_updated ;
CREATE DEFINER = CURRENT_USER TRIGGER BachecaElettronicadb.warn_ad_updated
AFTER UPDATE ON BachecaElettronicadb.Annuncio FOR EACH ROW
BEGIN
        declare username VARCHAR(45) DEFAULT NULL;
        declare loop_username INT DEFAULT 0;

        declare cur_username_ucc_user CURSOR FOR SELECT seguiti.UCC_Username FROM
BachecaElettronicadb.Annuncio AS annuncio JOIN BachecaElettronicadb.`Seguito-UCC` AS
seguiti ON annuncio.Codice = seguiti.Annuncio_Codice;
        declare cur_username_uscc_user CURSOR FOR SELECT seguiti.USCC_Username FROM
BachecaElettronicadb.Annuncio AS annuncio JOIN BachecaElettronicadb.`Seguito-USCC` AS
seguiti ON annuncio.Codice = seguiti.Annuncio_Codice;
        declare CONTINUE HANDLER FOR NOT FOUND SET loop_username = 1;


        /*NOTA
         * L'idea del trigger è di soddisfare la regola aziendale di avvisare ogni utente
         * che segue l'annuncio che è stato modificato. Viene eseguito un ciclo in modo da permettere
          * di inserire per ogni utente il suo messaggio e sarà poi alla stored procedure la
responsabilità
         * di gestire i dati. Ho applicato una sorta di pattern GRASP in modo da avere che ogni utente
```

```
        * ha il suo messaggio e non un unico messaggio dove è condiviso per tutti, quindi da avere un low
        * coupling tra gli utenti (ogni row contentente il messaggio dipende solo dall'utente indirizzato)
 5      */

        -- Seleziona tutti gli utenti che seguono l'annuncio e inserisce una nuova entry in modo da notificare l'evento
        OPEN cur_username_ucc_user;
10      insert_new_ucc_warning: LOOP
                                        FETCH cur_username_ucc_user INTO username;

                                        IF loop_username = 1 THEN
15                                          LEAVE insert_new_ucc_warning;
                                        end IF;

                                        INSERT INTO BachecaElettronicadb.Notifica(Codice, Tipo_Notifica, Username_Utente, Data, Messaggio)
20                                      VALUES (NEW.Codice, 'Annuncio', username, now(), 'Annuncio Modificato');

        end LOOP insert_new_ucc_warning;
        CLOSE cur_username_ucc_user;
25
        SET loop_username = 0;
        SET username = NULL;

        OPEN cur_username_uscc_user;
30      insert_new_uscc_warning: LOOP
                                        FETCH cur_username_uscc_user INTO username;

                                        IF loop_username = 1 THEN
```

```
                                        LEAVE insert_new_uscc_warning;
                                    end IF;

                                                INSERT INTO
5   BachecaElettronicadb.Notifica(Codice, Tipo_Notifica, Username_Utente, Data, Messaggio)
                                                VALUES (NEW.Codice, 'Annuncio', username,
    now(), 'Annuncio Modificato');

        end LOOP insert_new_uscc_warning;
10      CLOSE cur_username_uscc_user;



    END//
    DELIMITER ;
15  -- ----------------------------------------------------------------

    -- Controlla il Codice Fiscale ------------------------------------

    DELIMITER //
20  DROP TRIGGER IF EXISTS BachecaElettronicadb.check_tax_code ;
    CREATE DEFINER = CURRENT_USER TRIGGER BachecaElettronicadb.check_tax_code
    BEFORE INSERT ON BachecaElettronicadb.InformazioneAnagrafica FOR EACH ROW
    BEGIN

25      -- Controlla che il formato del codice fiscale sia valido
        if (NEW.CF not regexp'^[A-Z]{6}[0-9]{2}[A-Z][0-9]{2}[A-Z][0-9]{3}[A-Z]$') then
            signal sqlstate '45019' set message_text = 'Incorrect Tax Code';
        end if;

30  END//
    DELIMITER ;
    -- ----------------------------------------------------------------

    -- Controlla Commento Inserimento ---------------------------------
```

```
DELIMITER //
DROP TRIGGER IF EXISTS BachecaElettronicadb.check_commento_insert ;
CREATE DEFINER = CURRENT_USER TRIGGER
BachecaElettronicadb.check_commento_insert BEFORE INSERT ON
BachecaElettronicadb.Commento FOR EACH ROW
BEGIN
    -- Controlla che l'annuncio sia attivo
    if (not exists (SELECT Codice FROM BachecaElettronicadb.Annuncio WHERE
BachecaElettronicadb.Annuncio.Codice = NEW.Annuncio_Codice AND
BachecaElettronicadb.Annuncio.Stato='Attivo')) then
        signal sqlstate '45007' set message_text = 'Ad not active now';
    end if;

END//
DELIMITER ;
-- ----------------------------------------------------------------

-- Controlla Commento Cancellazione ---------------------------------

DELIMITER //
DROP TRIGGER IF EXISTS BachecaElettronicadb.check_commento_delete ;
CREATE DEFINER = CURRENT_USER TRIGGER
BachecaElettronicadb.check_commento_delete BEFORE DELETE ON
BachecaElettronicadb.Commento FOR EACH ROW
BEGIN
    -- Controlla che l'annuncio sia attivo
    if (not exists (SELECT Codice FROM BachecaElettronicadb.Annuncio WHERE
BachecaElettronicadb.Annuncio.Codice = OLD.Annuncio_Codice AND
BachecaElettronicadb.Annuncio.Stato='Attivo')) then
        signal sqlstate '45007' set message_text = 'Ad not active now';
    end if;

END//
```

```
DELIMITER ;
-- ----------------------------------------------------------------

-- Controlla Annuncio in Seguito UCC ---------------------------------

DELIMITER //
DROP TRIGGER IF EXISTS BachecaElettronicadb.check_annuncio_seguito_ucc ;
CREATE DEFINER = CURRENT_USER TRIGGER BachecaElettronicadb.check_annuncio_seguito_ucc AFTER INSERT ON BachecaElettronicadb.`Seguito-UCC` FOR EACH ROW
BEGIN

    -- Controlla che l'annuncio esista e sia attivo
    if (not exists (SELECT Codice FROM BachecaElettronicadb.Annuncio WHERE BachecaElettronicadb.Annuncio.Codice = NEW.Annuncio_Codice)) then
        signal sqlstate '45006' set message_text = 'Ad not found';
    end if;

    if (not exists (SELECT Codice FROM BachecaElettronicadb.Annuncio WHERE BachecaElettronicadb.Annuncio.Codice = NEW.Annuncio_Codice and BachecaElettronicadb.Annuncio.Stato = 'Attivo')) then
        signal sqlstate '45007' set message_text = 'Ad not active now';
    end if;

END//
DELIMITER ;
-- ----------------------------------------------------------------

-- Controlla Annuncio in Seguito USCC --------------------------------

DELIMITER //
DROP TRIGGER IF EXISTS BachecaElettronicadb.check_annuncio_seguito_uscc ;
```

```
CREATE DEFINER = CURRENT_USER TRIGGER BachecaElettronicadb.check_annuncio_seguito_uscc AFTER INSERT ON BachecaElettronicadb.`Seguito-USCC` FOR EACH ROW
BEGIN

    -- Controlla che l'annuncio esista e sia attivo
    if (not exists (SELECT Codice FROM BachecaElettronicadb.Annuncio WHERE BachecaElettronicadb.Annuncio.Codice = NEW.Annuncio_Codice)) then
        signal sqlstate '45006' set message_text = 'Ad not found';
    end if;

    if (not exists (SELECT Codice FROM BachecaElettronicadb.Annuncio WHERE BachecaElettronicadb.Annuncio.Codice = NEW.Annuncio_Codice and BachecaElettronicadb.Annuncio.Stato = 'Attivo')) then
        signal sqlstate '45007' set message_text = 'Ad not active now';
    end if;

END//
DELIMITER ;
-- ----------------------------------------------------------------
```

Eventi

Nessun evento previsto.

Viste

Nessuna vista prevista.

Stored Procedures e transazioni

- Inserisce un nuovo utente UCC nel database;

```
-- Registrazione di un utente UCC ------------------------
DELIMITER //
DROP PROCEDURE IF EXISTS BachecaElettronicadb.registra_utente_UCC ;
CREATE PROCEDURE BachecaElettronicadb.registra_utente_UCC (IN username VARCHAR(45), IN password VARCHAR(45), IN numeroCarta VARCHAR(16), IN dataScadenza
```

```
DATE, IN cvc INT, IN cf_anagrafico VARCHAR(16), IN cognome VARCHAR(20), IN nome VARCHAR(20), IN indirizzoDiResidenza VARCHAR(20), IN cap INT, IN indirizzoDiFatturazione VARCHAR(20), IN tipoRecapitoPreferito VARCHAR(20), IN recapitoPreferito VARCHAR(40), IN tipoRecapitoNonPreferito VARCHAR(20), IN recapitoNonPreferito VARCHAR(40))
BEGIN
    declare idStorico INT;

    call BachecaElettronicadb.inserisci_Storico(idStorico);
    call BachecaElettronicadb.inserimentoInfoAnagrafiche(cf_anagrafico, cognome, nome, indirizzoDiResidenza, cap, indirizzoDiFatturazione, tipoRecapitoPreferito, recapitoPreferito);
    call BachecaElettronicadb.modifica_RecapitoNonPreferito(tipoRecapitoNonPreferito, recapitoNonPreferito, cf_anagrafico, null);

    INSERT INTO BachecaElettronicadb.UCC (Username, Password, NumeroCarta, DataScadenza, CVC, StoricoConversazione_ID, CF_Anagrafico)
    VALUES(username, md5(password), numeroCarta, dataScadenza, cvc, idStorico, cf_anagrafico);

END//
DELIMITER ;
-- ---------------------------------------------------------------------
```

- Questa procedure ha la responsabilità di inserire un nuovo Storico nel momento della registrazione di un utente. È stato scelto un livello di isolamento per evitare meno errori possibili con la funzione LAST_INSERT_ID() che potrebbe creare letture errate in caso di un grande traffico nel sistema.

```
-- Inserimento di uno Storico Conversazioni ------------------------
DELIMITER //
DROP PROCEDURE IF EXISTS BachecaElettronicadb.inserisci_Storico ;
CREATE PROCEDURE BachecaElettronicadb.inserisci_Storico (OUT id INT)
BEGIN
    declare exit handler for sqlexception
    begin
        ROLLBACK; -- rollback any changes made in the transaction
        RESIGNAL; -- raise again the sql exceptionto the caller
```

```
        end;

        /*NOTA
         * Ho scelto questo livello perchè voglio che non ci siano letture
         * sporche sul valore dell'ID in caso concorrenza su questa procedura.
         * Il trucco è quello di usare il lock preso dalla write, che verrà restituito al commit.
         */

        SET transaction isolation level read committed;
        start transaction;
                INSERT INTO BachecaElettronicadb.StoricoConversazione (ID) VALUES(NULL);
                SET id = LAST_INSERT_ID();
        commit;

END//
DELIMITER ;
-- -----------------------------------------------------------------
```

- Inserisce o Rimuove un recapito non preferito a seconda del parametro “rimuovi”

```
-- Inserimento o Rimozione Recapito non preferito ----------

DELIMITER //
DROP PROCEDURE IF EXISTS BachecaElettronicadb.modifica_RecapitoNonPreferito ;
CREATE PROCEDURE BachecaElettronicadb.modifica_RecapitoNonPreferito (IN tipo VARCHAR(20), IN recapito VARCHAR(40), IN cf_anagrafico VARCHAR(16), IN rimuovi INT)
BEGIN

        if (not exists (SELECT CF FROM BachecaElettronicadb.InformazioneAnagrafica WHERE BachecaElettronicadb.InformazioneAnagrafica.CF = cf_anagrafico))  then
                signal sqlstate '45004' set message_text = "Tax code not found";
        end if;

        if(rimuovi is null) then
                INSERT INTO BachecaElettronicadb.RecapitoNonPreferito (Recapito, Tipo, InformazioneAnagrafica_CF) VALUES(recapito, tipo, cf_anagrafico);
```

```
        else
                DELETE FROM BachecaElettronicadb.RecapitoNonPreferito
                WHERE BachecaElettronicadb.RecapitoNonPreferito.InformazioneAnagrafica_CF = cf_anagrafico AND BachecaElettronicadb.RecapitoNonPreferito.Recapito = recapito AND BachecaElettronicadb.RecapitoNonPreferito.Tipo = tipo;
        end if;

END//
DELIMITER ;
-- ----------------------------------------------------------------
```

- Visualizza tutti i contatti dell'utente

```
-- Visualizzazione contatti ----------------------------------------

DELIMITER //
DROP PROCEDURE IF EXISTS BachecaElettronicadb.visualizza_contatti ;
CREATE PROCEDURE BachecaElettronicadb.visualizza_contatti (IN cf_anagrafico VARCHAR(16), IN preferito INT)
BEGIN

        declare exit handler for sqlexception
        begin
                ROLLBACK; -- rollback any changes made in the transaction
                RESIGNAL; -- raise again the sql exceptionto the caller
        end;

        /*NOTA
         * Ho scelto questo livello per evitare letture inconsistenti sugli stessi dati
         */

        SET transaction read only;
        SET transaction isolation level repeatable read;
        start transaction;
```

```
        if (not exists (SELECT CF FROM BachecaElettronicadb.InformazioneAnagrafica WHERE BachecaElettronicadb.InformazioneAnagrafica.CF = cf_anagrafico)) then
            signal sqlstate '45004' set message_text = 'Tax code not found';
        end if;

        if (preferito is not null) then
            SELECT RecapitoPreferito AS 'Recapito', tipoRecapitoPreferito AS 'Tipo', CF AS 'Codice Fiscale'
            FROM BachecaElettronicadb.InformazioneAnagrafica
            WHERE BachecaElettronicadb.InformazioneAnagrafica.CF=cf_anagrafico;
        else
            SELECT Recapito, Tipo, InformazioneAnagrafica_CF AS 'Codice Fiscale'
            FROM BachecaElettronicadb.RecapitoNonPreferito
            WHERE BachecaElettronicadb.RecapitoNonPreferito.InformazioneAnagrafica_CF=cf_anagrafico;
        end if;

    commit;

END//
DELIMITER ;
-- -----------------------------------------------------------------
```

- Inserisce un nuovo utente USCC nel database

```
-- Registrazione di un utente USCC ----------------------------------
DELIMITER //
DROP PROCEDURE IF EXISTS BachecaElettronicadb.registra_utente_USCC ;
CREATE PROCEDURE BachecaElettronicadb.registra_utente_USCC (IN username VARCHAR(45), IN password VARCHAR(45), IN cf_anagrafico VARCHAR(16), IN cognome VARCHAR(20), IN nome VARCHAR(20), IN indirizzoDiResidenza VARCHAR(20), IN cap INT, IN indirizzoDiFatturazione VARCHAR(20), IN tipoRecapitoPreferito VARCHAR(20), IN recapitoPreferito VARCHAR(40), IN tipoRecapitoNonPreferito VARCHAR(20), IN recapitoNonPreferito VARCHAR(40))
BEGIN
    declare idStorico INT;
```

```
        call BachecaElettronicadb.inserisci_Storico(idStorico);
        call BachecaElettronicadb.inserimentoInfoAnagrafiche(cf_anagrafico, cognome, nome, indirizzoDiResidenza, cap, indirizzoDiFatturazione, tipoRecapitoPreferito, recapitoPreferito);
        call BachecaElettronicadb.modifica_RecapitoNonPreferito(tipoRecapitoNonPreferito, recapitoNonPreferito, cf_anagrafico, null);

        INSERT INTO BachecaElettronicadb.USCC (Username, Password, StoricoConversazione_ID, CF_Anagrafico)
        VALUES(username, md5(password), idStorico, cf_anagrafico);

END//
DELIMITER ;
-- -----------------------------------------------------------------
```

- Inserisce le informazioni Anagrafiche dell'utente. Viene usata nel momento della registrazione e nessun utente ha i privilegi di eseguirla direttamente

```
-- Inserimento delle Informazioni Anagrafiche ------------------------
DELIMITER //
DROP PROCEDURE IF EXISTS BachecaElettronicadb.inserimentoInfoAnagrafiche ;
CREATE PROCEDURE BachecaElettronicadb.inserimentoInfoAnagrafiche (IN cf VARCHAR(16), IN cognome VARCHAR(20), IN nome VARCHAR(20), IN indirizzoDiResidenza VARCHAR(20), IN cap INT, IN indirizzoDiFatturazione VARCHAR(20), IN tipoRecapitoPreferito VARCHAR(20), IN recapitoPreferito VARCHAR(40))
BEGIN

        INSERT INTO BachecaElettronicadb.InformazioneAnagrafica (CF, Cognome, Nome, IndirizzoDiResidenza, CAP, IndirizzoDiFatturazione, tipoRecapitoPreferito, RecapitoPreferito)
        VALUES(cf, cognome, nome, indirizzoDiResidenza, cap, indirizzoDiFatturazione, tipoRecapitoPreferito, recapitoPreferito);

END//
DELIMITER ;
-- -----------------------------------------------------------------
```

- Implementata per soddisfare alla regola aziendale in cui tutti gli utenti, che seguono un annuncio, devono essere notificati di ogni loro nuova modifica

```
-- Visualizzazione nuove notifiche ----------------------------------
DELIMITER //
DROP PROCEDURE IF EXISTS BachecaElettronicadb.visualizza_notifiche ;
CREATE PROCEDURE BachecaElettronicadb.visualizza_notifiche (IN username VARCHAR(45))
BEGIN

    declare exit handler for sqlexception
    begin
        ROLLBACK; -- rollback any changes made in the transaction
        RESIGNAL; -- raise again the sql exceptionto the caller
    end;

    /*NOTA
     * Ho scelto questo livello per evitare le letture sporche
     */

    SET transaction isolation level read committed;
    start transaction;

        /*NOTA
         * Questa stored procedure è utilizzata per mostrare all'utente, che lo ha chiamato,
         * le sue nuove notifiche. Applica un concetto simile ad una coda di messaggi
         * dove è possibile fare un'estrazione dei messaggi richiesti ed infine eliminarli per
         * non  essere visualizzati di nuovo. È correlata ai trigger precedentemente descritti.
         */

        -- Mostra le nuove notifiche rimaste in coda
        SELECT Codice, Tipo_Notifica, Data, Messaggio
        FROM BachecaElettronicadb.Notifica
        WHERE BachecaElettronicadb.Notifica.Username_Utente = username
        ORDER BY Data ASC;
```

```
        -- Elimina le notifiche visualizzate
        DELETE FROM BachecaElettronicadb.Notifica
        WHERE BachecaElettronicadb.Notifica.Username_Utente = username;

    commit;

END//
DELIMITER ;
-- ----------------------------------------------------------------
```

- Modifica le informazioni anagrafiche a seconda dei parametri NULL e NOT NULL

```
-- Modifica delle Informazioni Anagrafiche --------------------------
DELIMITER //
DROP PROCEDURE IF EXISTS BachecaElettronicadb.modificaInfoAnagrafiche ;
CREATE PROCEDURE BachecaElettronicadb.modificaInfoAnagrafiche (IN cf VARCHAR(16), IN cognome VARCHAR(20), IN nome VARCHAR(20), IN indirizzoDiResidenza VARCHAR(20), IN cap INT, IN indirizzoDiFatturazione VARCHAR(20), IN tipoRecapitoPreferito VARCHAR(20), IN recapitoPreferito VARCHAR(40))
BEGIN

    IF cognome IS NOT NULL THEN
        UPDATE BachecaElettronicadb.InformazioneAnagrafica SET BachecaElettronicadb.InformazioneAnagrafica.Cognome=cognome WHERE BachecaElettronicadb.InformazioneAnagrafica.CF=cf;
    END IF;
    IF nome IS NOT NULL THEN
        UPDATE BachecaElettronicadb.InformazioneAnagrafica SET BachecaElettronicadb.InformazioneAnagrafica.Nome=nome WHERE BachecaElettronicadb.InformazioneAnagrafica.CF=cf;
    END IF;
    IF indirizzoDiResidenza IS NOT NULL THEN
        UPDATE BachecaElettronicadb.InformazioneAnagrafica SET BachecaElettronicadb.InformazioneAnagrafica.IndirizzoDiResidenza=indirizzoDiResidenza
WHERE BachecaElettronicadb.InformazioneAnagrafica.CF=cf;
```

```
        END IF;
        IF indirizzoDiFatturazione IS NOT NULL THEN
            UPDATE BachecaElettronicadb.InformazioneAnagrafica SET
BachecaElettronicadb.InformazioneAnagrafica.IndirizzoDiFatturazione=indirizzoDiFatturazione
WHERE BachecaElettronicadb.InformazioneAnagrafica.CF=cf;
        END IF;
        IF cap IS NOT NULL THEN
            UPDATE BachecaElettronicadb.InformazioneAnagrafica SET
BachecaElettronicadb.InformazioneAnagrafica.CAP=cap,
IndirizzoDiFatturazione=indirizzoDiFatturazione WHERE
BachecaElettronicadb.InformazioneAnagrafica.CF=cf;
        END IF;
        IF ((tipoRecapitoPreferito IS NOT NULL) AND (recapitoPreferito IS NOT NULL)) THEN
            UPDATE BachecaElettronicadb.InformazioneAnagrafica SET
BachecaElettronicadb.InformazioneAnagrafica.TipoRecapitoPreferito = tipoRecapitoPreferito
WHERE BachecaElettronicadb.InformazioneAnagrafica.CF = cf;
            UPDATE BachecaElettronicadb.InformazioneAnagrafica SET
BachecaElettronicadb.InformazioneAnagrafica.RecapitoPreferito = recapitoPreferito WHERE
BachecaElettronicadb.InformazioneAnagrafica.CF = cf;
        END IF;

END//
DELIMITER ;
-- ----------------------------------------------------------------

-- Visualizzazione delle Informazioni Anagrafiche ----------
DELIMITER //
DROP PROCEDURE IF EXISTS BachecaElettronicadb.visualizzaInfoAnagrafiche ;
CREATE PROCEDURE BachecaElettronicadb.visualizzaInfoAnagrafiche (IN cf VARCHAR(16),
IN check_owner INT, IN username VARCHAR(45))
BEGIN

        declare exit handler for sqlexception
        begin
```

```
            ROLLBACK; -- rollback any changes made in the transaction
            RESIGNAL; -- raise again the sql exceptionto the caller
        end;

        /*NOTA
         * Ho scelto questo livello per evitare le letture sporche
         */

        SET transaction read only;
        SET transaction isolation level read committed;
        start transaction;

            if ((check_owner IS NOT NULL) AND (username IS NOT NULL)) then
                IF (not exists (SELECT Username FROM BachecaElettronicadb.UCC WHERE BachecaElettronicadb.UCC.CF_Anagrafico = cf AND BachecaElettronicadb.UCC.Username = username)) then
                    if (not exists (SELECT Username FROM BachecaElettronicadb.USCC WHERE BachecaElettronicadb.USCC.CF_Anagrafico = cf AND BachecaElettronicadb.USCC.Username = username)) then
                        signal sqlstate '45009' set message_text = 'The Tax code is not yours';
                    end if;
                end IF;
            end if;

            SELECT CF AS 'Codice Fiscale', Cognome, Nome, IndirizzoDiResidenza AS 'Indirizzo di Residenza', CAP, IndirizzoDiFatturazione AS 'Indirizzo di Fatturazione', tipoRecapitoPreferito AS 'Tipo Recapito', RecapitoPreferito AS 'Recapito Preferito'
            FROM BachecaElettronicadb.InformazioneAnagrafica
            WHERE BachecaElettronicadb.InformazioneAnagrafica.CF = cf;

        commit;

    END//
```

```
DELIMITER ;
-- ----------------------------------------------------------------

-- Inserimento di una nuova categoria --------------------------------
DELIMITER //
DROP PROCEDURE IF EXISTS BachecaElettronicadb.inserimentoNuovaCategoria ;
CREATE PROCEDURE BachecaElettronicadb.inserimentoNuovaCategoria (IN nome VARCHAR(20))
BEGIN

        INSERT INTO BachecaElettronicadb.Categoria (Nome) VALUES(nome);

END//
DELIMITER ;
-- ----------------------------------------------------------------

-- Visualizzazione categoria ---------------------------------------
DELIMITER //
DROP PROCEDURE IF EXISTS BachecaElettronicadb.visualizzaCategoria ;
CREATE PROCEDURE BachecaElettronicadb.visualizzaCategoria ()
BEGIN
        declare exit handler for sqlexception
        begin
                ROLLBACK; -- rollback any changes made in the transaction
                RESIGNAL; -- raise again the sql exceptionto the caller
        end;

        /*NOTA
         * Ho scelto questo livello per evitare le letture sporche
         * in caso di inserimento di una nuova categoria ma non andato a buon fine
         */

        SET transaction read only;
        SET transaction isolation level read committed;
```

```
        start transaction;

                SELECT Nome
                FROM BachecaElettronicadb.Categoria;

        commit;

    END//
    DELIMITER ;
    -- -----------------------------------------------------------------

    -- Inserimento di un nuovo annuncio ---------------------------------
    DELIMITER //
    DROP PROCEDURE IF EXISTS BachecaElettronicadb.inserimentoNuovoAnnuncio ;
    CREATE PROCEDURE BachecaElettronicadb.inserimentoNuovoAnnuncio (IN descrizione VARCHAR(100), IN importo INT, IN foto VARCHAR(9), IN ucc_username VARCHAR(45), IN categoria_nome VARCHAR(20))
    BEGIN

        if (foto IS NULL) then
                INSERT INTO BachecaElettronicadb.Annuncio (Codice, Stato, Descrizione, Importo, UCC_Username, Categoria_Nome) VALUES(NULL, 'Attivo', descrizione, importo, ucc_username, categoria_nome);
        else
                INSERT INTO BachecaElettronicadb.Annuncio (Codice, Stato, Descrizione, Importo, Foto, UCC_Username, Categoria_Nome) VALUES(NULL, 'Attivo', descrizione, importo, 'Presente', ucc_username, categoria_nome);
        end if;

    END//
    DELIMITER ;
    -- -----------------------------------------------------------------

    -- Inserimento o Rimozione di una foto in annuncio -------------------
```

```
DELIMITER //
DROP PROCEDURE IF EXISTS BachecaElettronicadb.modificaFotoAnnuncio ;
CREATE PROCEDURE BachecaElettronicadb.modificaFotoAnnuncio (IN codice INT, IN username VARCHAR(45), IN foto VARCHAR(9))
BEGIN
	declare exit handler for sqlexception
	begin
		ROLLBACK; -- rollback any changes made in the transaction
		RESIGNAL; -- raise again the sql exceptionto the caller
	end;

	/*NOTA
	 * Ho scelto questo livello per evitare le letture inconsistenti
	 * e per restituire il lock al commit
	 */

	SET transaction isolation level repeatable read;
	start transaction;

		if (not exists (SELECT Codice FROM BachecaElettronicadb.Annuncio WHERE BachecaElettronicadb.Annuncio.Codice = codice)) then
			signal sqlstate '45006' set  message_text = 'Ad not found';
		end if;

		if (not exists (SELECT Codice FROM BachecaElettronicadb.Annuncio WHERE BachecaElettronicadb.Annuncio.Codice = codice AND BachecaElettronicadb.Annuncio.UCC_Username = username)) then
			signal sqlstate '45008' set  message_text = 'You are not the owner of the ad';
		end if;

		if (not exists (SELECT Codice FROM BachecaElettronicadb.Annuncio WHERE BachecaElettronicadb.Annuncio.Codice = codice AND BachecaElettronicadb.Annuncio.Stato = 'Attivo')) then
			signal sqlstate '45007' set  message_text = 'Ad not active now';
```

```
                end if;

                if (foto is not NULL) then
                        UPDATE BachecaElettronicadb.Annuncio
                        SET BachecaElettronicadb.Annuncio.Foto = 'Presente'
                        WHERE BachecaElettronicadb.Annuncio.Codice = codice;
                else
                        UPDATE BachecaElettronicadb.Annuncio
                        SET BachecaElettronicadb.Annuncio.Foto = NULL
                        WHERE BachecaElettronicadb.Annuncio.Codice = codice;
                end if;
        commit;

END//
DELIMITER ;
-- ----------------------------------------------------------------

-- Visualizza annuncio ----------------------------------------------
DELIMITER //
DROP PROCEDURE IF EXISTS BachecaElettronicadb.visualizzaAnnuncio ;
CREATE PROCEDURE BachecaElettronicadb.visualizzaAnnuncio (IN annuncio_codice INT, IN
username VARCHAR(45), IN check_owner INT)
BEGIN
        declare exit handler for sqlexception
        begin
                ROLLBACK; -- rollback any changes made in the transaction
                RESIGNAL; -- raise again the sql exceptionto the caller
        end;

        /*NOTA
         * Ho scelto questo livello per evitare le letture inconsistenti sullo stesso dato
         */

        SET transaction read only;
```

```
SET transaction isolation level repeatable read;
start transaction;

    if ((username IS NOT NULL) AND not exists (SELECT Username FROM
5 BachecaElettronicadb.UCC WHERE BachecaElettronicadb.UCC.Username=username)) then
        signal sqlstate '45000' set message_text = 'User not found';
    end if;

    if ((annuncio_codice IS NOT NULL) AND not exists (SELECT Codice FROM
10 BachecaElettronicadb.Annuncio WHERE
BachecaElettronicadb.Annuncio.Codice=annuncio_codice)) then
        signal sqlstate '45006' set message_text = 'Ad not found';
    end if;

15  if ((check_owner = 1) AND (username IS NOT NULL) AND (annuncio_codice IS
NOT NULL)) then
        IF (not exists (SELECT Codice FROM BachecaElettronicadb.Annuncio
WHERE BachecaElettronicadb.Annuncio.Codice=annuncio_codice AND
BachecaElettronicadb.Annuncio.UCC_Username=username)) then
20          signal sqlstate '45008' set message_text = 'You are not the owner of the
ad';
        end IF;
    end if;

25  IF ((annuncio_codice IS NOT NULL) AND (username IS NOT NULL)) THEN
        SELECT Codice, Stato, Descrizione, Importo, Foto, UCC_Username AS
'Proprietario', Categoria_Nome AS 'Categoria'
        FROM BachecaElettronicadb.Annuncio
        WHERE BachecaElettronicadb.Annuncio.UCC_Username=username AND
30 BachecaElettronicadb.Annuncio.Codice=annuncio_codice AND
BachecaElettronicadb.Annuncio.Stato='Attivo';
    ELSEIF ((annuncio_codice IS NULL) AND (username IS NOT NULL)) THEN
        SELECT Codice, Stato, Descrizione, Importo, Foto, UCC_Username AS
'Proprietario', Categoria_Nome AS 'Categoria'
```

```
                FROM BachecaElettronicadb.Annuncio
                WHERE BachecaElettronicadb.Annuncio.UCC_Username=username AND BachecaElettronicadb.Annuncio.Stato='Attivo';
        ELSEIF ((annuncio_codice IS NOT NULL) AND (username IS NULL)) THEN
            SELECT Codice, Stato, Descrizione, Importo, Foto, UCC_Username AS 'Proprietario', Categoria_Nome AS 'Categoria'
                FROM BachecaElettronicadb.Annuncio
                WHERE BachecaElettronicadb.Annuncio.Codice=annuncio_codice AND BachecaElettronicadb.Annuncio.Stato='Attivo';
        ELSE
            SELECT Codice, Stato, Descrizione, Importo, Foto, UCC_Username AS 'Proprietario', Categoria_Nome AS 'Categoria'
                FROM BachecaElettronicadb.Annuncio
                WHERE BachecaElettronicadb.Annuncio.Stato='Attivo';
        end IF;

    commit;

END//
DELIMITER ;
-- -----------------------------------------------------------------

-- Inserimento o Rimozione commento ----------------------------------
DELIMITER //
DROP PROCEDURE IF EXISTS BachecaElettronicadb.modificaCommento ;
CREATE PROCEDURE BachecaElettronicadb.modificaCommento (IN testo VARCHAR(45), IN annuncio_codice INT, IN ID_rimozione_commento INT)
BEGIN

    if (ID_rimozione_commento is null) then
        INSERT INTO BachecaElettronicadb.Commento (Testo, Annuncio_Codice) VALUES(testo, annuncio_codice);
    else
        DELETE FROM BachecaElettronicadb.Commento
```

```
            WHERE BachecaElettronicadb.Commento.ID = ID_rimozione_commento;
        end if;

END//
DELIMITER ;
-- ----------------------------------------------------------------

-- Visualizzazione commento ----------------------------------------
DELIMITER //
DROP PROCEDURE IF EXISTS BachecaElettronicadb.visualizzaCommento ;
CREATE PROCEDURE BachecaElettronicadb.visualizzaCommento (IN annuncio_codice INT)
BEGIN
        declare exit handler for sqlexception
        begin
            ROLLBACK; -- rollback any changes made in the transaction
            RESIGNAL; -- raise again the sql exceptionto the caller
        end;

        /*NOTA
         * Ho scelto questo livello per evitare le letture sporche
         * in caso di inserimento di un nuovo commento ma ancora non committato
         */

        SET transaction read only;
        SET transaction isolation level read committed;
        start transaction;

            if (not exists (SELECT Codice FROM BachecaElettronicadb.Annuncio WHERE
BachecaElettronicadb.Annuncio.Codice=annuncio_codice AND
BachecaElettronicadb.Annuncio.Stato='Attivo')) then
                signal sqlstate '45007' set message_text = 'Ad not active now';
            end if;

            SELECT ID, Testo, Annuncio_codice AS "Codice dell'annuncio"
```

```
        FROM BachecaElettronicadb.Commento
        WHERE BachecaElettronicadb.Commento.Annuncio_Codice = annuncio_codice;

    commit;

END//
DELIMITER ;
-- ----------------------------------------------------------------

-- Inserimento Conversazione ----------------------------------------
DELIMITER //
DROP PROCEDURE IF EXISTS BachecaElettronicadb.inserimentoConversazione ;
CREATE PROCEDURE BachecaElettronicadb.inserimentoConversazione (OUT id_conversazione INT, IN ucc_username_1 VARCHAR(45), ucc_username_2 VARCHAR(45), IN uscc_username VARCHAR(45))
BEGIN
    declare exit handler for sqlexception
    begin
        ROLLBACK; -- rollback any changes made in the transaction
        RESIGNAL; -- raise again the sql exceptionto the caller
    end;

    /*NOTA
     * Ho scelto questo livello per evitare concorrenza sull'ID, usando il lock
     * ottenuto dal write che sarà poi rilasciato al commit.
     */

    SET transaction isolation level read committed;
    start transaction;

        INSERT INTO BachecaElettronicadb.Conversazione (Codice, UCC_Username_1, UCC_Username_2, USCC_Username) VALUES(NULL, ucc_username_1, ucc_username_2, uscc_username);
        SET id_conversazione = LAST_INSERT_ID();
```

```
        commit;

    END//
    DELIMITER ;
    -- -----------------------------------------------------------------
```

- Seleziona lo storico di un utente UCC che viene utilizzata per tracciare le conversazioni

```
    -- Seleziona Storico UCC per la stored procedure ----------
    DELIMITER //
    DROP FUNCTION IF EXISTS BachecaElettronicadb.seleziona_Storico_UCC ;
    CREATE FUNCTION BachecaElettronicadb.seleziona_Storico_UCC (username VARCHAR(45))
    RETURNS INT DETERMINISTIC
    BEGIN
        declare storico_id_ucc INT;

        -- Prende il codice dello storico dell'utente
        SELECT StoricoConversazione_ID INTO storico_id_ucc
        FROM BachecaElettronicadb.UCC
        WHERE BachecaElettronicadb.UCC.Username = username;

        RETURN storico_id_ucc;

    END//
    DELIMITER ;
    -- -----------------------------------------------------------------
```

- Seleziona lo storico di un utente USCC che viene utilizzata per tracciare le conversazioni

```
    -- Seleziona Storico USCC per la stored procedure ----------
    DELIMITER //
    DROP FUNCTION IF EXISTS BachecaElettronicadb.seleziona_Storico_USCC ;
    CREATE FUNCTION BachecaElettronicadb.seleziona_Storico_USCC (username VARCHAR(45))
    RETURNS INT DETERMINISTIC
    BEGIN
```

```
        declare storico_id_uscc INT;

        -- Prende il codice dello storico dell'utente
        SELECT StoricoConversazione_ID INTO storico_id_uscc
        FROM BachecaElettronicadb.USCC
        WHERE BachecaElettronicadb.USCC.Username = username;

        RETURN storico_id_uscc;

END//
DELIMITER ;
-- -----------------------------------------------------------------
```

- Traccia le nuove conversazioni per gli utenti. Essa non viene usata direttamente dall'utente ma indirettamente tramite la stored procedure "invioMessaggio"

```
-- Tracciamento Conversazioni ---------------------------------------
DELIMITER //
DROP PROCEDURE IF EXISTS BachecaElettronicadb.traccia_Conversazione ;
CREATE PROCEDURE BachecaElettronicadb.traccia_Conversazione (IN conversazione_codice INT, IN sender_ucc_username VARCHAR(45), IN receiver_ucc_username VARCHAR(45), IN receiver_uscc_username VARCHAR(45))
BEGIN

        declare storico_id_ucc, storico_id_uscc INT;

        if ((sender_ucc_username is not null) and (receiver_uscc_username is not null)) then
                SET storico_id_ucc = seleziona_Storico_UCC(sender_ucc_username);
                SET storico_id_uscc = seleziona_Storico_USCC(receiver_uscc_username);

                INSERT INTO BachecaElettronicadb.Tracciato (Conversazione_Codice, StoricoConversazione_ID) VALUES(conversazione_codice, storico_id_ucc);      -- traccia lo storico di UCC
                INSERT INTO BachecaElettronicadb.Tracciato (Conversazione_Codice, StoricoConversazione_ID) VALUES(conversazione_codice, storico_id_uscc);      -- traccia lo storico di USCC
```

```
            INSERT INTO BachecaElettronicadb.ConversazioneCodice (CodiceConv, StoricoConversazione_ID) VALUES(conversazione_codice, storico_id_ucc); -- inserisce il nuovo codice della conversazione nello storico di UCC
            INSERT INTO BachecaElettronicadb.ConversazioneCodice (CodiceConv, StoricoConversazione_ID) VALUES(conversazione_codice, storico_id_uscc); -- inserisce il nuovo codice della conversazione nello storico di USCC

        elseif ((sender_ucc_username is not null) and (receiver_ucc_username is not null)) then
            SET storico_id_ucc = seleziona_Storico_UCC(sender_ucc_username);
            SET storico_id_uscc = seleziona_Storico_UCC(receiver_ucc_username);

            INSERT INTO BachecaElettronicadb.Tracciato (Conversazione_Codice, StoricoConversazione_ID) VALUES(conversazione_codice, storico_id_ucc); -- traccia lo storico di UCC
            INSERT INTO BachecaElettronicadb.Tracciato (Conversazione_Codice, StoricoConversazione_ID) VALUES(conversazione_codice, storico_id_uscc); -- traccia lo storico di UCC
            INSERT INTO BachecaElettronicadb.ConversazioneCodice (CodiceConv, StoricoConversazione_ID) VALUES(conversazione_codice, storico_id_ucc); -- inserisce il nuovo codice della conversazione nello storico di UCC
            INSERT INTO BachecaElettronicadb.ConversazioneCodice (CodiceConv, StoricoConversazione_ID) VALUES(conversazione_codice, storico_id_uscc); -- inserisce il nuovo codice della conversazione nello storico di UCC
        else
            signal sqlstate '45010' set message_text = 'Error Parameters';
        end if;

    END//
    DELIMITER ;
    -- ----------------------------------------------------------------
```

- Utilizzata per controllare di quale tipo è l’utente (UCC oppure USCC)

```
    -- Controlla il tipo di utente ------------------------------------
    DELIMITER //
    DROP FUNCTION IF EXISTS BachecaElettronicadb.controlla_utente ;
```

```
CREATE FUNCTION BachecaElettronicadb.controlla_utente (username VARCHAR(45)) RETURNS VARCHAR(4) DETERMINISTIC
BEGIN

    if (exists (SELECT Username FROM BachecaElettronicadb.UCC WHERE BachecaElettronicadb.UCC.Username = username)) then
        RETURN 'UCC';

    elseif (exists (SELECT Username FROM BachecaElettronicadb.USCC WHERE BachecaElettronicadb.USCC.Username = username)) then
        RETURN 'USCC';

    else
        signal sqlstate '45000' set message_text = 'User not found';  -- utente non trovato nel database
    end if;

END//
DELIMITER ;
-- ----------------------------------------------------------------
```

- Questa è una delle stored procedure più complicate. Il suo compito è di far comunicare due utenti e creare nuove conversazioni se questi non hanno avuto mai un contatto. È stata progettata in modo da non dare tutta la responsabilità solo ad essa ma separare i compiti con altre stored procedures o functions di appoggio. Il livello di isolamento scelto è stato repeatable read visto che si eseguono molte letture sullo stesso dato e quindi si cerca di non avere inconsistenze sui dat

```
-- Invio messaggio da parte utente ----------------------------------
DELIMITER //
DROP PROCEDURE IF EXISTS BachecaElettronicadb.invioMessaggio ;
CREATE PROCEDURE BachecaElettronicadb.invioMessaggio (IN sender_username VARCHAR(45), IN receiver_username VARCHAR(45), IN testo VARCHAR(100))
BEGIN
    declare conversazione_codice INT DEFAULT NULL;
    declare type_user_sender VARCHAR(4);
```

```
declare type_user_receiver VARCHAR(4);
declare sender_is_ucc, sender_is_uscc, receiver_is_ucc, receiver_is_uscc INT;

declare exit handler for sqlexception
BEGIN
        rollback; -- rollback any changes made in the transaction
        resignal; -- raise again the sql exceptionto the caller
END;

/*NOTA
 * Ho scelto questo livello per evitare letture
 * inconsistenti sull'utente nel database, vista la necessita di fare
 * più di una lettura sullo stesso dato
 */

SET transaction isolation level repeatable read;
start transaction;

        if (sender_username = receiver_username) then
                signal sqlstate '45010' set message_text = 'Error parameters';
        end if;

        SET sender_is_ucc = 0;
        SET sender_is_uscc = 0;
        SET receiver_is_ucc = 0;
        SET receiver_is_uscc = 0;

        SET type_user_sender = controlla_utente(sender_username);                    --
Restituisce la natura dell'utente che invia il messaggio
        SET type_user_receiver = controlla_utente(receiver_username);          --
Restituisce la natura dell'utente che riceve il messaggio

        -- Seleziona gli username dei due utenti
        if (type_user_sender = 'UCC') then
```

```
                SET sender_is_ucc = 1;

                SELECT Username INTO sender_username
                FROM BachecaElettronicadb.UCC
                WHERE BachecaElettronicadb.UCC.Username = sender_username;
        else
                SET sender_is_uscc = 1;

                SELECT Username INTO sender_username
                FROM BachecaElettronicadb.USCC
                WHERE BachecaElettronicadb.USCC.Username = sender_username;
        end if;

        if (type_user_receiver = 'UCC') then
                SET receiver_is_ucc = 1;

                SELECT Username INTO receiver_username
                FROM BachecaElettronicadb.UCC
                WHERE BachecaElettronicadb.UCC.Username = receiver_username;
        else
                SET receiver_is_uscc = 1;

                SELECT Username INTO receiver_username
                FROM BachecaElettronicadb.USCC
                WHERE BachecaElettronicadb.USCC.Username = receiver_username;
        end if;

        if ((sender_is_uscc = 1) AND (receiver_is_uscc = 1)) then
                signal sqlstate '45013' set message_text = 'You cannot write to another USCC
user';
        end if;

        commit; -- Inserisco un commit onde evitare un dead lock con le altre query e per non
ottenere l'errore: Error 1568 (25001)
```

```
SET conversazione_codice = controllo_conversazione(NULL, sender_username, receiver_username);  -- cerca il codice della conversazione

if(conversazione_codice = -1) then -- conversazione non trovata quindi inesistente, si inserisce una nuova
    -- traccia le conversazioni
    if ((sender_is_ucc = 1) and (receiver_is_ucc = 1)) then
        call BachecaElettronicadb.inserimentoConversazione(conversazione_codice, sender_username, receiver_username, NULL); -- nuova conversazione
        call traccia_Conversazione(conversazione_codice, sender_username, receiver_username, NULL); -- inserisce tutti i vincoli
    elseif ((sender_is_ucc = 1) and (receiver_is_uscc = 1)) then
        call BachecaElettronicadb.inserimentoConversazione(conversazione_codice, sender_username, NULL, receiver_username);
        call traccia_Conversazione(conversazione_codice, sender_username, NULL, receiver_username);
    else -- (sender_is_uscc = 1) and (receiver_is_ucc = 1)
        call BachecaElettronicadb.inserimentoConversazione(conversazione_codice, receiver_username, NULL, sender_username);
        call traccia_Conversazione(conversazione_codice, receiver_username, NULL, sender_username);
    end if;
end if;

/* NOTA
 * Non è presente la condizione (sender_is_uscc = 1) and (receiver_is_uscc = 1))
 * perchè non è possibile che due utenti USCC si scrivano
 */

if (conversazione_codice is not null) then
```

```
                INSERT INTO BachecaElettronicadb.Messaggio (Data, Conversazione_Codice, Testo)  -- conversazione esistente, allora inserisco il messaggio
                VALUES (now(), conversazione_codice, testo);
        end if;

    commit;

END//
DELIMITER ;
-- ----------------------------------------------------------------
```

- Funzione di appoggio per altre stored procedure, utilizzata per controllare se gli utenti (passati nei parametri) abbiano l'autorizzazione ad accedere a quella conversazione oppure utilizzata per cercare una conversazione tra i due utenti (facendo delle combinazioni sugli username)

```
-- Controllo Privacy e Ricerca conversazione ------------------------
DELIMITER //
DROP FUNCTION IF EXISTS BachecaElettronicadb.controllo_conversazione ;
CREATE FUNCTION BachecaElettronicadb.controllo_conversazione (id_conversation_check INT, sender_username VARCHAR(45), receiver_username VARCHAR(45)) RETURNS INT
DETERMINISTIC
BEGIN
    declare conversazione_codice_controllo INT DEFAULT NULL;

    -- Ricerca del codice della conversazione, basando la ricerca sulla combinazione degli username
    if (id_conversation_check is null) then
        SELECT Codice into conversazione_codice_controllo
        FROM BachecaElettronicadb.Conversazione
        WHERE BachecaElettronicadb.Conversazione.UCC_Username_1 = sender_username and BachecaElettronicadb.Conversazione.USCC_Username = receiver_username;

        if(conversazione_codice_controllo is null) then
            SELECT Codice into conversazione_codice_controllo
            FROM BachecaElettronicadb.Conversazione
```

```
                WHERE BachecaElettronicadb.Conversazione.UCC_Username_1 = sender_username and BachecaElettronicadb.Conversazione.UCC_Username_2 = receiver_username;
            end if;
            if(conversazione_codice_controllo is null) then
                SELECT Codice into conversazione_codice_controllo
                FROM BachecaElettronicadb.Conversazione
                WHERE BachecaElettronicadb.Conversazione.UCC_Username_1 = receiver_username and BachecaElettronicadb.Conversazione.UCC_Username_2 = sender_username;
            end if;
            if(conversazione_codice_controllo is null) then
                SELECT Codice into conversazione_codice_controllo
                FROM BachecaElettronicadb.Conversazione
                WHERE BachecaElettronicadb.Conversazione.UCC_Username_1 = receiver_username and BachecaElettronicadb.Conversazione.USCC_Username = sender_username;
            end if;

            if(conversazione_codice_controllo is null) then -- errore, conversazione non trovata quindi inesistente
                SET conversazione_codice_controllo = -1;
            end if;
        end if;

        -- Controlla che gli utenti, con gli username passati, siano autorizzati ad accedere al codice di conversazione passato
        if (id_conversation_check is not null) then
            SELECT Codice into conversazione_codice_controllo
            FROM BachecaElettronicadb.Conversazione
            WHERE BachecaElettronicadb.Conversazione.Codice = id_conversation_check and BachecaElettronicadb.Conversazione.UCC_Username_1 = sender_username and BachecaElettronicadb.Conversazione.USCC_Username = receiver_username;

            if(conversazione_codice_controllo is null) then
                SELECT Codice into conversazione_codice_controllo
                FROM BachecaElettronicadb.Conversazione
```

```
                WHERE BachecaElettronicadb.Conversazione.Codice = id_conversation_check and BachecaElettronicadb.Conversazione.UCC_Username_1 = sender_username and BachecaElettronicadb.Conversazione.UCC_Username_2 = receiver_username;
            end if;
            if(conversazione_codice_controllo is null) then
                SELECT Codice into conversazione_codice_controllo
                FROM BachecaElettronicadb.Conversazione
                WHERE BachecaElettronicadb.Conversazione.Codice = id_conversation_check and BachecaElettronicadb.Conversazione.UCC_Username_1 = receiver_username and BachecaElettronicadb.Conversazione.UCC_Username_2 = sender_username;
            end if;
            if(conversazione_codice_controllo is null) then
                SELECT Codice into conversazione_codice_controllo
                FROM BachecaElettronicadb.Conversazione
                WHERE BachecaElettronicadb.Conversazione.Codice = id_conversation_check and BachecaElettronicadb.Conversazione.UCC_Username_1 = receiver_username and BachecaElettronicadb.Conversazione.USCC_Username = sender_username;
            end if;

            if(conversazione_codice_controllo is null) then -- errore, conversazione non trovata quindi inesistente
                SET conversazione_codice_controllo = -1;
            end if;
        end if;

        RETURN conversazione_codice_controllo;

    END//
    DELIMITER ;
    -- ----------------------------------------------------------------

    -- Visualizza messaggio ----------------------------------------
    DELIMITER //
    DROP PROCEDURE IF EXISTS BachecaElettronicadb.visualizzaMessaggio ;
```

```
CREATE PROCEDURE BachecaElettronicadb.visualizzaMessaggio (IN id_conversation INT, sender_username VARCHAR(45), IN receiver_username VARCHAR(45))
BEGIN
	declare conversazione_codice INT DEFAULT NULL;
	declare exit handler for sqlexception
	begin
		ROLLBACK; -- rollback any changes made in the transaction
		RESIGNAL; -- raise again the sql exceptionto the caller
	end;

	/*NOTA
	 * Ho scelto questo livello per evitare letture sporche
	 */

	SET transaction read only;
	SET transaction isolation level read committed;
	start transaction;

		-- sender_username è l'username dell'utente che sta chiamando la stored procedure
		-- receiver_username è l'username dell'utente con cui sender_username ha avuto una conversazione

		-- l'utente ha passato un codice di conversazione, probabilmente visualizzando lo storico
		if(id_conversation is not null) then
			SET conversazione_codice = controllo_conversazione(id_conversation, sender_username, receiver_username);
			IF (conversazione_codice = -1) then
				signal sqlstate '45012' set message_text = 'You are not authorized to access this conversation';
			end IF;
		end if;

		if(id_conversation is null) then
```

```
                SET conversazione_codice = controllo_conversazione(NULL, sender_username, receiver_username);
            end if;

5           if(conversazione_codice = -1) then -- errore, conversazione non trovata quindi inesistente
                signal sqlstate '45011' set message_text = 'Conversation not found';
            else
                SELECT Testo, Data
10              FROM BachecaElettronicadb.Messaggio
                WHERE BachecaElettronicadb.Messaggio.Conversazione_Codice = conversazione_codice
                ORDER BY Data ASC;
            end if;
15
        commit;

    END//
    DELIMITER ;
20  -- ----------------------------------------------------------------

    -- Visualizzazione Storico UCC ----------------------------------------
    DELIMITER //
    DROP PROCEDURE IF EXISTS BachecaElettronicadb.visualizzaStorico_UCC ;
25  CREATE PROCEDURE BachecaElettronicadb.visualizzaStorico_UCC (IN ucc_username VARCHAR(45))
    BEGIN
        declare idStorico INT;
        declare exit handler for sqlexception
30      begin
            ROLLBACK; -- rollback any changes made in the transaction
            RESIGNAL; -- raise again the sql exceptionto the caller
        end;
```

```
        /*NOTA
         * Ho scelto questo livello per evitare letture sporche
         */

5       SET transaction read only;
        SET transaction isolation level read committed;
        start transaction;

                -- Trova il codice dello storico creato nel momento della registrazione
10              SELECT StoricoConversazione_ID INTO idStorico FROM
BachecaElettronicadb.UCC WHERE BachecaElettronicadb.UCC.Username = ucc_username;

                -- Stampa tutte le conversazioni dell'utente con il relativo partecipante
                SELECT Codice, UCC_Username_2 AS 'Username utente UCC', USCC_Username
15  AS 'Username utente USCC'
                FROM BachecaElettronicadb.Conversazione
                WHERE BachecaElettronicadb.Conversazione.Codice in (SELECT CodiceConv

                FROM BachecaElettronicadb.ConversazioneCodice

20              WHERE BachecaElettronicadb.ConversazioneCodice.StoricoConversazione_ID =
idStorico);
        commit;
END//
25  DELIMITER ;
-- ----------------------------------------------------------------

-- Visualizzazione Storico USCC ----------------------------------
DELIMITER //
30  DROP PROCEDURE IF EXISTS BachecaElettronicadb.visualizzaStorico_USCC ;
CREATE PROCEDURE BachecaElettronicadb.visualizzaStorico_USCC (IN uscc_username
VARCHAR(45))
BEGIN
        declare idStorico INT;
```

```
declare exit handler for sqlexception
begin
        ROLLBACK; -- rollback any changes made in the transaction
        RESIGNAL; -- raise again the sql exceptionto the caller
end;

/*NOTA
 * Ho scelto questo livello per evitare letture sporche
 */

SET transaction read only;
SET transaction isolation level read committed;
start transaction;

        -- Trova il codice dello storico creato nel momento della registrazione
        SELECT StoricoConversazione_ID INTO idStorico FROM BachecaElettronicadb.USCC WHERE BachecaElettronicadb.USCC.Username = uscc_username;

        -- Stampa tutte le conversazioni dell'utente con il relativo partecipante
        SELECT Codice, UCC_Username_1 AS 'Username utente UCC', UCC_Username_2 AS 'Username utente UCC'
        FROM BachecaElettronicadb.Conversazione
        WHERE BachecaElettronicadb.Conversazione.Codice in (SELECT CodiceConv

        FROM BachecaElettronicadb.ConversazioneCodice

        WHERE BachecaElettronicadb.ConversazioneCodice.StoricoConversazione_ID = idStorico);
commit;


END//
DELIMITER ;
-- ----------------------------------------------------------------
```

```
-- Seguire un annuncio ----------------------------------------------
DELIMITER //
DROP PROCEDURE IF EXISTS BachecaElettronicadb.segui_Annuncio ;
CREATE PROCEDURE BachecaElettronicadb.segui_Annuncio (IN codice_annuncio INT, IN username VARCHAR(45))
BEGIN
	declare exit handler for sqlexception
	begin
		ROLLBACK; -- rollback any changes made in the transaction
		RESIGNAL; -- raise again the sql exceptionto the caller
	end;

	/*NOTA
	 * Ho scelto questo livello per evitare letture sporche
	 */

	SET transaction isolation level read committed;
	start transaction;

		/* NOTA
		 * L'idea è: Verifica che l'utente sia un UCC, altrimenti controlla se è un USCC.
		 * Se non fosse nè UCC e nè USCC allora l'utente non esiste
		 */

		if (not exists (SELECT Username FROM BachecaElettronicadb.UCC WHERE BachecaElettronicadb.UCC.Username = username)) then
			IF (not exists (SELECT Username FROM BachecaElettronicadb.USCC WHERE BachecaElettronicadb.USCC.Username = username)) THEN
				signal sqlstate '45000' set message_text = 'User not found';
			ELSE
				INSERT INTO BachecaElettronicadb.`Seguito-USCC`(USCC_Username, Annuncio_Codice) VALUES(username, codice_annuncio);
			end IF;
```

```
                else
                        INSERT INTO BachecaElettronicadb.`Seguito-UCC`(UCC_Username, Annuncio_Codice) VALUES(username, codice_annuncio);
                end if;

        commit;

END//
DELIMITER ;
-- ----------------------------------------------------------------

-- Visualizzazione degli Annunci seguiti ----------------------------
DELIMITER //
DROP PROCEDURE IF EXISTS BachecaElettronicadb.visualizza_Annunci_Seguiti ;
CREATE PROCEDURE BachecaElettronicadb.visualizza_Annunci_Seguiti (IN username VARCHAR(45))
BEGIN
        declare exit handler for sqlexception
        begin
                ROLLBACK; -- rollback any changes made in the transaction
                RESIGNAL; -- raise again the sql exceptionto the caller
        end;

        /*NOTA
         * Ho scelto questo livello per evitare letture sporche
         */

        SET transaction read only;
        SET transaction isolation level read committed;
        start transaction;

                /* NOTA
                 * L'idea è: Verifica che l'utente sia un UCC, altrimenti controlla se è un USCC.
                 * Se non fosse nè UCC e nè USCC allora l'utente non esiste
```

```
*/

        if (not exists (SELECT Username FROM BachecaElettronicadb.UCC WHERE BachecaElettronicadb.UCC.Username = username)) then
            IF (not exists (SELECT Username FROM BachecaElettronicadb.USCC WHERE BachecaElettronicadb.USCC.Username = username)) THEN
                signal sqlstate '45000' set message_text = 'User not found';
            ELSE
                SELECT annuncio.Codice, annuncio.Stato, annuncio.UCC_Username AS 'Proprietario'
                FROM BachecaElettronicadb.`Seguito-USCC` AS seguiti JOIN BachecaElettronicadb.Annuncio AS annuncio ON seguiti.Annuncio_Codice = annuncio.Codice
                WHERE seguiti.USCC_Username = username;
            end IF;
        else
            SELECT annuncio.Codice, annuncio.Stato, annuncio.UCC_Username AS 'Proprietario'
            FROM BachecaElettronicadb.`Seguito-UCC` AS seguiti JOIN BachecaElettronicadb.Annuncio AS annuncio ON seguiti.Annuncio_Codice = annuncio.Codice
            WHERE seguiti.UCC_Username = username;
        end if;

    commit;

END//
DELIMITER ;
-- -----------------------------------------------------------------

-- Inserimento o Rimozione nota in un Annuncio ----------------------
DELIMITER //
DROP PROCEDURE IF EXISTS BachecaElettronicadb.modificaNota ;
CREATE PROCEDURE BachecaElettronicadb.modificaNota (IN testo VARCHAR(45), IN annuncio_codice INT, IN ID_rimozione_nota INT, IN username VARCHAR(45))
BEGIN
```

```
declare exit handler for sqlexception
begin
        ROLLBACK; -- rollback any changes made in the transaction
        RESIGNAL; -- raise again the sql exceptionto the caller
end;

/*NOTA
 * Ho scelto questo livello per evitare letture sporche
 */

SET transaction isolation level read committed;
start transaction;

-- possibile implementazione trigger che controlla se una nuova nota inserita sia riferita ad un annuncio attivo

        if (not exists (SELECT Codice FROM BachecaElettronicadb.Annuncio WHERE BachecaElettronicadb.Annuncio.Codice = annuncio_codice and BachecaElettronicadb.Annuncio.UCC_Username = username)) then
                signal sqlstate '45008' set message_text = 'You are not the owner of the ad';
        end if;

        if (not exists (SELECT Codice FROM BachecaElettronicadb.Annuncio WHERE BachecaElettronicadb.Annuncio.Codice = annuncio_codice and BachecaElettronicadb.Annuncio.Stato = 'Attivo')) then
                signal sqlstate '45007' set message_text = 'Ad not active now';
        end if;

        if (ID_rimozione_nota is null) then
                INSERT INTO BachecaElettronicadb.Nota (ID, Testo, Annuncio_Codice) VALUES(NULL, testo, annuncio_codice);
        else
                DELETE FROM BachecaElettronicadb.Nota
                WHERE BachecaElettronicadb.Nota.ID = ID_rimozione_nota;
```

```
        end if;

    commit;

END//
DELIMITER ;
-- ----------------------------------------------------------------

-- Visualizza nota ------------------------------------------------
DELIMITER //
DROP PROCEDURE IF EXISTS BachecaElettronicadb.visualizzaNota ;
CREATE PROCEDURE BachecaElettronicadb.visualizzaNota (IN annuncio_Codice INT)
BEGIN
    declare exit handler for sqlexception
    begin
        ROLLBACK; -- rollback any changes made in the transaction
        RESIGNAL; -- raise again the sql exceptionto the caller
    end;

    /*NOTA
     * Ho scelto questo livello per evitare letture sporche
     */

    SET transaction read only;
    SET transaction isolation level read committed;
    start transaction;

        if (not exists (SELECT Codice FROM BachecaElettronicadb.Annuncio WHERE
BachecaElettronicadb.Annuncio.Codice = annuncio_Codice and
BachecaElettronicadb.Annuncio.Stato = 'Attivo')) then
            signal sqlstate '45007' set message_text = 'Ad not active now';
        end if;

        SELECT ID, Testo, Annuncio_Codice AS "Codice dell'annuncio"
```

```
            FROM BachecaElettronicadb.Nota
            WHERE BachecaElettronicadb.Nota.Annuncio_Codice = annuncio_Codice;

        commit;

END//
DELIMITER ;
-- -----------------------------------------------------------------

-- Rimozione di un Annuncio ----------------------------------------
DELIMITER //
DROP PROCEDURE IF EXISTS BachecaElettronicadb.rimuoviAnnuncio ;
CREATE PROCEDURE BachecaElettronicadb.rimuoviAnnuncio (IN codice_annuncio INT, IN
ucc_username VARCHAR(45))
BEGIN

        /*NOTA
         * Non ho impostato nessun livello di isolamento perchè
         * non c'è concorrenza sulla singolo annuncio, ovvero
         * solo un solo utente può modificare il suo annuncio.
         */

        -- Controlla che il paramentro sia corretto con la regola aziendale impostata
        if (not exists (SELECT Codice FROM BachecaElettronicadb.Annuncio WHERE
BachecaElettronicadb.Annuncio.Codice=codice_annuncio and BachecaElettronicadb.Annuncio.Stato
= 'Attivo' and BachecaElettronicadb.Annuncio.UCC_Username = ucc_username)) then
            signal sqlstate '45006' set message_text = 'Ad not found';
        end if;

        UPDATE BachecaElettronicadb.Annuncio
        SET BachecaElettronicadb.Annuncio.Stato = 'Rimosso'
        WHERE BachecaElettronicadb.Annuncio.Codice = codice_annuncio AND
BachecaElettronicadb.Annuncio.Stato = 'Attivo' and
BachecaElettronicadb.Annuncio.UCC_Username = ucc_username;
```

```
END//
DELIMITER ;
-- -----------------------------------------------------------------
5

-- Annuncio Venduto ------------------------------------------------
DELIMITER //
DROP PROCEDURE IF EXISTS BachecaElettronicadb.vendutoAnnuncio ;
10 CREATE PROCEDURE BachecaElettronicadb.vendutoAnnuncio (IN codice_annuncio INT, IN
ucc_username VARCHAR(45))
BEGIN

        /*NOTA
15       * Non ho impostato nessun livello di isolamento perchè
         * non c'è concorrenza sulla singolo annuncio, ovvero
         * solo un solo utente può modificare il suo annuncio.
         */

20      -- Controlla che il paramentro sia corretto con la regola aziendale impostata
        if (not exists (SELECT Codice FROM BachecaElettronicadb.Annuncio WHERE
BachecaElettronicadb.Annuncio.Codice=codice_annuncio and BachecaElettronicadb.Annuncio.Stato
= 'Attivo' and BachecaElettronicadb.Annuncio.UCC_Username = ucc_username)) then
                signal sqlstate '45006' set message_text = 'Ad not found';
25      end if;

        UPDATE BachecaElettronicadb.Annuncio
        SET BachecaElettronicadb.Annuncio.Stato = 'Venduto'
        WHERE BachecaElettronicadb.Annuncio.Codice = codice_annuncio AND
30 BachecaElettronicadb.Annuncio.Stato = 'Attivo' and
BachecaElettronicadb.Annuncio.UCC_Username = ucc_username;

END//
DELIMITER ;
```

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

- Questa è un'altra stored procedure una delle più "complicate". Ha il compito di generare un nuovo report e i privilegi sono solamente dell'amministratore. Crea un cursore sugli annunci venduti e ancora non calcolati dell'utente, che con un loop calcola la somma totale degli annunci, numero degli annunci ed altre informazioni da mandare in persistenza. Inoltre durante la progettazione si è deciso di lasciare la possibilità di riscuotere l'annuncio solamente all'amministratore che si è occupato di generare il report. La generazione di un nuovo report poteva essere implementato anche attraverso un trigger che esegue un AFTER UPDATE ON Annunci, dove lo stato era in "venduto" ma si è scelto di implementare una stored procedure, rispetto ad un trigger, in quanto essi potevano garantire un uso delle risorse migliore, perché il calcolo viene effettuato a discrezione dell'amministratore. Ovvero un trigger si sarebbe attivato ogni qualvolta che un annuncio venisse indicato come venduto, quindi per un elevato traffico nel sistema, questo portava una degradazione delle prestazioni e anche un inutile calcolo da fare immediatamente dal punto di vista delle specifiche.

  NOTA: La riscossione del report è solamente per scopi gestionali degli amministratori, in modo da tracciare i report da loro riscossi.

-- **Generazione di un nuovo report** ----------

```
DELIMITER //
DROP PROCEDURE IF EXISTS BachecaElettronicadb.genera_report ;
CREATE PROCEDURE BachecaElettronicadb.genera_report (IN ucc_username VARCHAR(45), IN amministratore_username VARCHAR(45))
BEGIN

    declare numeroCarta VARCHAR(16);
    declare codice_annuncio INT;
    declare numero_annunci INT DEFAULT 0;
    declare importo_totale INT DEFAULT 0;
    declare loop_amount INT DEFAULT 0;
    declare cur_id_annuncio CURSOR FOR SELECT Codice FROM BachecaElettronicadb.Annuncio WHERE BachecaElettronicadb.Annuncio.UCC_Username = ucc_username and BachecaElettronicadb.Annuncio.Stato = 'Venduto';
    declare CONTINUE HANDLER FOR NOT FOUND SET loop_amount = 1;
```

```
declare exit handler for sqlexception
begin
        ROLLBACK; -- rollback any changes made in the transaction
        RESIGNAL; -- raise again the sql exceptionto the caller
end;

/*NOTA
 * Ho scelto questo livello per evitare letture inconsistenti
 * leggendo l'utente più di una volta
 */

SET transaction isolation level repeatable read;
start transaction;

        /*NOTA
         * Ho scelto di implementare una stored procedure, rispetto ad un trigger, in quanto essi potevano garantire un uso delle risorse migliore, perché
         * il calcolo viene effettuato a discrezione dell'amministratore.
         * Ovvero un trigger si sarebbe attivato ogni qualvolta che un annuncio venisse indicato come venduto, quindi per un elevato traffico nel
         * sistema, questo portava una degradazione delle prestazioni e anche un inutile calcolo da fare immediatamente dal punto di vista delle specifiche.
         */

        /*NOTA
         * L'idea della stored procedure è:
         * Calcolare la somma degli importi di tutti gli annunci venduti non calcolati e la percentuale per l'amministratore che ha chiamanto la stored procedure
         */

        if (not exists (SELECT Username FROM BachecaElettronicadb.UCC WHERE BachecaElettronicadb.UCC.Username = ucc_username)) then
                signal sqlstate '45000' set message_text = 'User not found';
        end if;
```

```
if (not exists (SELECT Codice FROM BachecaElettronicadb.Annuncio WHERE BachecaElettronicadb.Annuncio.UCC_Username = ucc_username and BachecaElettronicadb.Annuncio.Stato = 'Venduto')) then
    signal sqlstate '45014' set message_text = 'This user has not sold any ads';
end if;

-- Cerca il numero di carta dell'utente
SELECT BachecaElettronicadb.UCC.NumeroCarta INTO numeroCarta
FROM BachecaElettronicadb.UCC
WHERE BachecaElettronicadb.UCC.Username = ucc_username;

-- Calcola l'importo totale degli annunci venduti ancora non riscossi
OPEN cur_id_annuncio;
calculate_amount: LOOP
                    FETCH cur_id_annuncio INTO codice_annuncio;

                    IF loop_amount = 1 THEN
                        LEAVE calculate_amount;
                    end IF;

                     -- calcola gli importi dei singoli annunci ancora non calcolati
                    SET numero_annunci = numero_annunci + 1;
                     SET importo_totale = importo_totale + (SELECT Importo
                    FROM BachecaElettronicadb.Annuncio
                     WHERE BachecaElettronicadb.Annuncio.Codice = codice_annuncio AND BachecaElettronicadb.Annuncio.Report_calcolato = false);

                    -- Imposta a true l'attributo in modo tale da non essere ricalcolato
```

```
                                    UPDATE BachecaElettronicadb.Annuncio
                                    SET BachecaElettronicadb.Annuncio.Report_calcolato
= true
                                        WHERE BachecaElettronicadb.Annuncio.Codice =
5   codice_annuncio;

            end LOOP calculate_amount;
            CLOSE cur_id_annuncio;

10          INSERT INTO BachecaElettronicadb.Report (Codice, UCC_Username,
ImportoTotale, NumeroAnnunci, NumeroCarta, Data, Amministratore_Username,
Riscosso_Amministratore, Importo_Amministratore, Importo_UCC)
            VALUES(NULL, ucc_username, importo_totale, numero_annunci, numeroCarta,
now(), amministratore_username, false, importo_totale*0.3, importo_totale*0.7);
15
        commit;
    END//
    DELIMITER ;
    -- ----------------------------------------------------------------
20
    -- Riscossione di un report Amministratore ----------------------------
    DELIMITER //
    DROP PROCEDURE IF EXISTS BachecaElettronicadb.riscossione_report ;
    CREATE PROCEDURE BachecaElettronicadb.riscossione_report (IN codice_report INT, IN
25  ucc_username VARCHAR(45), IN amministratore_username VARCHAR(45))
    BEGIN
        declare importo INT;
        declare exit handler for sqlexception
        begin
30          ROLLBACK; -- rollback any changes made in the transaction
            RESIGNAL; -- raise again the sql exceptionto the caller
        end;

        /*NOTA
```

```
        * Ho scelto questo livello per evitare letture inconsistenti con il report
        */

        SET transaction isolation level repeatable read;
        start transaction;

            if (not exists (SELECT Username FROM BachecaElettronicadb.UCC WHERE BachecaElettronicadb.UCC.Username = ucc_username)) then
                signal sqlstate '45000' set message_text = 'User not found';
            end if;

            if (not exists (SELECT Codice FROM BachecaElettronicadb.Report WHERE BachecaElettronicadb.Report.Codice = codice_report and BachecaElettronicadb.Report.UCC_Username = ucc_username)) then
                signal sqlstate '45015' set message_text = 'Report not found';
            end if;

            if (exists (SELECT Codice FROM BachecaElettronicadb.Report WHERE BachecaElettronicadb.Report.Codice = codice_report and BachecaElettronicadb.Report.Riscosso_Amministratore = true)) then
                signal sqlstate '45016' set message_text = 'Report already collected';
            end if;

            -- Solo l'amministratore che ha generato il report può riscuotere
            if (not exists (SELECT Codice FROM BachecaElettronicadb.Report WHERE BachecaElettronicadb.Report.Codice = codice_report and BachecaElettronicadb.Report.Amministratore_Username = amministratore_username)) then
                signal sqlstate '45017' set message_text = 'You are not the owner of the report';
            end if;

            SELECT ImportoTotale INTO importo
            FROM BachecaElettronicadb.Report
            WHERE BachecaElettronicadb.Report.Codice = codice_report;
```

```
                UPDATE BachecaElettronicadb.Report
                SET BachecaElettronicadb.Report.Riscosso_Amministratore = true
                WHERE BachecaElettronicadb.Report.Codice = codice_report;

        commit;

END//
DELIMITER ;
-- -----------------------------------------------------------------

-- Visualizzazione report ------------------------------------------
DELIMITER //
DROP PROCEDURE IF EXISTS BachecaElettronicadb.visualizza_report ;
CREATE PROCEDURE BachecaElettronicadb.visualizza_report (IN codice_report INT, IN
ucc_username VARCHAR(45))
BEGIN
        declare exit handler for sqlexception
        begin
                ROLLBACK; -- rollback any changes made in the transaction
                RESIGNAL; -- raise again the sql exceptionto the caller
        end;

        /*NOTA
         * Ho scelto questo livello per evitare letture sporche
         */

        SET transaction read only;
        SET transaction isolation level read committed;
        start transaction;

                if (not exists (SELECT Username FROM BachecaElettronicadb.UCC WHERE
BachecaElettronicadb.UCC.Username = ucc_username)) then
                        signal sqlstate '45000' set message_text = 'User not found';
                end if;
```

```
        if (codice_report is null) then
            SELECT Codice, UCC_Username, ImportoTotale, NumeroAnnunci, NumeroCarta, Importo_UCC, Amministratore_Username, Riscosso_Amministratore, Importo_Amministratore
            FROM BachecaElettronicadb.Report
            WHERE BachecaElettronicadb.Report.UCC_Username = ucc_username;
        else
            SELECT Codice, UCC_Username, ImportoTotale, NumeroAnnunci, NumeroCarta, Importo_UCC, Amministratore_Username, Riscosso_Amministratore, Importo_Amministratore
            FROM BachecaElettronicadb.Report
            WHERE BachecaElettronicadb.Report.Codice = codice_report and BachecaElettronicadb.Report.UCC_Username = ucc_username;
        end if;

    commit;

END//
DELIMITER ;
-- -----------------------------------------------------------------
```

- Visualizza i dati di base dell'utente. È stata risparmiata una stored procedure grazie alla doppia condizione

```
-- Visualizzazione informazioni utente ------------------------------
DELIMITER //
DROP PROCEDURE IF EXISTS BachecaElettronicadb.visualizza_Info_Utente ;
CREATE PROCEDURE BachecaElettronicadb.visualizza_Info_Utente (IN username VARCHAR(45))
BEGIN
    declare exit handler for sqlexception
    begin
        ROLLBACK; -- rollback any changes made in the transaction
        RESIGNAL; -- raise again the sql exceptionto the caller
    end;
```

```
/*NOTA
 * Ho scelto questo livello per evitare letture sporche
 */

SET transaction read only;
SET transaction isolation level read committed;
start transaction;

        /* Nota
         * L'idea è: Verifica che l'utente sia un UCC, altrimenti controlla se è un USCC.
         * Se non sarà nè UCC e nè USCC allora l'utente non esiste.
         */

        if (not exists (SELECT Username FROM BachecaElettronicadb.UCC WHERE BachecaElettronicadb.UCC.Username=username)) then
                IF (not exists (SELECT Username FROM BachecaElettronicadb.USCC WHERE BachecaElettronicadb.USCC.Username=username)) THEN
                        signal sqlstate '45000' set message_text = 'User not found';
                ELSE
                        SELECT Username, Password, CF_Anagrafico AS 'Codice Fiscale'
                        FROM BachecaElettronicadb.USCC
                        WHERE BachecaElettronicadb.USCC.Username = username;
                end IF;
        else
                SELECT Username, Password, NumeroCarta AS 'Numero Carta', DataScadenza AS 'Scadenza', CVC, CF_Anagrafico AS 'Codice Fiscale'
                FROM BachecaElettronicadb.UCC
                WHERE BachecaElettronicadb.UCC.Username = username;
        end if;

commit;

END//
```

```
DELIMITER ;


-- Login ------------------------------------------------------------
DELIMITER //
DROP PROCEDURE IF EXISTS BachecaElettronicadb.login ;
CREATE PROCEDURE BachecaElettronicadb.login (IN username VARCHAR(45), IN password VARCHAR(45), OUT role INT)
BEGIN

	SET role = -1;

	if exists (SELECT Username FROM BachecaElettronicadb.Amministratore WHERE BachecaElettronicadb.Amministratore.Username = username and BachecaElettronicadb.Amministratore.Password = md5(password)) then
		SET role = 1;
	end if;

	if exists (SELECT Username FROM BachecaElettronicadb.UCC WHERE BachecaElettronicadb.UCC.Username = username and BachecaElettronicadb.UCC.Password = md5(password)) and role = -1 then
		SET role = 2;
	end if;

	if exists (SELECT Username FROM BachecaElettronicadb.USCC WHERE BachecaElettronicadb.USCC.Username = username and BachecaElettronicadb.USCC.Password = md5(password)) and role = -1 then
		SET role = 3;
	end if;

	if (role = -1) then
		SET role = 4;
	end if;
```

```
END//
DELIMITER ;
-- ----------------------------------------------------------------
```

Appendice: Implementazione

23.Codice SQL per instanziare il database

-- MySQL Workbench Forward Engineering

```
SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0;
SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0;
SET @OLD_SQL_MODE=@@SQL_MODE, SQL_MODE='ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZERO_DATE,ERROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION';

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

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

-- -----------------------------------------------------
-- Table `BachecaElettronicadb`.`Amministratore`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `BachecaElettronicadb`.`Amministratore` ;

CREATE TABLE IF NOT EXISTS `BachecaElettronicadb`.`Amministratore` (
  `Username` VARCHAR(45) NOT NULL,
  `Password` VARCHAR(45) NOT NULL,
```

```
  PRIMARY KEY (`Username`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;

CREATE UNIQUE INDEX `Username_UNIQUE` ON `BachecaElettronicadb`.`Amministratore` (`Username` ASC) VISIBLE;


-- -----------------------------------------------------
-- Table `BachecaElettronicadb`.`Categoria`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `BachecaElettronicadb`.`Categoria` ;

CREATE TABLE IF NOT EXISTS `BachecaElettronicadb`.`Categoria` (
  `Nome` VARCHAR(20) NOT NULL,
  PRIMARY KEY (`Nome`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;

CREATE UNIQUE INDEX `Nome_UNIQUE` ON `BachecaElettronicadb`.`Categoria` (`Nome` ASC) VISIBLE;


-- -----------------------------------------------------
-- Table `BachecaElettronicadb`.`InformazioneAnagrafica`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `BachecaElettronicadb`.`InformazioneAnagrafica` ;

CREATE TABLE IF NOT EXISTS `BachecaElettronicadb`.`InformazioneAnagrafica` (
  `CF` VARCHAR(16) NOT NULL,
  `Cognome` VARCHAR(20) NOT NULL,
  `Nome` VARCHAR(20) NOT NULL,
```

```
  `IndirizzoDiResidenza` VARCHAR(20) NOT NULL,
  `CAP` INT NOT NULL,
  `IndirizzoDiFatturazione` VARCHAR(20) NULL DEFAULT NULL,
  `TipoRecapitoPreferito` ENUM('email', 'cellulare', 'social', 'sms') NOT NULL,
  `RecapitoPreferito` VARCHAR(40) NOT NULL,
  PRIMARY KEY (`CF`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;

CREATE UNIQUE INDEX `CF_UNIQUE` ON `BachecaElettronicadb`.`InformazioneAnagrafica`
(`CF` ASC) VISIBLE;


-- -----------------------------------------------------
-- Table `BachecaElettronicadb`.`StoricoConversazione`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `BachecaElettronicadb`.`StoricoConversazione` ;

CREATE TABLE IF NOT EXISTS `BachecaElettronicadb`.`StoricoConversazione` (
  `ID` INT NOT NULL AUTO_INCREMENT,
  PRIMARY KEY (`ID`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;

CREATE UNIQUE INDEX `ID_UNIQUE` ON `BachecaElettronicadb`.`StoricoConversazione`
(`ID` ASC) VISIBLE;


-- -----------------------------------------------------
-- Table `BachecaElettronicadb`.`UCC`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `BachecaElettronicadb`.`UCC` ;
```

```
CREATE TABLE IF NOT EXISTS `BachecaElettronicadb`.`UCC` (
  `Username` VARCHAR(45) NOT NULL,
  `Password` VARCHAR(45) NOT NULL,
  `NumeroCarta` VARCHAR(16) NOT NULL,
  `DataScadenza` DATE NOT NULL,
  `CVC` INT NOT NULL,
  `StoricoConversazione_ID` INT NOT NULL,
  `CF_Anagrafico` VARCHAR(16) NOT NULL,
  PRIMARY KEY (`Username`),
  CONSTRAINT `fk_UCC_InformazioneAnagrafica1`
    FOREIGN KEY (`CF_Anagrafico`)
    REFERENCES `BachecaElettronicadb`.`InformazioneAnagrafica` (`CF`),
  CONSTRAINT `fk_UCC_StoricoConversazione1`
    FOREIGN KEY (`StoricoConversazione_ID`)
    REFERENCES `BachecaElettronicadb`.`StoricoConversazione` (`ID`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;

CREATE UNIQUE INDEX `Username_UNIQUE` ON `BachecaElettronicadb`.`UCC` (`Username`
ASC) VISIBLE;

CREATE INDEX `fk_UCC_StoricoConversazione1_idx` ON `BachecaElettronicadb`.`UCC`
(`StoricoConversazione_ID` ASC) VISIBLE;

CREATE INDEX `fk_UCC_InformazioneAnagrafica1_idx` ON `BachecaElettronicadb`.`UCC`
(`CF_Anagrafico` ASC) VISIBLE;


-- -----------------------------------------------------
-- Table `BachecaElettronicadb`.`Annuncio`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `BachecaElettronicadb`.`Annuncio` ;
```

```
CREATE TABLE IF NOT EXISTS `BachecaElettronicadb`.`Annuncio` (
  `Codice` INT NOT NULL AUTO_INCREMENT,
  `Stato` ENUM('Attivo', 'Venduto', 'Rimosso') NOT NULL,
  `Descrizione` VARCHAR(100) NOT NULL,
  `Importo` INT NOT NULL,
  `Foto` VARCHAR(9) NULL DEFAULT NULL,
  `Report_calcolato` TINYINT NOT NULL DEFAULT '0',
  `UCC_Username` VARCHAR(45) NOT NULL,
  `Categoria_Nome` VARCHAR(20) NOT NULL,
  PRIMARY KEY (`Codice`),
  CONSTRAINT `fk_Annuncio_Categoria1`
    FOREIGN KEY (`Categoria_Nome`)
    REFERENCES `BachecaElettronicadb`.`Categoria` (`Nome`),
  CONSTRAINT `fk_Annuncio_UCC`
    FOREIGN KEY (`UCC_Username`)
    REFERENCES `BachecaElettronicadb`.`UCC` (`Username`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;

CREATE UNIQUE INDEX `Codice_UNIQUE` ON `BachecaElettronicadb`.`Annuncio` (`Codice` ASC) VISIBLE;

CREATE INDEX `fk_Annuncio_UCC_idx` ON `BachecaElettronicadb`.`Annuncio` (`UCC_Username` ASC) VISIBLE;

CREATE INDEX `fk_Annuncio_Categoria1_idx` ON `BachecaElettronicadb`.`Annuncio` (`Categoria_Nome` ASC) VISIBLE;


-- -----------------------------------------------------
-- Table `BachecaElettronicadb`.`Commento`
-- -----------------------------------------------------
```

```
DROP TABLE IF EXISTS `BachecaElettronicadb`.`Commento` ;

CREATE TABLE IF NOT EXISTS `BachecaElettronicadb`.`Commento` (
  `ID` INT NOT NULL AUTO_INCREMENT,
  `Testo` VARCHAR(45) NOT NULL,
  `Annuncio_Codice` INT NOT NULL,
  PRIMARY KEY (`ID`, `Annuncio_Codice`),
  CONSTRAINT `fk_Commento_Annuncio1`
    FOREIGN KEY (`Annuncio_Codice`)
    REFERENCES `BachecaElettronicadb`.`Annuncio` (`Codice`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;

CREATE UNIQUE INDEX `Annuncio_Codice_UNIQUE` ON `BachecaElettronicadb`.`Commento`
(`Annuncio_Codice` ASC) VISIBLE;

CREATE INDEX `fk_Commento_Annuncio1_idx` ON `BachecaElettronicadb`.`Commento`
(`Annuncio_Codice` ASC) VISIBLE;


-- -----------------------------------------------------
-- Table `BachecaElettronicadb`.`USCC`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `BachecaElettronicadb`.`USCC` ;

CREATE TABLE IF NOT EXISTS `BachecaElettronicadb`.`USCC` (
  `Username` VARCHAR(45) NOT NULL,
  `Password` VARCHAR(45) NOT NULL,
  `StoricoConversazione_ID` INT NOT NULL,
  `CF_Anagrafico` VARCHAR(16) NOT NULL,
  PRIMARY KEY (`Username`),
  CONSTRAINT `fk_USCC_InformazioneAnagrafica1`
    FOREIGN KEY (`CF_Anagrafico`)
```

```
  REFERENCES `BachecaElettronicadb`.`InformazioneAnagrafica` (`CF`),
 CONSTRAINT `fk_USCC_StoricoConversazione1`
  FOREIGN KEY (`StoricoConversazione_ID`)
  REFERENCES `BachecaElettronicadb`.`StoricoConversazione` (`ID`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;

CREATE UNIQUE INDEX `Username_UNIQUE` ON `BachecaElettronicadb`.`USCC`
(`Username` ASC) VISIBLE;

CREATE INDEX `fk_USCC_StoricoConversazione1_idx` ON `BachecaElettronicadb`.`USCC`
(`StoricoConversazione_ID` ASC) VISIBLE;

CREATE INDEX `fk_USCC_InformazioneAnagrafica1_idx` ON `BachecaElettronicadb`.`USCC`
(`CF_Anagrafico` ASC) VISIBLE;


-- -----------------------------------------------------
-- Table `BachecaElettronicadb`.`Conversazione`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `BachecaElettronicadb`.`Conversazione` ;

CREATE TABLE IF NOT EXISTS `BachecaElettronicadb`.`Conversazione` (
 `Codice` INT NOT NULL AUTO_INCREMENT,
 `UCC_Username_1` VARCHAR(45) NOT NULL,
 `UCC_Username_2` VARCHAR(45) NULL DEFAULT NULL,
 `USCC_Username` VARCHAR(45) NULL DEFAULT NULL,
 PRIMARY KEY (`Codice`),
 CONSTRAINT `fk_Conversazione_UCC1`
  FOREIGN KEY (`UCC_Username_1`)
  REFERENCES `BachecaElettronicadb`.`UCC` (`Username`),
 CONSTRAINT `fk_Conversazione_UCC2`
  FOREIGN KEY (`UCC_Username_2`)
```

```
    REFERENCES `BachecaElettronicadb`.`UCC` (`Username`),
  CONSTRAINT `fk_Conversazione_USCC1`
    FOREIGN KEY (`USCC_Username`)
    REFERENCES `BachecaElettronicadb`.`USCC` (`Username`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;

CREATE UNIQUE INDEX `Codice_UNIQUE` ON `BachecaElettronicadb`.`Conversazione`
(`Codice` ASC) VISIBLE;

CREATE INDEX `fk_Conversazione_UCC1_idx` ON `BachecaElettronicadb`.`Conversazione`
(`UCC_Username_1` ASC) VISIBLE;

CREATE INDEX `fk_Conversazione_USCC1_idx` ON `BachecaElettronicadb`.`Conversazione`
(`USCC_Username` ASC) VISIBLE;

CREATE INDEX `fk_Conversazione_UCC2_idx` ON `BachecaElettronicadb`.`Conversazione`
(`UCC_Username_2` ASC) VISIBLE;


-- -----------------------------------------------------
-- Table `BachecaElettronicadb`.`ConversazioneCodice`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `BachecaElettronicadb`.`ConversazioneCodice` ;

CREATE TABLE IF NOT EXISTS `BachecaElettronicadb`.`ConversazioneCodice` (
  `CodiceConv` INT NOT NULL,
  `StoricoConversazione_ID` INT NOT NULL,
  PRIMARY KEY (`StoricoConversazione_ID`, `CodiceConv`),
  CONSTRAINT `fk_ConversazioneCodice_StoricoConversazione1`
    FOREIGN KEY (`StoricoConversazione_ID`)
    REFERENCES `BachecaElettronicadb`.`StoricoConversazione` (`ID`))
ENGINE = InnoDB
```

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

CREATE INDEX `fk_ConversazioneCodice_StoricoConversazione1_idx` ON
`BachecaElettronicadb`.`ConversazioneCodice` (`StoricoConversazione_ID` ASC) VISIBLE;


-- -----------------------------------------------------
-- Table `BachecaElettronicadb`.`Messaggio`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `BachecaElettronicadb`.`Messaggio` ;

CREATE TABLE IF NOT EXISTS `BachecaElettronicadb`.`Messaggio` (
  `ID` INT NOT NULL AUTO_INCREMENT,
  `Data` DATETIME NOT NULL,
  `Conversazione_Codice` INT NOT NULL,
  `Testo` VARCHAR(100) NOT NULL,
  PRIMARY KEY (`ID`, `Conversazione_Codice`),
  CONSTRAINT `fk_Messaggio_Conversazione1`
    FOREIGN KEY (`Conversazione_Codice`)
    REFERENCES `BachecaElettronicadb`.`Conversazione` (`Codice`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;

CREATE INDEX `fk_Messaggio_Conversazione1_idx` ON `BachecaElettronicadb`.`Messaggio`
(`Conversazione_Codice` ASC) VISIBLE;


-- -----------------------------------------------------
-- Table `BachecaElettronicadb`.`Nota`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `BachecaElettronicadb`.`Nota` ;
```

```
CREATE TABLE IF NOT EXISTS `BachecaElettronicadb`.`Nota` (
  `ID` INT NOT NULL AUTO_INCREMENT,
  `Testo` VARCHAR(45) NOT NULL,
  `Annuncio_Codice` INT NOT NULL,
  PRIMARY KEY (`ID`, `Annuncio_Codice`),
  CONSTRAINT `fk_Nota_Annuncio1`
    FOREIGN KEY (`Annuncio_Codice`)
    REFERENCES `BachecaElettronicadb`.`Annuncio` (`Codice`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;

CREATE UNIQUE INDEX `Annuncio_Codice_UNIQUE` ON `BachecaElettronicadb`.`Nota` (`Annuncio_Codice` ASC) VISIBLE;

CREATE INDEX `fk_Nota_Annuncio1_idx` ON `BachecaElettronicadb`.`Nota` (`Annuncio_Codice` ASC) VISIBLE;


-- -----------------------------------------------------
-- Table `BachecaElettronicadb`.`Notifica`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `BachecaElettronicadb`.`Notifica` ;

CREATE TABLE IF NOT EXISTS `BachecaElettronicadb`.`Notifica` (
  `Codice_Notifica` INT NOT NULL AUTO_INCREMENT,
  `Codice` INT NOT NULL,
  `Tipo_Notifica` VARCHAR(45) NOT NULL,
  `Username_Utente` VARCHAR(45) NOT NULL,
  `Data` DATETIME NOT NULL,
  `Messaggio` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`Codice_Notifica`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
```

```
COLLATE = utf8mb4_0900_ai_ci;

CREATE UNIQUE INDEX `Codice_Notifica_UNIQUE` ON `BachecaElettronicadb`.`Notifica`
(`Codice_Notifica` ASC) VISIBLE;


-- -----------------------------------------------------
-- Table `BachecaElettronicadb`.`RecapitoNonPreferito`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `BachecaElettronicadb`.`RecapitoNonPreferito` ;

CREATE TABLE IF NOT EXISTS `BachecaElettronicadb`.`RecapitoNonPreferito` (
  `Recapito` VARCHAR(40) NOT NULL,
  `Tipo` ENUM('email', 'cellulare', 'social', 'sms') NOT NULL,
  `InformazioneAnagrafica_CF` VARCHAR(16) NOT NULL,
  PRIMARY KEY (`InformazioneAnagrafica_CF`, `Recapito`),
  CONSTRAINT `fk_RecapitoNonPreferito_InformazioneAnagrafica1`
    FOREIGN KEY (`InformazioneAnagrafica_CF`)
    REFERENCES `BachecaElettronicadb`.`InformazioneAnagrafica` (`CF`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;

CREATE INDEX `fk_RecapitoNonPreferito_InformazioneAnagrafica1_idx` ON
`BachecaElettronicadb`.`RecapitoNonPreferito` (`InformazioneAnagrafica_CF` ASC) VISIBLE;


-- -----------------------------------------------------
-- Table `BachecaElettronicadb`.`Report`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `BachecaElettronicadb`.`Report` ;

CREATE TABLE IF NOT EXISTS `BachecaElettronicadb`.`Report` (
  `Codice` INT NOT NULL AUTO_INCREMENT,
```

```
  `UCC_Username` VARCHAR(45) NOT NULL,
  `ImportoTotale` INT NOT NULL,
  `NumeroAnnunci` INT NOT NULL,
  `NumeroCarta` VARCHAR(16) NOT NULL,
  `Data` DATE NOT NULL,
  `Amministratore_Username` VARCHAR(45) NOT NULL,
  `Riscosso_Amministratore` TINYINT NOT NULL DEFAULT '0',
  `Importo_Amministratore` INT NOT NULL DEFAULT '0',
  `Importo_UCC` INT NOT NULL DEFAULT '0',
  PRIMARY KEY (`Codice`),
  CONSTRAINT `fk_Report_UCC1`
    FOREIGN KEY (`UCC_Username`)
    REFERENCES `BachecaElettronicadb`.`UCC` (`Username`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;

CREATE UNIQUE INDEX `Codice_UNIQUE` ON `BachecaElettronicadb`.`Report` (`Codice` ASC) VISIBLE;

CREATE INDEX `fk_Report_UCC1_idx` ON `BachecaElettronicadb`.`Report` (`UCC_Username` ASC) VISIBLE;


-- -----------------------------------------------------
-- Table `BachecaElettronicadb`.`Seguito-UCC`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `BachecaElettronicadb`.`Seguito-UCC` ;

CREATE TABLE IF NOT EXISTS `BachecaElettronicadb`.`Seguito-UCC` (
  `Annuncio_Codice` INT NOT NULL,
  `UCC_Username` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`Annuncio_Codice`, `UCC_Username`),
  CONSTRAINT `fk_Seguito-UCC_Annuncio1`
```

```
    FOREIGN KEY (`Annuncio_Codice`)
    REFERENCES `BachecaElettronicadb`.`Annuncio` (`Codice`),
  CONSTRAINT `fk_Seguito-UCC_UCC1`
    FOREIGN KEY (`UCC_Username`)
    REFERENCES `BachecaElettronicadb`.`UCC` (`Username`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;

CREATE INDEX `fk_Seguito-UCC_UCC1_idx` ON `BachecaElettronicadb`.`Seguito-UCC`
(`UCC_Username` ASC) VISIBLE;

CREATE INDEX `fk_Seguito-UCC_Annuncio1_idx` ON `BachecaElettronicadb`.`Seguito-UCC`
(`Annuncio_Codice` ASC) VISIBLE;


-- -----------------------------------------------------
-- Table `BachecaElettronicadb`.`Seguito-USCC`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `BachecaElettronicadb`.`Seguito-USCC` ;

CREATE TABLE IF NOT EXISTS `BachecaElettronicadb`.`Seguito-USCC` (
  `Annuncio_Codice` INT NOT NULL,
  `USCC_Username` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`Annuncio_Codice`, `USCC_Username`),
  CONSTRAINT `fk_Seguito-USCC_Annuncio1`
    FOREIGN KEY (`Annuncio_Codice`)
    REFERENCES `BachecaElettronicadb`.`Annuncio` (`Codice`),
  CONSTRAINT `fk_Seguito-USCC_USCC1`
    FOREIGN KEY (`USCC_Username`)
    REFERENCES `BachecaElettronicadb`.`USCC` (`Username`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;
```

```
CREATE INDEX `fk_Seguito-USCC_Annuncio1_idx` ON `BachecaElettronicadb`.`Seguito-USCC` (`Annuncio_Codice` ASC) VISIBLE;

CREATE INDEX `fk_Seguito-USCC_USCC1_idx` ON `BachecaElettronicadb`.`Seguito-USCC` (`USCC_Username` ASC) VISIBLE;


-- -----------------------------------------------------
-- Table `BachecaElettronicadb`.`Tracciato`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `BachecaElettronicadb`.`Tracciato` ;

CREATE TABLE IF NOT EXISTS `BachecaElettronicadb`.`Tracciato` (
  `Conversazione_Codice` INT NOT NULL,
  `StoricoConversazione_ID` INT NOT NULL,
  PRIMARY KEY (`Conversazione_Codice`, `StoricoConversazione_ID`),
  CONSTRAINT `fk_Tracciato_Conversazione1`
    FOREIGN KEY (`Conversazione_Codice`)
    REFERENCES `BachecaElettronicadb`.`Conversazione` (`Codice`)
    ON DELETE RESTRICT
    ON UPDATE RESTRICT,
  CONSTRAINT `fk_Tracciato_StoricoConversazione1`
    FOREIGN KEY (`StoricoConversazione_ID`)
    REFERENCES `BachecaElettronicadb`.`StoricoConversazione` (`ID`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;

CREATE INDEX `fk_Tracciato_StoricoConversazione1_idx` ON `BachecaElettronicadb`.`Tracciato` (`StoricoConversazione_ID` ASC) VISIBLE;

CREATE INDEX `fk_Tracciato_Conversazione1_idx` ON `BachecaElettronicadb`.`Tracciato` (`Conversazione_Codice` ASC) VISIBLE;
```

Codice del Front-End

- **main.c**

```
#include "defines.h"

struct configuration conf;
static MYSQL *conn;

typedef enum {
	ADMINISTRATOR = 1,
	UCC,
	USCC,
	FAILED_LOGIN
} role_t;

static role_t attempt_login(MYSQL *login_conn, char *username, char *password){
	MYSQL_STMT *login_stored_procedure;
	MYSQL_BIND param[3];
	int role = 0;

	if(!setup_prepared_stmt(&login_stored_procedure, "call login(?, ?, ?)", login_conn)) {
		finish_with_stmt_error(login_conn, login_stored_procedure, "Unable to initialize ad statement", true);
		goto err2;
	}

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

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

	param[1].buffer_type = MYSQL_TYPE_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_stored_procedure, param) != 0) {
                print_stmt_error(login_stored_procedure, "Could not bind parameters for login");
                goto err;
        }

        if (mysql_stmt_execute(login_stored_procedure) != 0) {
                print_stmt_error(login_stored_procedure, NULL);
                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_stored_procedure, param)) {
                print_stmt_error(login_stored_procedure, "Could not bind result");
                goto err;
        }


        if (mysql_stmt_fetch(login_stored_procedure)) {
                print_stmt_error(login_stored_procedure, "Could not fetch result");
                goto err;
        }

        mysql_stmt_close(login_stored_procedure);
```

```
        return role;

        err:
                mysql_stmt_close(login_stored_procedure);

        err2:
                return FAILED_LOGIN;
}

void sign_up(MYSQL *account_conn) {

        MYSQL_STMT *prepared_stmt;

        char username[MAX_LENGHT_USERNAME], password[MAX_LENGHT_USERNAME],
cf[MAX_CF_LENGHT], surname[20], name[20], residential_address[20], billing_address[20],
type_favorite_contact[20], favorite_contact[40], type_not_favorite_contact[20],
not_favorite_contact[40];
        char cap_string[5];
        int cap;

        char *list_choice_type_it[] = {"email", "cellulare", "social", "sms"};
        char *list_choice_type_en[] = {"email", "mobile phone", "social", "sms"};
        int lenght_choice_type = 4;

        char favorite_choice[20], not_favorite_choice[20];

        bool request, request_billing_address, is_null;

        is_null = 1;

        print_color("Username: ", "yellow", ' ', false, false, false, false);
        getInput(MAX_LENGHT_USERNAME, username, false);

        print_color("Password: ", "yellow", ' ', false, false, false, false);
```

```
        getInput(MAX_LENGHT_USERNAME, password, true);

        print_color("Tax code: ", "yellow", ' ', false, false, false, false);
        getInput(MAX_CF_LENGHT, cf, false);
5       cf[16] = '\0';

        print_color("Surname: ", "yellow", ' ', false, false, false, false);
        getInput(20, surname, false);

10      print_color("Name: ", "yellow", ' ', false, false, false, false);
        getInput(20, name, false);

        print_color("Residential address: ", "yellow", ' ', false, false, false, false);
        getInput(20, residential_address, false);
15
        print_color("CAP: ", "light cyan", ' ', false, false, false, false);
        getInput(5, cap_string, false);
        cap = atoi(cap_string);

20      request_billing_address = yesOrNo("Do you want to edit your billing address?", 'y', 'n');

        if(request_billing_address) {
                print_color("Billing address: ", "yellow", ' ', false, false, false, false);
                getInput(20, billing_address, false);
25      }

        while(1) {
                print_color("  Which do you choose to enter as your favorite contact?", "orange", ' ',
true, true, false, false);
30
                for(int i=0; i<lenght_choice_type; i++) {
                        print_color(" - ", "cyan", ' ', false, false, false, false);
                        print_color(list_choice_type_en[i], "light blue", ' ', false, true, false, false);
                }
```

```
                // Take input
                getInput(20, favorite_choice, false);
                for(int i=0; i<lenght_choice_type; i++) {

                        if(strcmp(list_choice_type_en[i], favorite_choice) == 0) {
                                sprintf(type_favorite_contact, "%s", list_choice_type_it[i]);
                                type_favorite_contact[strlen(list_choice_type_it[i])] = '\0';

                                print_color("Contact: ", "yellow", ' ', false, false, false, false);
                                getInput(40, favorite_contact, false);

                                goto execution_favorite_contact;
                        }
                }
        }

        execution_favorite_contact:

        while(1) {
                print_color("  Which do you choose to enter as your not favorite contact?", "orange", '
', true, true, false, false);

                for(int i=0; i<lenght_choice_type; i++) {
                        print_color(" - ", "cyan", ' ', false, false, false, false);
                        print_color(list_choice_type_en[i], "light blue", ' ', false, true, false, false);
                }

                // Take input
                getInput(20, not_favorite_choice, false);
                for(int i=0; i<lenght_choice_type; i++) {

                        if(strcmp(list_choice_type_en[i], not_favorite_choice) == 0) {
                                sprintf(type_not_favorite_contact, "%s", list_choice_type_it[i]);
```

```
                        type_not_favorite_contact[strlen(list_choice_type_it[i])] = '\0';

                        print_color("Contact: ", "yellow", ' ', false, false, false, false);
                        getInput(40, not_favorite_contact, false);
                        goto execution_not_favorite_contact;
                    }
                }
            }

            execution_not_favorite_contact:

            request = yesOrNo("Do you want to add your credit card number?", 'y', 'n');

            if(request) {
                char card_number[17], cvc_string[3], day_string[3], month_string[3], year_string[5];
                MYSQL_TIME expiration_date;
                int cvc;

                MYSQL_BIND param[15];

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

                restart_card_number:
                print_color("Card Number: ", "yellow", ' ', false, false, false, false);
                getInput(17, card_number, false);
                card_number[16] = '\0';
                if (strlen(card_number)<16) {
                    print_color("   Error Card Number", "light red", ' ', false, true, false, true);
                    goto restart_card_number;
                }

                restart_cvc:
                print_color("CVC: ", "yellow", ' ', false, false, false, false);
                getInput(3, cvc_string, false);
```

```
cvc_string[3] = '\0';
cvc = atoi(cvc_string);
if (strlen(cvc_string)<3) {
        print_color("   Error CVC", "light red", ' ', false, true, false, true);
        goto restart_cvc;
}

print_color("Enter the expiration date in numeric format", "orange", ' ', true, true, false, false);

restart_day:
print_color("Day: ", "light cyan", ' ', false, false, false, false);
getInput(3, day_string, false);
day_string[2] = '\0';
if(atoi(day_string)>0 && atoi(day_string)<32) {
        print_color("   Error Day", "light red", ' ', false, true, false, true);
        goto restart_day;
}
expiration_date.day = atoi(day_string);

restart_month:
print_color("Month: ", "light cyan", ' ', false, false, false, false);
getInput(3, month_string, false);
month_string[2] = '\0';
if(atoi(month_string)>0 && atoi(month_string)<13) {
        print_color("   Error Month", "light red", ' ', false, true, false, true);
        goto restart_month;
}
expiration_date.month= atoi(month_string);

restart_year:
print_color("Year: ", "light cyan", ' ', false, false, false, false);
getInput(5, year_string, false);
year_string[4] = '\0';
```

```
        if(atoi(year_string)>=2021 && atoi(year_string)<2051) {
                print_color("   Error Year", "light red", ' ', false, true, false, true);
                goto restart_year;
        }
5       expiration_date.year= atoi(year_string);

        if(!request_billing_address)
                param[10].is_null = &is_null;

10      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;
15      param[1].buffer = password;
        param[1].buffer_length = strlen(surname);

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

        param[3].buffer_type = MYSQL_TYPE_DATE;
        param[3].buffer = (char *) &expiration_date;
        param[3].buffer_length = sizeof(expiration_date);
25
        param[4].buffer_type = MYSQL_TYPE_LONG;
        param[4].buffer = &cvc;
        param[4].buffer_length = sizeof(cvc);

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

        param[6].buffer_type = MYSQL_TYPE_VAR_STRING;
```

```
param[6].buffer = surname;
param[6].buffer_length = strlen(surname);

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

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

param[9].buffer_type = MYSQL_TYPE_LONG;
param[9].buffer = &cap;
param[9].buffer_length = sizeof(favorite_contact);

param[10].buffer_type = MYSQL_TYPE_VAR_STRING;
param[10].buffer = billing_address;
param[10].buffer_length = strlen(billing_address);

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

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

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

param[14].buffer_type = MYSQL_TYPE_VAR_STRING;
param[14].buffer = not_favorite_contact;
param[14].buffer_length = strlen(not_favorite_contact);
```

```
        if (!setup_prepared_stmt(&prepared_stmt, "call registra_utente_UCC (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)", account_conn))
            finish_with_stmt_error(account_conn, prepared_stmt, "Unable to initialize ad statement", true);

        if (mysql_stmt_bind_param(prepared_stmt, param) != 0)
            finish_with_stmt_error(account_conn, prepared_stmt, "Unable to bind parameters to create user\n", true);
    } else {
        MYSQL_BIND param[12];

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

        if(!request_billing_address)
            param[7].is_null = &is_null;

        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(surname);

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

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

        param[4].buffer_type = MYSQL_TYPE_VAR_STRING;
```

```
        param[4].buffer = name;
        param[4].buffer_length = strlen(name);

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

        param[6].buffer_type = MYSQL_TYPE_LONG;
        param[6].buffer = &cap;
        param[6].buffer_length = sizeof(favorite_contact);

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

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

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

        param[10].buffer_type = MYSQL_TYPE_VAR_STRING;
        param[10].buffer = type_not_favorite_contact;
        param[10].buffer_length = strlen(type_not_favorite_contact);

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

        if (!setup_prepared_stmt(&prepared_stmt, "call registra_utente_USCC
(?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)", account_conn))
```

```
                finish_with_stmt_error(account_conn, prepared_stmt, "Unable to initialize ad statement", true);

        if (mysql_stmt_bind_param(prepared_stmt, param) != 0)
            finish_with_stmt_error(account_conn, prepared_stmt, "Unable to bind parameters to create user\n", true);
    }

    if (mysql_stmt_execute(prepared_stmt) != 0) {
        print_stmt_error(prepared_stmt, NULL);
        mysql_stmt_close(prepared_stmt);
        return;
    }
    else
        print_color("   Successfully created!", "light blue", ' ', false, true, false, true);

    mysql_stmt_close(prepared_stmt);

    // Sign in account
    strcat(conf.username, username);
    strcat(conf.password, password);

    if(request)
        run_as_ucc(account_conn, conf);
    else
        run_as_uscc(account_conn, conf);

    return;
}


int main(void){
    role_t role;
```

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

    conn = mysql_init(NULL);        //connection initialized

    if (conn == NULL) {
        fprintf(stderr, "%s\n", mysql_error(conn));
        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, "%s\n", mysql_error(conn));
        mysql_close(conn);
        exit(EXIT_FAILURE);
    }

    printf("\n  \t\t\033[40m\033[1;31m@@@@@@@@ @@@@@@@@ @@@@@@@@ @@ @@ @@@@@@@@ @@@@@@@@ @@@@@@@@\033[0m\n");
    printf("  \t\t\033[40m\033[1;31m@@  @ @@  @@ @@  @@ @@ @@ @@  @@ @@\033[0m\n");
    printf("  \t\t\033[40m\033[1;31m@@@@@@@@ @@@@@@@@ @@ @@@@@@@@ @@@@@@@@ @@  @@@@@@@@\033[0m\n");
    printf("  \t\t\033[40m\033[1;31m@@  @ @@  @@ @@  @@ @@ @@ @@  @@ @@\033[0m\n");
    printf("  \t\t\033[40m\033[1;31m@@@@@@@@ @@  @@ @@@@@@@@ @@ @@ @@@@@@@@ @@@@@@@@ @@  @@\033[0m\n");

    printf("\n\t\e[34m\e[1m@@@@@@@@ @@  @@@@@@@ @@@@@@@@ @@@@@@@ @@@@@@@ @@@@ @@ @@ @@@@@@@@ @@@@@@@@\e[22m\e[39m\n");
```

```
        printf("\t\e[34m\e[1m@@      @@      @@      @@      @@      @@  @@  @@  @@
@@ @@  @@  @@  @@      @@   @@\e[22m\e[39m\n");
        printf("\t\e[34m\e[1m@@@@@@@   @@          @@@@@@@      @@      @@
@@@@@@@ @@  @@  @@  @@ @@  @@  @@      @@@@@@@\e[22m\e[39m\n");
        printf("\t\e[34m\e[1m@@      @@      @@      @@      @@      @@  @@  @@  @@
@@  @@@@  @@  @@      @@   @@\e[22m\e[39m\n");
        printf("\t\e[34m\e[1m@@@@@@@  @@@@@@@@  @@@@@@@   @@      @@
@@  @@ @@@@@@@ @@   @@@ @@ @@@@@@@ @@   @@\e[22m\e[39m\n\n");

        if (yesOrNo("\nDo you want to sign up?", 'y', 'n')) {
                sign_up(conn);
                goto restart;
        }

        goto start;

        restart: if (yesOrNo("Do you want continue?", 'y', 'n') == false) goto exit;

        start:
                print_color("      ENTER YOUR USERNAME AND PASSWORD", "pink", ' ', true,
true, false, true);

                print_color("Username: ", "yellow", ' ', false, false, true, false);
                getInput(45, conf.username, false);
                print_color("Password: ", "light cyan", ' ', false, false, false, false);
                getInput(45, conf.password, true);

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

        switch(role){
                case ADMINISTRATOR:
                        run_as_administrator(conn, conf);
                        break;
                case UCC:
```

```
                run_as_ucc(conn, conf);
                break;
        case USCC:
                run_as_uscc(conn, conf);
                break;
        case FAILED_LOGIN:
                print_color("Incorrect Username or Password!", "red", ' ', false, true, false, true);
                goto restart;
                break;
        default:
                print_color("Error to login", "red", ' ', false, true, false, true);
                abort();        //it may not delete temporary files and may not flush stream buffer
    }
    goto restart;

    exit:

        print_color("   Bye!\n", "light blue", ' ', true, true, true, true);
        mysql_close(conn);

    return 0;
}
```

- **ucc.c**

```
#include "defines.h"

struct configuration conf;
MYSQL *conn;
int ad_code;
```

```
void new_ad() {
    MYSQL_STMT *prepared_stmt;
    MYSQL_BIND param[5];

    char description[100], photo[9], category[20], amount_string[6];
    int amount;
    bool request, is_null;

    is_null = 1;

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

    print_color("Ad description: ", "yellow", ' ', false, false, false, false);
    strcpy(description, getInput(100, description, false));

    request = yesOrNo("Do you want to add a photo?", 'y', 'n');

    if (request)
        strcpy(photo, "Presente");
    else
        param[2].is_null = &is_null;

    print_color("Category: ", "yellow", ' ', false, false, false, false);
    strcpy(category, getInput(20, category, false));

    while(1) {
        print_color("Amount: ", "light cyan", ' ', false, false, false, false);
        getInput(6, amount_string, false);
        amount = atoi(amount_string);

        if(amount < 0)
            print_error(NULL, "The amount must be positive");
```

```
            else
                break;
        }

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

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

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

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

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


        if (!setup_prepared_stmt(&prepared_stmt, "call inserimentoNuovoAnnuncio(?, ?, ?, ?, ?)", conn))
            finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize ad statement", true);

        if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
            finish_with_stmt_error(conn, prepared_stmt, "Unable to bind parameters for a new ad\n", true);
        }
```

```
        if (mysql_stmt_execute(prepared_stmt) != 0)
                print_stmt_error(prepared_stmt, NULL);
        else
                print_color("   Successfully generated!", "light blue", ' ', false, true, false, true);

        mysql_stmt_close(prepared_stmt);
        return;
}

bool view_ad(char *username, bool check_owner_value) {
        MYSQL_STMT *prepared_stmt;
        MYSQL_BIND param[3];

        int check_owner;                                                        //
Parameters are used to check
        char ad_code_string[MAX_AD_CODE_LENGHT];                        //  of  the
owner of the ads for other functions

        if(check_owner_value) {

                check_owner = 1;

                print_color("Ad code: ", "light cyan", ' ', false, false, false, false);
                getInput(MAX_AD_CODE_LENGHT, ad_code_string, false);
                ad_code = atoi(ad_code_string);

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


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

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

            param[2].buffer_type = MYSQL_TYPE_LONG;
            param[2].buffer = &check_owner;
            param[2].buffer_length = sizeof(check_owner);
            goto execution;
        }

        check_owner = 0;

        char ucc_username[45];
        bool request, is_null;

        is_null = 1;
        check_owner = 0;

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

        request = yesOrNo("Do you have any preference on ad?", 'y', 'n');

        if(request == true) {
            print_color("Ad code: ", "light cyan", ' ', false, false, false, false);
            getInput(MAX_AD_CODE_LENGHT, ad_code_string, false);
            ad_code = atoi(ad_code_string);
            param[1].is_null = &is_null;
            goto execution_ucc_views_ad;
        }
        if(request == false){
            param[0].is_null = &is_null;
        }

        request = yesOrNo("Do you have any preference on user?", 'y', 'n');
```

```
        if(request == true) {
                print_color("Username: ", "yellow", ' ', false, false, false, false);
                getInput(45, ucc_username, false);
        }
        if(request == false){
                param[1].is_null = &is_null;
        }

        execution_ucc_views_ad:

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

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

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


        execution:

        if (!setup_prepared_stmt(&prepared_stmt, "call visualizzaAnnuncio (?, ?, ?)", conn))
                finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize ad statement",
true);

        if (mysql_stmt_bind_param(prepared_stmt, param) != 0)
                finish_with_stmt_error(conn, prepared_stmt, "Unable to bind parameters to view ads\
n", true);
```

```
        if (mysql_stmt_execute(prepared_stmt) != 0) {
                print_stmt_error(prepared_stmt, NULL);
                mysql_stmt_close(prepared_stmt);
                return false;
        }

        if (!check_owner_value)
                dump_result_set(conn, prepared_stmt, "Ads list\n");          // dump the result set

        mysql_stmt_close(prepared_stmt);
        return true;
}

void view_category_online() {
        MYSQL_STMT *prepared_stmt;

        if (!setup_prepared_stmt(&prepared_stmt, "call visualizzaCategoria()", conn))
                finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize view category statement\n", true);

        if (mysql_stmt_execute(prepared_stmt) != 0)
                print_stmt_error (prepared_stmt, "An error occurred while viewing categories.");

        dump_result_set(conn, prepared_stmt, "Online Category list\n");              // dump the result set
        mysql_stmt_next_result(prepared_stmt);

        mysql_stmt_close(prepared_stmt);
        return;
}

void remove_ad() {
```

```
        // Check if the user is the owner
        if(view_ad(conf.username, true) == false)
                return;

        MYSQL_STMT *prepared_stmt;
        MYSQL_BIND param[2];

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

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

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

        if (!setup_prepared_stmt(&prepared_stmt, "call rimuoviAnnuncio (?, ?)", conn))
                finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize ad statement",
true);

        if (mysql_stmt_bind_param(prepared_stmt, param) != 0)
                finish_with_stmt_error(conn, prepared_stmt, "Unable to bind parameters to view ads\
n", true);

        if (mysql_stmt_execute(prepared_stmt) != 0)
                print_stmt_error(prepared_stmt, NULL);
        else
                print_color("   Successfully removed!", "light blue", ' ', false, true, false, true);

        mysql_stmt_close(prepared_stmt);

        ad_code = -1;
        return;
```

```
}

void ad_sold() {

    // Check if the user is the owner
    if(view_ad(conf.username, true) == false)
        return;

    MYSQL_STMT *prepared_stmt;
    MYSQL_BIND param[2];

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

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

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

    if (!setup_prepared_stmt(&prepared_stmt, "call vendutoAnnuncio (?, ?)", conn))
        finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize ad statement", true);

    if (mysql_stmt_bind_param(prepared_stmt, param) != 0)
        finish_with_stmt_error(conn, prepared_stmt, "Unable to bind parameters to view ads\n", true);

    if (mysql_stmt_execute(prepared_stmt) != 0)
        print_stmt_error(prepared_stmt, NULL);
    else
        print_color("    Successfully sold!", "light blue", ' ', false, true, false, true);
```

```
        mysql_stmt_close(prepared_stmt);

        ad_code = -1;
        return;
}

bool view_personal_information(char cf_owner[], char username_owner[], int check_owner) {
        MYSQL_STMT *prepared_stmt;

        // Check information for other functions
        if(check_owner == 1){
                MYSQL_BIND param[3];

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

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

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

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

                if (!setup_prepared_stmt(&prepared_stmt, "call visualizzaInfoAnagrafiche (?, ?, ?)", conn))
                        finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize ad statement", true);

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

```
                finish_with_stmt_error(conn, prepared_stmt, "Unable to bind parameters to view information\n", true);

                goto execution;
        }


        // View UCC information
        if(yesOrNo("Do you want to see your account information?", 'y', 'n')) {
                MYSQL_BIND param[1];
                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 (!setup_prepared_stmt(&prepared_stmt, "call visualizza_Info_Utente (?)", conn))
                        finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize ad statement", true);

                if (mysql_stmt_bind_param(prepared_stmt, param) != 0)
                        finish_with_stmt_error(conn, prepared_stmt, "Unable to bind parameters to view information\n", true);

                goto execution;
        }


        // View personal information
        MYSQL_BIND param[3];

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

        char cf[MAX_CF_LENGHT];
```

```
        bool is_null;

        is_null = 1;

        print_color("Tax code: ", "yellow", ' ', false, false, false, false);
        getInput(MAX_CF_LENGHT, cf, false);
        cf[16] = '\0';

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

        param[1].is_null = &is_null;          // check_owner
        param[2].is_null = &is_null;          // username

        if (!setup_prepared_stmt(&prepared_stmt, "call visualizzaInfoAnagrafiche (?, ?, ?)", conn))
                finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize ad statement", true);


        if (mysql_stmt_bind_param(prepared_stmt, param) != 0)
                finish_with_stmt_error(conn, prepared_stmt, "Unable to bind parameters to view information\n", true);

        execution:
        if (mysql_stmt_execute(prepared_stmt) != 0) {
                print_stmt_error(prepared_stmt, NULL);
                mysql_stmt_close(prepared_stmt);
                return false;
        }

        if(check_owner != 1)
                dump_result_set(conn, prepared_stmt, "Personal Information\n");          // dump the result set
```

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

void edit_personal_information(char cf_set_favorite[], char type_set_favorite[], char contact_set_favorites[]) {
        MYSQL_STMT *prepared_stmt;
        MYSQL_BIND param[8];

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

        bool is_null = 1;

        // Set contact as favorite
        if(cf_set_favorite != NULL && type_set_favorite != NULL && contact_set_favorites != NULL) {
                param[0].buffer_type = MYSQL_TYPE_VAR_STRING;
                param[0].buffer = cf_set_favorite;
                param[0].buffer_length = strlen(cf_set_favorite);

                param[1].is_null = &is_null;
                param[2].is_null = &is_null;
                param[3].is_null = &is_null;
                param[4].is_null = &is_null;
                param[5].is_null = &is_null;

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

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

```
            goto execution;
        }

        char cf[MAX_CF_LENGHT], surname[20], name[20], residential_address[20], cap_string[6], billing_address[20], type_favorite_contact[20], favorite_contact[40];
        int cap;

        char *list_choice_type_it[] = {"email", "cellulare", "social", "sms"};
        char *list_choice_type_en[] = {"email", "mobile phone", "social", "sms"};
        int lenght_choice_type = 4;

        char choice[20];

        bool request;

        print_color("Tax code: ", "yellow", ' ', false, false, false, false);
        getInput(MAX_CF_LENGHT, cf, false);
        cf[16] = '\0';

        // Check if the user has this tax code
        if(!view_personal_information(cf, conf.username, 1))
            return;

        request = yesOrNo("Do you want to edit your surname?", 'y', 'n');

        if(request == true) {
            print_color("Surname: ", "yellow", ' ', false, false, false, false);
            getInput(20, surname, false);
        }
        if(request == false){
            param[1].is_null = &is_null;
        }
```

```
request = yesOrNo("Do you want to edit your name?", 'y', 'n');

if(request == true) {
        print_color("Name: ", "yellow", ' ', false, false, false, false);
        getInput(20, name, false);
}
if(request == false){
        param[2].is_null = &is_null;
}

request = yesOrNo("Do you want to edit your residential address?", 'y', 'n');

if(request == true) {
        print_color("Residential address: ", "yellow", ' ', false, false, false, false);
        getInput(20, residential_address, false);
}
if(request == false){
        param[3].is_null = &is_null;
}

request = yesOrNo("Do you want to edit your CAP?", 'y', 'n');

if(request == true) {
        print_color("CAP: ", "light cyan", ' ', false, false, false, false);
        getInput(6, cap_string, false);
        cap_string[5] = '\0';
        cap = atoi(cap_string);
}
if(request == false){
        param[4].is_null = &is_null;
}
```

```
        request = yesOrNo("Do you want to edit your billing address?", 'y', 'n');

        if(request == true) {
                print_color("Billing address: ", "yellow", ' ', false, false, false, false);
                getInput(20, billing_address, false);
        }
        if(request == false){
                param[5].is_null = &is_null;
        }


        request = yesOrNo("Do you want to edit your favorite contact?", 'y', 'n');

        while(1) {
                        if (request)
                                print_color("  Which do you choose to edit?", "orange", ' ', true, true, false, false);
                        else {
                                param[6].is_null = &is_null;
                                param[7].is_null = &is_null;
                                break;
                        }

                        for(int i=0; i<lenght_choice_type; i++) {
                                print_color(" - ", "cyan", ' ', false, false, false, false);
                                print_color(list_choice_type_en[i], "light blue", ' ', false, true, false, false);
                        }

                        // Take input
                        getInput(20, choice, false);
                        for(int i=0; i<lenght_choice_type; i++) {
```

```
                        if(strcmp(list_choice_type_en[i], choice) == 0) {
                            sprintf(type_favorite_contact, "%s", list_choice_type_it[i]);
                            type_favorite_contact[strlen(list_choice_type_it[i])] = '\0';

                            print_color("Contact: ", "yellow", ' ', false, false, false, false);
                            getInput(40, favorite_contact, false);

                            goto execution_editing_information;
                        }
                    }
                }

                execution_editing_information:


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

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

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

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

            param[4].buffer_type = MYSQL_TYPE_LONG;
            param[4].buffer = &cap;
```

```
        param[4].buffer_length = sizeof(cap);

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

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

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

        execution:
        if (!setup_prepared_stmt(&prepared_stmt, "call modificaInfoAnagrafiche (?, ?, ?, ?, ?, ?, ?, ?)", conn))
                finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize ad statement", true);

        if (mysql_stmt_bind_param(prepared_stmt, param) != 0)
                finish_with_stmt_error(conn, prepared_stmt, "Unable to bind parameters to edit information\n", true);

        if (mysql_stmt_execute(prepared_stmt) != 0)
                print_stmt_error(prepared_stmt, NULL);
        else
                print_color("   Successfully edited!", "light blue", ' ', false, true, false, true);

        mysql_stmt_close(prepared_stmt);
        return;
}

void edit_photo_ad() {
```

```
        MYSQL_STMT *prepared_stmt;
        MYSQL_BIND param[3];

        char ad_code_string[MAX_AD_CODE_LENGHT], photo[9];
        int ad_code_edit;

        bool request, is_null;

        is_null = 1;

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

        print_color("Ad code: ", "light cyan", ' ', false, false, false, false);
        strcpy(ad_code_string, getInput(MAX_AD_CODE_LENGHT, ad_code_string, false));
        ad_code_edit = atoi(ad_code_string);

        request = yesOrNo("Do you want to add a photo to an ad?", 'y', 'n');

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

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


        if(request) {
                strcpy(photo, "Presente");
        }
        else
                param[2].is_null = &is_null;

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

```
        param[2].buffer = photo;
        param[2].buffer_length = strlen(photo);


    if (!setup_prepared_stmt(&prepared_stmt, "call modificaFotoAnnuncio (?, ?, ?)", conn))
        finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize ad statement", true);

        if (mysql_stmt_bind_param(prepared_stmt, param) != 0)
    finish_with_stmt_error(conn, prepared_stmt, "Unable to bind parameters to view information\n", true);

        if (mysql_stmt_execute(prepared_stmt) != 0)
            print_stmt_error(prepared_stmt, NULL);
        else
            print_color("    Successfully edited!", "light blue", ' ', false, true, false, true);

        mysql_stmt_close(prepared_stmt);
        return;
}

void insert_remove_comment() {
        MYSQL_STMT *prepared_stmt;
        MYSQL_BIND param[3];

        char ad_code_string[MAX_AD_CODE_LENGHT], id_comment_string[MAX_COMMENT_ID_LENGHT], text[45];
        int ad_code_comment, id_comment;

        bool request, is_null;

        is_null = 1;

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

```
        print_color("Ad code: ", "light cyan", ' ', false, false, false, false);
        strcpy(ad_code_string, getInput(MAX_AD_CODE_LENGHT, ad_code_string, false));
        ad_code_comment = atoi(ad_code_string);

        request = yesOrNo("Do you want to add a comment to an ad?", 'y', 'n');

        if(request) {
                print_color("Text of the comment: ", "yellow", ' ', false, false, false, false);
                getInput(45, text, false);
                param[2].is_null = &is_null;
        } else {
                print_color("Comment ID to remove: ", "light cyan", ' ', false, false, false, false);
                strcpy(id_comment_string,          getInput(MAX_COMMENT_ID_LENGHT,
id_comment_string, false));
                id_comment = atoi(id_comment_string);
                param[0].is_null = &is_null;
        }

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

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

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


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

```
        finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize ad statement", true);

    if (mysql_stmt_bind_param(prepared_stmt, param) != 0)
        finish_with_stmt_error(conn, prepared_stmt, "Unable to bind parameters to add or remove comment\n", true);

    if (mysql_stmt_execute(prepared_stmt) != 0) {
        print_stmt_error(prepared_stmt, NULL);
        mysql_stmt_close(prepared_stmt);
        return;
    }

    if(request)
        print_color("   Successfully added!", "light blue", ' ', false, true, false, true);
    else
        print_color("   Successfully removed!", "light red", ' ', false, true, false, true);

    mysql_stmt_close(prepared_stmt);
    return;
}


void view_comment() {
    MYSQL_STMT *prepared_stmt;
    MYSQL_BIND param[1];

    char ad_code_string[MAX_AD_CODE_LENGHT];
    int ad_code_comment;

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

    print_color("Ad code: ", "light cyan", ' ', false, false, false, false);
    strcpy(ad_code_string, getInput(MAX_AD_CODE_LENGHT, ad_code_string, false));
```

```
	ad_code_comment = atoi(ad_code_string);

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

	if (!setup_prepared_stmt(&prepared_stmt, "call visualizzaCommento (?)", conn))
		finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize ad statement", true);

	if (mysql_stmt_bind_param(prepared_stmt, param) != 0)
		finish_with_stmt_error(conn, prepared_stmt, "Unable to bind parameters to view comments\n", true);

	if (mysql_stmt_execute(prepared_stmt) != 0)
		print_stmt_error(prepared_stmt, NULL);
	else
		dump_result_set(conn, prepared_stmt, "Comments\n");

	mysql_stmt_close(prepared_stmt);
	return;
}

void insert_remove_note() {
	MYSQL_STMT *prepared_stmt;
	MYSQL_BIND param[4];

	char ad_code_string[MAX_AD_CODE_LENGHT], id_note_string[MAX_NOTE_ID_LENGHT], text[45];
	int ad_code_note, id_note;

	bool request, is_null;

	is_null = 1;
```

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

print_color("Ad code: ", "light cyan", ' ', false, false, false, false);
strcpy(ad_code_string, getInput(MAX_AD_CODE_LENGHT, ad_code_string, false));
ad_code_note = atoi(ad_code_string);

request = yesOrNo("Do you want to add a note to the ad?", 'y', 'n');

if(request) {
	print_color("Text of the note: ", "yellow", ' ', false, false, false, false);
	getInput(45, text, false);
	param[2].is_null = &is_null;
} else {
	print_color("ID of the note to remove: ", "light cyan", ' ', false, false, false, false);
	strcpy(id_note_string, getInput(MAX_NOTE_ID_LENGHT, id_note_string, false));
	id_note = atoi(id_note_string);
	param[0].is_null = &is_null;
}

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

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

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

param[3].buffer_type = MYSQL_TYPE_VAR_STRING;
param[3].buffer = conf.username;
```

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


	if (!setup_prepared_stmt(&prepared_stmt, "call modificaNota (?, ?, ?, ?)", conn))
		finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize ad statement", true);

	if (mysql_stmt_bind_param(prepared_stmt, param) != 0)
		finish_with_stmt_error(conn, prepared_stmt, "Unable to bind parameters to add or remove note\n", true);

	if (mysql_stmt_execute(prepared_stmt) != 0) {
		print_stmt_error(prepared_stmt, NULL);
		mysql_stmt_close(prepared_stmt);
		return;
	}

	if(request)
		print_color("   Successfully added!", "light blue", ' ', false, true, false, true);
	else
		print_color("   Successfully removed!", "light red", ' ', false, true, false, true);

	mysql_stmt_close(prepared_stmt);
	return;
}

void view_note() {
	MYSQL_STMT *prepared_stmt;
	MYSQL_BIND param[1];

	char ad_code_string[MAX_AD_CODE_LENGHT];
	int ad_code_note;

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

```
        print_color("Ad code: ", "light cyan", ' ', false, false, false, false);
        strcpy(ad_code_string, getInput(MAX_AD_CODE_LENGHT, ad_code_string, false));
        ad_code_note = atoi(ad_code_string);

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

        if (!setup_prepared_stmt(&prepared_stmt, "call visualizzaNota (?)", conn))
                finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize ad statement", true);

        if (mysql_stmt_bind_param(prepared_stmt, param) != 0)
                finish_with_stmt_error(conn, prepared_stmt, "Unable to bind parameters to view note\n", true);

        if (mysql_stmt_execute(prepared_stmt) != 0)
                print_stmt_error(prepared_stmt, NULL);
        else
                dump_result_set(conn, prepared_stmt, "Note\n");

        mysql_stmt_close(prepared_stmt);
        return;
}

void insert_remove_contact() {

        char type[20], contact[40], cf[MAX_CF_LENGHT];
        int remove = 1;

        char *list_choice_type_it[] = {"email", "cellulare", "social", "sms"}, choice[20];
        char *list_choice_type_en[] = {"email", "mobile phone", "social", "sms"};
```

```
	int lenght_choice_type = 4, i;

	bool request, request_2, is_null;

	is_null = 1;

	print_color("Tax code: ", "yellow", ' ', false, false, false, false);
	getInput(MAX_CF_LENGHT, cf, false);

	// Check if the user has this tax code
	if(!view_personal_information(cf, conf.username, 1))
		return;

	request = yesOrNo("Do you want to add a contact?", 'y', 'n');

	while(1) {
		if (request)
			print_color("  Which do you choose to add?", "orange", ' ', false, true, false, false);
		else
			print_color("  Which do you choose to remove?", "orange", ' ', false, true, false, false);

		for(i=0; i<lenght_choice_type; i++) {
			print_color(" - ", "cyan", ' ', false, false, false, false);
			print_color(list_choice_type_en[i], "light blue", ' ', false, true, false, false);
		}

		// Take input
		getInput(20, choice, false);
		for(i=0; i<lenght_choice_type; i++) {

			if(strcmp(list_choice_type_en[i], choice) == 0) {
				sprintf(type, "%s", list_choice_type_it[i]);
```

```
                        type[strlen(list_choice_type_it[i])] = '\0';

                        print_color("Contact: ", "yellow", ' ', false, false, false, false);
                        getInput(40, contact, false);
                        goto execution;
                }
        }
}

execution:

if(request) {
        request_2 = yesOrNo("Do you want to set it as a favorite?", 'y', 'n');

        // If the user sends yes then it set as favorite
        if (request_2) {
                edit_personal_information(cf, type, contact);
                return;
        }
}

MYSQL_STMT *prepared_stmt;
MYSQL_BIND param[4];

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

// If the user wants to remove a contact
if(request)
        param[3].is_null= &is_null;

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

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

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

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


        if (!setup_prepared_stmt(&prepared_stmt, "call modifica_RecapitoNonPreferito (?, ?, ?, ?)",
conn))
                finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize ad statement",
true);

        if (mysql_stmt_bind_param(prepared_stmt, param) != 0)
                finish_with_stmt_error(conn, prepared_stmt, "Unable to bind parameters to add or
remove note\n", true);

        if (mysql_stmt_execute(prepared_stmt) != 0) {
                print_stmt_error(prepared_stmt, NULL);
                mysql_stmt_close(prepared_stmt);
                return;
        }

        if(request)
                print_color("   Successfully added!", "light blue", ' ', false, true, false, true);
        else
                print_color("   Successfully removed!", "light red", ' ', false, true, false, true);

        mysql_stmt_close(prepared_stmt);
```

```
        return;
}

void view_contact() {
    MYSQL_STMT *prepared_stmt;
    MYSQL_BIND param[2];

    char cf[MAX_CF_LENGHT];
    int favorite = 1;

    bool request, is_null;

    is_null = 1;

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

    print_color("Tax code: ", "yellow", ' ', false, false, false, false);
    getInput(MAX_CF_LENGHT, cf, false);

    request = yesOrNo("Do you want to see favorite contact?", 'y', 'n');

    // Set the var to null
    if(!request)
        param[1].is_null = &is_null;

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

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

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

```
        finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize ad statement", true);

    if (mysql_stmt_bind_param(prepared_stmt, param) != 0)
        finish_with_stmt_error(conn, prepared_stmt, "Unable to bind parameters to view contacts\n", true);

    if (mysql_stmt_execute(prepared_stmt) != 0)
        print_stmt_error(prepared_stmt, NULL);
    else
        dump_result_set(conn, prepared_stmt, "Contacts\n");

    mysql_stmt_close(prepared_stmt);
    return;
}

void send_message() {

    char receiver_username[45], text[100];

    print_color("    Username of the user that you want to send the message", "white", ' ', true, true, false, false);
    print_color("Username: ", "yellow", ' ', true, false, false, false);
    getInput(45, receiver_username, false);

    print_color("Message text: ", "yellow", ' ', false, false, false, false);
    getInput(100, text, false);

    MYSQL_STMT *prepared_stmt;
    MYSQL_BIND param[3];

    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 = receiver_username;
        param[1].buffer_length = strlen(receiver_username);

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

        if (!setup_prepared_stmt(&prepared_stmt, "call invioMessaggio (?, ?, ?)", conn))
                finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize ad statement", true);

        if (mysql_stmt_bind_param(prepared_stmt, param) != 0)
                finish_with_stmt_error(conn, prepared_stmt, "Unable to bind parameters to send message\n", true);

        if (mysql_stmt_execute(prepared_stmt) != 0)
                print_stmt_error(prepared_stmt, NULL);
        else
                print_color("    Successfully sent", "light blue", ' ', false, true, false, true);

        mysql_stmt_close(prepared_stmt);
        return;
}

void view_message() {

        char receiver_username[45], code_conversation_string[MAX_CODE_CONVERSATION];
        int code_conversation;

        bool request, is_null;
```

```
        is_null = 1;

        print_color("    Username of the user that you want to view the message", "white", ' ', true,
5   true, false, false);
        print_color("Username: ", "yellow", ' ', false, false, false, false);
        getInput(45, receiver_username, false);

        MYSQL_STMT *prepared_stmt;
10      MYSQL_BIND param[3];

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

        request = yesOrNo("Do you know the conversation code", 'y', 'n');
15
        if(request) {
                print_color("Code: ", "ligh cyan", ' ', false, false, false, false);
                getInput(MAX_CODE_CONVERSATION, code_conversation_string, false);
                code_conversation = atoi(code_conversation_string);
20      } else
                param[0].is_null = &is_null;


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

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

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

```
	if (!setup_prepared_stmt(&prepared_stmt, "call visualizzaMessaggio (?, ?, ?)", conn))
		finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize ad statement", true);

	if (mysql_stmt_bind_param(prepared_stmt, param) != 0)
		finish_with_stmt_error(conn, prepared_stmt, "Unable to bind parameters to view messages\n", true);

	if (mysql_stmt_execute(prepared_stmt) != 0)
		print_stmt_error(prepared_stmt, NULL);
	else
		dump_result_set(conn, prepared_stmt, "Messages\n");

	mysql_stmt_close(prepared_stmt);
	return;
}

void view_conversation_history() {
	MYSQL_STMT *prepared_stmt;
	MYSQL_BIND param[1];

	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 (!setup_prepared_stmt(&prepared_stmt, "call visualizzaStorico_UCC (?)", conn))
		finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize ad statement", true);

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

```
        finish_with_stmt_error(conn, prepared_stmt, "Unable to bind parameters to view conversation history\n", true);

    if (mysql_stmt_execute(prepared_stmt) != 0)
        print_stmt_error(prepared_stmt, NULL);
    else
        dump_result_set(conn, prepared_stmt, "Conversation history\n");

    mysql_stmt_close(prepared_stmt);
    return;
}

void follow_ad() {

    char ad_code_string[MAX_AD_CODE_LENGHT];
    int ad_code_follow;

    print_color("   Enter the ad code that you want to follow", "white", ' ', true, true, false, false);
    print_color("Ad code: ", "light cyan", ' ', false, false, false, false);
    getInput(MAX_AD_CODE_LENGHT, ad_code_string, false);
    ad_code_follow = atoi(ad_code_string);

    MYSQL_STMT *prepared_stmt;
    MYSQL_BIND param[2];

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

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

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

```
	if (!setup_prepared_stmt(&prepared_stmt, "call segui_Annuncio (?, ?)", conn))
		finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize ad statement", true);

	if (mysql_stmt_bind_param(prepared_stmt, param) != 0)
		finish_with_stmt_error(conn, prepared_stmt, "Unable to bind parameters to follow the ad\n", true);

	if (mysql_stmt_execute(prepared_stmt) != 0)
		print_stmt_error(prepared_stmt, NULL);
	else
		print_color("   Successfully followed", "light blue", ' ', false, true, false, true);

	mysql_stmt_close(prepared_stmt);
	return;
}

void view_ad_followed() {
	MYSQL_STMT *prepared_stmt;
	MYSQL_BIND param[1];

	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 (!setup_prepared_stmt(&prepared_stmt, "call visualizza_Annunci_Seguiti (?)", conn))
		finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize ad statement", true);

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

```
        finish_with_stmt_error(conn, prepared_stmt, "Unable to bind parameters to view ad followed\n", true);

    if (mysql_stmt_execute(prepared_stmt) != 0)
        print_stmt_error(prepared_stmt, NULL);
    else
        dump_result_set(conn, prepared_stmt, "Ad followed\n");

    mysql_stmt_close(prepared_stmt);
    return;
}

void view_report_ucc() {
    MYSQL_STMT *prepared_stmt;
    MYSQL_BIND param[2];

    char report_code_string[10];
    int report_code;

    bool request, is_null;

    is_null = 1;

    request = yesOrNo("Do you know the report code?", 'y', 'n');

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

    if(request) {
        print_color("Report code: ", "light cyan", ' ', false, false, false, false);
        getInput(10, report_code_string, false);
        report_code = atoi(report_code_string);
    } else
        param[0].is_null = &is_null;
```

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

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

        if (!setup_prepared_stmt(&prepared_stmt, "call visualizza_report(?, ?)", conn))
                finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize report statement", true);

        if (mysql_stmt_bind_param(prepared_stmt, param) != 0)
                finish_with_stmt_error(conn, prepared_stmt, "Unable to bind parameters to collect the report", true);

        if (mysql_stmt_execute(prepared_stmt) != 0)
                print_stmt_error(prepared_stmt, NULL);
        else
                dump_result_set(conn, prepared_stmt, "Reports\n");

        mysql_stmt_close(prepared_stmt);
        return;
}

void view_new_notifications_ucc() {
        MYSQL_STMT *prepared_stmt;
        MYSQL_BIND param[1];

        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 (!setup_prepared_stmt(&prepared_stmt, "call visualizza_notifiche (?)", conn))
		finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize ad statement", true);

	if (mysql_stmt_bind_param(prepared_stmt, param) != 0)
		finish_with_stmt_error(conn, prepared_stmt, "Unable to bind parameters to view new notifications\n", true);

	if (mysql_stmt_execute(prepared_stmt) != 0)
		print_stmt_error(prepared_stmt, NULL);
	else
		dump_result_set(conn, prepared_stmt, "New notifications\n");

	mysql_stmt_close(prepared_stmt);
	return;
}


int run_as_ucc(MYSQL *main_conn, struct configuration main_conf){
	conn = main_conn;
	conf = main_conf;
	ad_code = -1;
	int num_list = 22, chosen_num;
                                                // length of list
	char *list[] = {"1","2","3","4","5","6","7","8","9","10", "11", "12", "13", "14", "15", "16", "17", "18", "19", "20", "21", "22"};                                // list of choice
	char option;

	print_color("Welcome ", "cyan", ' ', true, false, true, false);
	print_color(conf.username, "cyan", ' ', false, true, true, false);
```

```
        if(!parse_config("Users/UCC.json", &conf)) {
                print_color("  Unable to load ucc configuration", "red", ' ', false, true, true, true);
                exit(EXIT_FAILURE);
        }

        if(mysql_change_user(conn, conf.db_username, conf.db_password, conf.database)) {
                print_color("  mysql_change_user() failed", "red", ' ', false, true, true, true);
                exit(EXIT_FAILURE);
        }

        while(1){

                option = ' ';
                chosen_num = -1;

                // Sleep process for 3 seconds so that the new instructions can be read by the user
                //poll(0, 0, 290);

                print_color("  What would do you want to do? ", "white", ' ', true, true, false, false);
                print_color(list[0], "light blue", ' ', true, false, false, false); print_color(") Insert a new Ad", "light cyan", ' ', false, false, false, false);
                print_color(list[1], "light blue", ' ', true, false, false, false); print_color(") View Ad", "orange", ' ', false, false, false, false);
                print_color(list[2], "light blue", ' ', true, false, false, false); print_color(") View categories", "light cyan", ' ', false, false, false, false);
                print_color(list[3], "light blue", ' ', true, false, false, false); print_color(") Remove Ad", "orange", ' ', false, false, false, false);
                print_color(list[4], "light blue", ' ', true, false, false, false); print_color(") Sell Ad", "light cyan", ' ', false, false, false, false);
                print_color(list[5], "light blue", ' ', true, false, false, false); print_color(") View Personal Information", "orange", ' ', false, false, false, false);
                print_color(list[6], "light blue", ' ', true, false, false, false); print_color(") Edit Personal Information", "light cyan", ' ', false, false, false, false);
```

```
print_color(list[7], "light blue", ' ', true, false, false, false); print_color(") View New Notifications", "orange", ' ', false, false, false, false);
print_color(list[8], "light blue", ' ', true, false, false, false); print_color(") Add or Remove a picture of your Ad", "light cyan", ' ', false, false, false, false);
print_color(list[9], "light blue", ' ', true, false, false, false); print_color(") View comments of an Ad", "orange", ' ', false, false, false, false);
print_color(list[10], "light blue", ' ', true, false, false, false); print_color(") Add or Remove a comment to an Ad", "light cyan", ' ', false, false, false, false);
print_color(list[11], "light blue", ' ', true, false, false, false); print_color(") View note of an Ad", "orange", ' ', false, false, false, false);
print_color(list[12], "light blue", ' ', true, false, false, false); print_color(") Add or Remove a note to an Ad", "light cyan", ' ', false, false, false, false);
print_color(list[13], "light blue", ' ', true, false, false, false); print_color(") View your contacts", "orange", ' ', false, false, false, false);
print_color(list[14], "light blue", ' ', true, false, false, false); print_color(") Add o remove a contact", "light cyan", ' ', false, false, false, false);
print_color(list[15], "light blue", ' ', true, false, false, false); print_color(") Send a message", "orange", ' ', false, false, false, false);
print_color(list[16], "light blue", ' ', true, false, false, false); print_color(") View your messages", "light cyan", ' ', false, false, false, false);
print_color(list[17], "light blue", ' ', true, false, false, false); print_color(") View your conversation history", "orange", ' ', false, false, false, false);
print_color(list[18], "light blue", ' ', true, false, false, false); print_color(") Follow an Ad", "light cyan", ' ', false, false, false, false);
print_color(list[19], "light blue", ' ', true, false, false, false); print_color(") View Ad followed", "orange", ' ', false, false, false, false);
print_color(list[20], "light blue", ' ', true, false, false, false); print_color(") View own report", "light cyan", ' ', false, false, false, false);
print_color(list[21], "light red", ' ', true, false, false, false); print_color(") QUIT", "light red", ' ', false, true, false, false);

multiChoice("Which do you choose?", list, num_list, &chosen_num, &option);

if(option == '#') {
```

```
            print_color("  Number doesn't exists", "red", ' ', false, true, false, true);
            continue;
        }

    5   if(chosen_num == num_list-1) {
            print_color("  Goodbye ", "orange", ' ', true, false, true, true);
            print_color(conf.username, "orange", ' ', false, true, true, true);
            printf("\n");
            break;
    10  }

        switch(chosen_num) {
            case 0:
                new_ad();
    15          break;
            case 1:
                view_ad("NULL", false);
                break;
            case 2:
    20          view_category_online();
                break;
            case 3:
                remove_ad();
                break;
    25      case 4:
                ad_sold();
                break;
            case 5:
                view_personal_information("", "", 0);
    30          break;
            case 6:
                edit_personal_information(NULL, NULL, NULL);
                break;
            case 7:
```

```
            view_new_notifications_ucc();
            break;
        case 8:
            edit_photo_ad();
            break;
        case 9:
            view_comment();
            break;
        case 10:
            insert_remove_comment();
            break;
        case 11:
            view_note();
            break;
        case 12:
            insert_remove_note();
            break;
        case 13:
            view_contact();
            break;
        case 14:
            insert_remove_contact();
            break;
        case 15:
            send_message();
            break;
        case 16:
            view_message();
            break;
        case 17:
            view_conversation_history();
            break;
        case 18:
            follow_ad();
```

```
                        break;
                case 19:
                        view_ad_followed();
                        break;
                case 20:
                        view_report_ucc();
                        break;
                default:
                        print_color("  Error to choose a number", "red", ' ', false, true, false, true);
                }
        }

        return 0;
}
```

- **uscc.c**

```
#include "defines.h"

struct configuration conf;
MYSQL *conn;

void view_ad_uscc() {
        MYSQL_STMT *prepared_stmt;
        MYSQL_BIND param[3];


        char ad_code_string[MAX_AD_CODE_LENGHT], ucc_username[45];
        int ad_code;

        bool request, is_null;

        is_null = 1;
```

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

	request = yesOrNo("Do you have any preference on ad?", 'y', 'n');

	if(request == true) {
		print_color("Ad code: ", "light cyan", ' ', false, false, false, false);
		getInput(MAX_AD_CODE_LENGHT, ad_code_string, false);
		ad_code = atoi(ad_code_string);
		param[1].is_null = &is_null;
		goto execution;
	}
	if(request == false){
		param[0].is_null = &is_null;
	}

	request = yesOrNo("Do you have any preference on user?", 'y', 'n');

	if(request == true) {
		print_color("Username: ", "yellow", ' ', false, false, false, false);
		getInput(45, ucc_username, false);
	}
	if(request == false){
		param[1].is_null = &is_null;
	}

	execution:

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

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

```
	param[2].is_null = &is_null;

	if (!setup_prepared_stmt(&prepared_stmt, "call visualizzaAnnuncio (?, ?, ?)", conn))
		finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize ad statement", true);

	if (mysql_stmt_bind_param(prepared_stmt, param) != 0)
		finish_with_stmt_error(conn, prepared_stmt, "Unable to bind parameters to view ads\n", true);

	if (mysql_stmt_execute(prepared_stmt) != 0)
		print_stmt_error(prepared_stmt, NULL);
	else
		dump_result_set(conn, prepared_stmt, "Ads list\n");		// dump the result set

	mysql_stmt_close(prepared_stmt);
	return;
}

void view_category_online_uscc() {
	MYSQL_STMT *prepared_stmt;

	if (!setup_prepared_stmt(&prepared_stmt, "call visualizzaCategoria()", conn))
		finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize view category statement\n", true);

	if (mysql_stmt_execute(prepared_stmt) != 0)
		print_stmt_error (prepared_stmt, "An error occurred while viewing categories.");

	dump_result_set(conn, prepared_stmt, "Online Category list\n");		// dump the result set
	mysql_stmt_next_result(prepared_stmt);
```

```
        mysql_stmt_close(prepared_stmt);
}

bool view_personal_information_uscc(char cf_owner[], char username_owner[], int check_owner) {
        MYSQL_STMT *prepared_stmt;

        // Check information for other functions
        if(cf_owner != NULL && username_owner != NULL && check_owner == 1){
                MYSQL_BIND param[3];

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

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

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

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

                if (!setup_prepared_stmt(&prepared_stmt, "call visualizzaInfoAnagrafiche (?, ?, ?)", conn))
                        finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize ad statement", true);

                if (mysql_stmt_bind_param(prepared_stmt, param) != 0)
                        finish_with_stmt_error(conn, prepared_stmt, "Unable to bind parameters to view information\n", true);

                goto execution;
```

```
        }


        // View USCC information
5       if(yesOrNo("Do you want to see your account information?", 'y', 'n')) {
                MYSQL_BIND param[1];
                memset(param, 0, sizeof(param));

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

                if (!setup_prepared_stmt(&prepared_stmt, "call visualizza_Info_Utente (?)", conn))
                        finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize ad
15  statement", true);

                if (mysql_stmt_bind_param(prepared_stmt, param) != 0)
                        finish_with_stmt_error(conn, prepared_stmt, "Unable to bind parameters to
    view information\n", true);
20
                goto execution;
        }


25      // View personal information
        MYSQL_BIND param[3];

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

30      char cf[MAX_CF_LENGHT];
        bool is_null;

        is_null = 1;
```

```
        print_color("Tax code: ", "yellow", ' ', false, false, false, false);
        getInput(MAX_CF_LENGHT, cf, false);
        cf[16] = '\0';

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

        param[1].is_null = &is_null;          // check_owner
        param[2].is_null = &is_null;          // username

        if (!setup_prepared_stmt(&prepared_stmt, "call visualizzaInfoAnagrafiche (?, ?, ?)", conn))
                finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize ad statement", true);


        if (mysql_stmt_bind_param(prepared_stmt, param) != 0)
                finish_with_stmt_error(conn, prepared_stmt, "Unable to bind parameters to view information\n", true);

        execution:
        if (mysql_stmt_execute(prepared_stmt) != 0) {
                print_stmt_error(prepared_stmt, NULL);
                mysql_stmt_close(prepared_stmt);
                return false;
        }

        if(check_owner != 1)
                dump_result_set(conn, prepared_stmt, "Personal Information\n");          // dump the result set


        mysql_stmt_close(prepared_stmt);
        return true;
```

```
}

void edit_personal_information_uscc(char cf_set_favorite[], char type_set_favorite[], char contact_set_favorites[]) {
    MYSQL_STMT *prepared_stmt;
    MYSQL_BIND param[8];

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

    bool is_null = 1;

    // Set contact as favorite
    if(cf_set_favorite != NULL && type_set_favorite != NULL && contact_set_favorites != NULL) {
        param[0].buffer_type = MYSQL_TYPE_VAR_STRING;
        param[0].buffer = cf_set_favorite;
        param[0].buffer_length = strlen(cf_set_favorite);

        param[1].is_null = &is_null;
        param[2].is_null = &is_null;
        param[3].is_null = &is_null;
        param[4].is_null = &is_null;
        param[5].is_null = &is_null;

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

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

        goto execution;
    }
```

```
char cf[MAX_CF_LENGHT], surname[20], name[20], residential_address[20], cap_string[5], billing_address[20], type_favorite_contact[20], favorite_contact[40];

char *list_choice_type_it[] = {"email", "cellulare", "social", "sms"};
char *list_choice_type_en[] = {"email", "mobile phone", "social", "sms"};
int lenght_choice_type = 4;

char choice[20];
int cap;

bool request;

print_color("Tax code: ", "yellow", ' ', false, false, false, false);
getInput(MAX_CF_LENGHT, cf, false);
cf[16] = '\0';

// Check if the user has this tax code
if(!view_personal_information_uscc(cf, conf.username, 1))
    return;

request = yesOrNo("Do you want to edit your surname?", 'y', 'n');

if(request == true) {
    print_color("Surname: ", "yellow", ' ', false, false, false, false);
    getInput(20, surname, false);
}
if(request == false){
    param[1].is_null = &is_null;
}

request = yesOrNo("Do you want to edit your name?", 'y', 'n');

if(request == true) {
```

```
            print_color("Name: ", "yellow", ' ', false, false, false, false);
            getInput(20, name, false);
        }
        if(request == false){
            param[2].is_null = &is_null;
        }

        request = yesOrNo("Do you want to edit your residential address?", 'y', 'n');

        if(request == true) {
            print_color("Residential address: ", "yellow", ' ', false, false, false, false);
            getInput(20, residential_address, false);
        }
        if(request == false){
            param[3].is_null = &is_null;
        }

        request = yesOrNo("Do you want to edit your CAP?", 'y', 'n');

        if(request == true) {
            print_color("CAP: ", "light cyan", ' ', false, false, false, false);
            getInput(5, cap_string, false);
            cap = atoi(cap_string);
        }
        if(request == false){
            param[4].is_null = &is_null;
        }

        request = yesOrNo("Do you want to edit your billing address?", 'y', 'n');

        if(request == true) {
            print_color("Billing address: ", "yellow", ' ', false, false, false, false);
            getInput(20, billing_address, false);
        }
```

```
        if(request == false){
                param[5].is_null = &is_null;
        }

        request = yesOrNo("Do you want to edit your favorite contact?", 'y', 'n');

        while(1) {
                if (request)
                        print_color("  Which do you choose to edit?", "orange", ' ', true, true, false,
false);
                else {
                        param[6].is_null = &is_null;
                        param[7].is_null = &is_null;
                        break;
                }

                for(int i=0; i<lenght_choice_type; i++) {
                        print_color(" - ", "cyan", ' ', false, false, false, false);
                        print_color(list_choice_type_en[i], "light blue", ' ', false, true, false, false);
                }

                // Take input
                getInput(20, choice, false);
                for(int i=0; i<lenght_choice_type; i++) {

                        if(strcmp(list_choice_type_en[i], choice) == 0) {
                                sprintf(type_favorite_contact, "%s", list_choice_type_it[i]);
                                type_favorite_contact[strlen(list_choice_type_it[i])] = '\0';

                                print_color("Contact: ", "yellow", ' ', false, false, false, false);
                                getInput(40, favorite_contact, false);

                                goto execution_editing_information;
                        }
```

```
        }
    }

    execution_editing_information:

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

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

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

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

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

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

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

    param[7].buffer_type = MYSQL_TYPE_VAR_STRING;
```

```
	param[7].buffer = favorite_contact;
	param[7].buffer_length = strlen(favorite_contact);

	execution:
	if (!setup_prepared_stmt(&prepared_stmt, "call modificaInfoAnagrafiche (?, ?, ?, ?, ?, ?, ?, ?)", conn))
		finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize ad statement", true);

	if (mysql_stmt_bind_param(prepared_stmt, param) != 0)
		finish_with_stmt_error(conn, prepared_stmt, "Unable to bind parameters to edit information\n", true);

	if (mysql_stmt_execute(prepared_stmt) != 0)
		print_stmt_error(prepared_stmt, NULL);
	else
		print_color("    Successfully edited!", "light blue", ' ', false, true, false, true);

	mysql_stmt_close(prepared_stmt);
	return;
}

void insert_comment_uscc() {
	MYSQL_STMT *prepared_stmt;
	MYSQL_BIND param[3];

	char ad_code_string[MAX_AD_CODE_LENGHT], text[45];
	int ad_code_comment;

	bool is_null;

	is_null = 1;

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

```
        print_color("Ad code: ", "light cyan", ' ', false, false, false, false);
        strcpy(ad_code_string, getInput(MAX_AD_CODE_LENGHT, ad_code_string, false));
        ad_code_comment = atoi(ad_code_string);

        print_color("Text of the comment: ", "yellow", ' ', false, false, false, false);
        getInput(45, text, false);

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

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

        param[2].is_null = &is_null;

        if (!setup_prepared_stmt(&prepared_stmt, "call modificaCommento (?, ?, ?)", conn))
                finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize ad statement", true);

        if (mysql_stmt_bind_param(prepared_stmt, param) != 0)
                finish_with_stmt_error(conn, prepared_stmt, "Unable to bind parameters to add or remove comment\n", true);

        if (mysql_stmt_execute(prepared_stmt) != 0)
                print_stmt_error(prepared_stmt, NULL);
        else
                print_color("   Successfully added!", "light blue", ' ', false, true, false, true);

        mysql_stmt_close(prepared_stmt);
        return;
}
```

```
void view_comment_uscc() {
	MYSQL_STMT *prepared_stmt;
	MYSQL_BIND param[1];

	char ad_code_string[MAX_AD_CODE_LENGHT];
	int ad_code_comment;

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

	print_color("Ad code: ", "light cyan", ' ', false, false, false, false);
	strcpy(ad_code_string, getInput(MAX_AD_CODE_LENGHT, ad_code_string, false));
	ad_code_comment = atoi(ad_code_string);

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

	if (!setup_prepared_stmt(&prepared_stmt, "call visualizzaCommento (?)", conn))
		finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize ad statement", true);

	if (mysql_stmt_bind_param(prepared_stmt, param) != 0)
		finish_with_stmt_error(conn, prepared_stmt, "Unable to bind parameters to view comments\n", true);

	if (mysql_stmt_execute(prepared_stmt) != 0)
		print_stmt_error(prepared_stmt, NULL);
	else
		dump_result_set(conn, prepared_stmt, "Comments\n");

	mysql_stmt_close(prepared_stmt);
	return;
```

```
}

void view_note_uscc() {
	MYSQL_STMT *prepared_stmt;
	MYSQL_BIND param[1];

	char ad_code_string[MAX_AD_CODE_LENGHT];
	int ad_code_note;

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

	print_color("Ad code: ", "light cyan", ' ', false, false, false, false);
	strcpy(ad_code_string, getInput(MAX_AD_CODE_LENGHT, ad_code_string, false));
	ad_code_note = atoi(ad_code_string);

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

	if (!setup_prepared_stmt(&prepared_stmt, "call visualizzaNota (?)", conn))
		finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize ad statement", true);

	if (mysql_stmt_bind_param(prepared_stmt, param) != 0)
		finish_with_stmt_error(conn, prepared_stmt, "Unable to bind parameters to view note\n", true);

	if (mysql_stmt_execute(prepared_stmt) != 0)
		print_stmt_error(prepared_stmt, NULL);
	else
		dump_result_set(conn, prepared_stmt, "Note\n");

	mysql_stmt_close(prepared_stmt);
	return;
```

```
}

void insert_remove_contact_uscc() {

    char type[20], contact[40], cf[MAX_CF_LENGHT];
    int remove = 1;

    char *list_choice_type_it[] = {"email", "cellulare", "social", "sms"}, choice[20];
    char *list_choice_type_en[] = {"email", "mobile phone", "social", "sms"};
    int lenght_choice_type = 4, i;

    bool request, request_2, is_null;

    is_null = 1;

    print_color("Tax code: ", "yellow", ' ', false, false, false, false);
    getInput(MAX_CF_LENGHT, cf, false);

    // Check if the user has this tax code
    if(!view_personal_information_uscc(cf, conf.username, 1))
        return;

    request = yesOrNo("Do you want to add a contact?", 'y', 'n');

    while(1) {
        if (request)
            print_color("    Which do you choose to add?\n", "white", ' ', true, true, false,
false);
        else
            print_color(    "Which do you choose to remove?\n", "white", ' ', true, true,
false, false);

        for(i=0; i<lenght_choice_type; i++) {
            print_color(" - ", "cyan", ' ', false, false, false, false);
```

```
                print_color(list_choice_type_en[i], "light blue", ' ', false, true, false, false);
        }

        // Take input
        getInput(20, choice, false);
        for(i=0; i<lenght_choice_type; i++) {

                if(strcmp(list_choice_type_en[i], choice) == 0) {
                        sprintf(type, "%s", list_choice_type_it[i]);
                        type[strlen(list_choice_type_it[i])] = '\0';

                        print_color("Contact: ", "yellow", ' ', false, false, false, false);
                        getInput(40, contact, false);
                        goto execution;
                }
        }
}

execution:

if(request) {
        request_2 = yesOrNo("Do you want to set it as a favorite?", 'y', 'n');

        // If the user sends yes then it set as favorite
        if (request_2) {
                edit_personal_information_uscc(cf, type, contact);
                return;
        }
}

MYSQL_STMT *prepared_stmt;
MYSQL_BIND param[4];

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

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

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

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

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

        // If the user wants to remove a contact
        if(request)
                param[3].is_null= &is_null;

        if (!setup_prepared_stmt(&prepared_stmt, "call modifica_RecapitoNonPreferito (?, ?, ?, ?)", conn))
                finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize ad statement", true);

        if (mysql_stmt_bind_param(prepared_stmt, param) != 0)
                finish_with_stmt_error(conn, prepared_stmt, "Unable to bind parameters to add or remove note\n", true);

        if (mysql_stmt_execute(prepared_stmt) != 0) {
                print_stmt_error(prepared_stmt, NULL);
                mysql_stmt_close(prepared_stmt);
                return;
```

```
        }

        if(request)
                print_color("   Successfully added!", "light blue", ' ', false, true, false, true);
        else
                print_color("   Successfully removed!", "light red", ' ', false, true, false, true);

        mysql_stmt_close(prepared_stmt);
        return;
}

void view_contact_uscc() {
        MYSQL_STMT *prepared_stmt;
        MYSQL_BIND param[2];

        char cf[MAX_CF_LENGHT];
        int favorite = 1;

        bool request, is_null;

        is_null = 1;

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

        print_color("Tax code: ", "yellow", ' ', false, false, false, false);
        getInput(MAX_CF_LENGHT, cf, false);

        request = yesOrNo("Do you want to see favorite contact?", 'y', 'n');

        // Set the var to null
        if(!request)
                param[1].is_null = &is_null;

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

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

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

        if (!setup_prepared_stmt(&prepared_stmt, "call visualizza_contatti (?, ?)", conn))
                finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize ad statement",
true);

        if (mysql_stmt_bind_param(prepared_stmt, param) != 0)
                finish_with_stmt_error(conn, prepared_stmt, "Unable to bind parameters to view
contacts\n", true);

        if (mysql_stmt_execute(prepared_stmt) != 0)
                print_stmt_error(prepared_stmt, NULL);
        else
                dump_result_set(conn, prepared_stmt, "Contacts\n");

        mysql_stmt_close(prepared_stmt);
        return;
}

void send_message_uscc() {

        char receiver_username[45], text[100];

        print_color("   Username of the user that you want to send the message", "white", ' ', true,
true, false, false);
        print_color("Username: ", "yellow", ' ', false, false, false, false);
        getInput(45, receiver_username, false);

        print_color("Message text: ", "yellow", ' ', false, false, false, false);
```

```
        getInput(100, text, false);

        MYSQL_STMT *prepared_stmt;
        MYSQL_BIND param[3];

        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 = receiver_username;
        param[1].buffer_length = strlen(receiver_username);

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

        if (!setup_prepared_stmt(&prepared_stmt, "call invioMessaggio (?, ?, ?)", conn))
                finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize ad statement", true);

        if (mysql_stmt_bind_param(prepared_stmt, param) != 0)
                finish_with_stmt_error(conn, prepared_stmt, "Unable to bind parameters to send message\n", true);

        if (mysql_stmt_execute(prepared_stmt) != 0)
                print_stmt_error(prepared_stmt, NULL);
        else
                print_color("    Successfully sent", "light blue", ' ', false, true, false, true);

        mysql_stmt_close(prepared_stmt);
        return;
```

```
}

void view_message_uscc() {

    char receiver_username[45], code_conversation_string[MAX_CODE_CONVERSATION];
    int code_conversation;

    bool request, is_null;

    is_null = 1;

    print_color("    Username of the user that you want to view the message", "white", ' ', true, true, false, false);
    print_color("Username: ", "yellow", ' ', true, false, false, false);
    getInput(45, receiver_username, false);

    MYSQL_STMT *prepared_stmt;
    MYSQL_BIND param[3];

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

    request = yesOrNo("Do you know the conversation code", 'y', 'n');

    if(request) {
        print_color("Code: ", "ligh cyan", ' ', false, false, false, false);
        getInput(MAX_CODE_CONVERSATION, code_conversation_string, false);
        code_conversation = atoi(code_conversation_string);
    } else
        param[0].is_null = &is_null;


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

```
	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 = receiver_username;
	param[2].buffer_length = strlen(receiver_username);

	if (!setup_prepared_stmt(&prepared_stmt, "call visualizzaMessaggio (?, ?, ?)", conn))
		finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize ad statement", true);

	if (mysql_stmt_bind_param(prepared_stmt, param) != 0)
		finish_with_stmt_error(conn, prepared_stmt, "Unable to bind parameters to view messages\n", true);

	if (mysql_stmt_execute(prepared_stmt) != 0)
		print_stmt_error(prepared_stmt, NULL);
	else
		dump_result_set(conn, prepared_stmt, "Messages\n");

	mysql_stmt_close(prepared_stmt);
	return;
}

void view_conversation_history_uscc() {
	MYSQL_STMT *prepared_stmt;
	MYSQL_BIND param[1];

	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 (!setup_prepared_stmt(&prepared_stmt, "call visualizzaStorico_USCC (?)", conn))
        finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize ad statement",
5   true);

    if (mysql_stmt_bind_param(prepared_stmt, param) != 0)
        finish_with_stmt_error(conn, prepared_stmt, "Unable to bind parameters to view
    conversation history\n", true);
10
    if (mysql_stmt_execute(prepared_stmt) != 0)
        print_stmt_error(prepared_stmt, NULL);
    else
        dump_result_set(conn, prepared_stmt, "Conversation history\n");
15
    mysql_stmt_close(prepared_stmt);
    return;
}

20  void follow_ad_uscc() {

    char ad_code_string[MAX_AD_CODE_LENGHT];
    int ad_code_follow;

25  print_color("   Enter the ad code that you want to follow", "white", ' ', true, true, false, false);
    print_color("Ad code: ", "light cyan", ' ', false, false, false, false);
    getInput(MAX_AD_CODE_LENGHT, ad_code_string, false);
    ad_code_follow = atoi(ad_code_string);

30  MYSQL_STMT *prepared_stmt;
    MYSQL_BIND param[2];

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

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

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

        if (!setup_prepared_stmt(&prepared_stmt, "call segui_Annuncio (?, ?)", conn))
                finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize ad statement", true);

        if (mysql_stmt_bind_param(prepared_stmt, param) != 0)
                finish_with_stmt_error(conn, prepared_stmt, "Unable to bind parameters to follow the ad\n", true);

        if (mysql_stmt_execute(prepared_stmt) != 0)
                print_stmt_error(prepared_stmt, NULL);
        else
                print_color("    Successfully followed", "light blue", ' ', false, true, false, true);

        mysql_stmt_close(prepared_stmt);
        return;
}

void view_ad_followed_uscc() {
        MYSQL_STMT *prepared_stmt;
        MYSQL_BIND param[1];

        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 (!setup_prepared_stmt(&prepared_stmt, "call visualizza_Annunci_Seguiti (?)", conn))
                finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize ad statement", true);

        if (mysql_stmt_bind_param(prepared_stmt, param) != 0)
                finish_with_stmt_error(conn, prepared_stmt, "Unable to bind parameters to view ad followed\n", true);

        if (mysql_stmt_execute(prepared_stmt) != 0)
                print_stmt_error(prepared_stmt, NULL);
        else
                dump_result_set(conn, prepared_stmt, "Ad followed\n");

        mysql_stmt_close(prepared_stmt);
        return;
}

void view_new_notifications_uscc() {
        MYSQL_STMT *prepared_stmt;
        MYSQL_BIND param[1];

        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 (!setup_prepared_stmt(&prepared_stmt, "call visualizza_notifiche (?)", conn))
                finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize ad statement", true);

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

```
		finish_with_stmt_error(conn, prepared_stmt, "Unable to bind parameters to view new notifications\n", true);

	if (mysql_stmt_execute(prepared_stmt) != 0)
		print_stmt_error(prepared_stmt, NULL);
	else
		dump_result_set(conn, prepared_stmt, "New notifications\n");

	mysql_stmt_close(prepared_stmt);
	return;
}


int run_as_uscc(MYSQL *main_conn, struct configuration main_conf){

	conn = main_conn;
	conf = main_conf;

	int num_list = 16, chosen_num;
                                                                    // length of list
	char *list[] = {"1","2","3","4","5","6","7","8","9","10", "11", "12", "13", "14", "15", "16"};
                                // list of choice
	char option;

	print_color("Welcome ", "cyan", ' ', true, false, true, false);
	print_color(conf.username, "cyan", ' ', false, true, true, false);

	if(!parse_config("Users/USCC.json", &conf)) {
		print_color("  Unable to load uscc configuration", "red", ' ', false, true, true, true);
		exit(EXIT_FAILURE);
	}

	if(mysql_change_user(conn, conf.db_username, conf.db_password, conf.database)) {
```

```
            print_color("  mysql_change_user() failed", "red", ' ', false, true, true, true);
            exit(EXIT_FAILURE);
        }

5       while(1){

            option = ' ';
            chosen_num = -1;

10          // Sleep process for 3 seconds so that the new instructions can be read by the user
            //poll(0, 0, 290);

            print_color("  What would do you want to do? ", "white", ' ', true, true, false, false);
            print_color(list[0], "light blue", ' ', true, false, false, false); print_color(") View ad",
15  "orange", ' ', false, false, false, false);
            print_color(list[1], "light blue", ' ', true, false, false, false); print_color(") View
categories", "light cyan", ' ', false, false, false, false);
            print_color(list[2], "light blue", ' ', true, false, false, false); print_color(") View
Personal Information", "orange", ' ', false, false, false, false);
20          print_color(list[3], "light blue", ' ', true, false, false, false); print_color(") Edit Personal
Information", "light cyan", ' ', false, false, false, false);
            print_color(list[4], "light blue", ' ', true, false, false, false); print_color(") View
comments of an ad", "orange", ' ', false, false, false, false);
            print_color(list[5], "light blue", ' ', true, false, false, false); print_color(") Add a
25  comment to an ad", "light cyan", ' ', false, false, false, false);
            print_color(list[6], "light blue", ' ', true, false, false, false); print_color(") View note of
an ad", "orange", ' ', false, false, false, false);
            print_color(list[7], "light blue", ' ', true, false, false, false); print_color(") View your
contacts", "light cyan", ' ', false, false, false, false);
30          print_color(list[8], "light blue", ' ', true, false, false, false); print_color(") Add o
remove a contact", "orange", ' ', false, false, false, false);
            print_color(list[9], "light blue", ' ', true, false, false, false); print_color(") Send a
message", "light cyan", ' ', false, false, false, false);
```

```
        print_color(list[10], "light blue", ' ', true, false, false, false); print_color(") View your messages", "orange", ' ', false, false, false, false);
        print_color(list[11], "light blue", ' ', true, false, false, false); print_color(") View your conversation history", "light cyan", ' ', false, false, false, false);
        print_color(list[12], "light blue", ' ', true, false, false, false); print_color(") View new Notifications", "orange", ' ', false, false, false, false);
        print_color(list[13], "light blue", ' ', true, false, false, false); print_color(") Follow an ad", "light cyan", ' ', false, false, false, false);
        print_color(list[14], "light blue", ' ', true, false, false, false); print_color(") View ad followed", "orange", ' ', false, false, false, false);
        print_color(list[15], "light red", ' ', true, false, false, false); print_color(") QUIT", "light red", ' ', false, true, false, false);

        multiChoice("Which do you choose?", list, num_list, &chosen_num, &option);

        if(option == '#') {
                print_color("  Number doesn't exists", "red", ' ', false, true, false, true);
                continue;
        }

        if(chosen_num == num_list-1) {
                print_color("  Goodbye ", "orange", ' ', true, false, true, true);
                print_color(conf.username, "orange", ' ', false, true, true, true);
                printf("\n");
                break;
        }

        switch(chosen_num) {
                case 0:
                        view_ad_uscc();
                        break;
                case 1:
                        view_category_online_uscc();
                        break;
```

```
            case 2:
                view_personal_information_uscc("", "", 0);
                break;
            case 3:
                edit_personal_information_uscc(NULL, NULL, NULL);
                break;
            case 4:
                view_comment_uscc();
                break;
            case 5:
                insert_comment_uscc();
                break;
            case 6:
                view_note_uscc();
                break;
            case 7:
                view_contact_uscc();
                break;
            case 8:
                insert_remove_contact_uscc();
                break;
            case 9:
                send_message_uscc();
                break;
            case 10:
                view_message_uscc();
                break;
            case 11:
                view_conversation_history_uscc();
                break;
            case 12:
                view_new_notifications_uscc();
                break;
            case 13:
```

```
                follow_ad_uscc();
                break;
            case 14:
                view_ad_followed_uscc();
                break;
            default:
                print_color("  Error to choose a number", "red", ' ', false, true, false, true);
        }
    }

    return 0;
}
```

- **administrator.c**

```
#include "defines.h"

struct configuration conf;
MYSQL *conn;


void new_category() {
    MYSQL_STMT *prepared_stmt;
    MYSQL_BIND param[1];

    char name[20];                  // name of category

    print_color("Name of new category: ", "yellow", ' ', false, false, false, false);
    getInput(20, name, false);

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

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

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


	if (!setup_prepared_stmt(&prepared_stmt, "call inserimentoNuovaCategoria(?)", conn))
		finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize new category statement\n", true);

	if (mysql_stmt_bind_param(prepared_stmt, param) != 0)
		finish_with_stmt_error(conn, prepared_stmt, "Unable to bind parameters for a new category\n", true);

	if (mysql_stmt_execute(prepared_stmt) != 0)
		print_stmt_error(prepared_stmt, NULL);
	else
		print_color("    Successfully added!", "light blue", ' ', false, true, false, true);

	mysql_stmt_close(prepared_stmt);
	return;
}

void view_category() {
	MYSQL_STMT *prepared_stmt;

	if (!setup_prepared_stmt(&prepared_stmt, "call visualizzaCategoria()", conn))
		finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize view category statement\n", true);

	if (mysql_stmt_execute(prepared_stmt) != 0)
		print_stmt_error (prepared_stmt, "An error occurred while viewing categories.");

	dump_result_set(conn, prepared_stmt, "Online Category list\n");		// dump the result set
	mysql_stmt_next_result(prepared_stmt);
```

```
        mysql_stmt_close(prepared_stmt);
        return;
}

void view_ad_administrator() {
        MYSQL_STMT *prepared_stmt;
        MYSQL_BIND param[3];

        char                                    ad_code_string[MAX_AD_CODE_LENGHT],
ucc_username[MAX_LENGHT_USERNAME];
        int ad_code;

        bool request, is_null;

        is_null = 1;

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

        request = yesOrNo("Do you have any preference on ad?", 'y', 'n');

        if(request == true) {
                print_color("Ad code: ", "light cyan", ' ', false, false, false, false);
                getInput(MAX_AD_CODE_LENGHT, ad_code_string, false);
                ad_code = atoi(ad_code_string);
                param[1].is_null = &is_null;
                goto execution;
        }
        if(request == false){
                param[0].is_null = &is_null;
        }

        request = yesOrNo("Do you have any preference on user?", 'y', 'n');

        if(request == true) {
```

```
            print_color("Username: ", "yellow", ' ', false, false, false, false);
            getInput(MAX_LENGHT_USERNAME, ucc_username, false);
        }
        if(request == false){
            param[1].is_null = &is_null;
        }

        execution:

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

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

        param[2].is_null = &is_null;

        if (!setup_prepared_stmt(&prepared_stmt, "call visualizzaAnnuncio (?, ?, ?)", conn))
            finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize ad statement", true);

        if (mysql_stmt_bind_param(prepared_stmt, param) != 0)
            finish_with_stmt_error(conn, prepared_stmt, "Unable to bind parameters to view ads\n", true);

        if (mysql_stmt_execute(prepared_stmt) != 0)
            print_stmt_error(prepared_stmt, NULL);
        else
            dump_result_set(conn, prepared_stmt, "Ads list\n");          // dump the result set

        mysql_stmt_close(prepared_stmt);
        return;
```

```
}

void generate_report() {
	MYSQL_STMT *prepared_stmt;
	MYSQL_BIND param[2];

	char ucc_username[MAX_LENGHT_USERNAME];

	print_color("User's username: ", "yellow", ' ', false, false, false, false);
	getInput(MAX_LENGHT_USERNAME, ucc_username, false);

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

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

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

	if (!setup_prepared_stmt(&prepared_stmt, "call genera_report(?, ?)", conn))
		finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize report statement\n",
true);

	if (mysql_stmt_bind_param(prepared_stmt, param) != 0)
		finish_with_stmt_error(conn, prepared_stmt, "Unable to bind parameters for a new
report\n", true);

	if (mysql_stmt_execute(prepared_stmt) != 0)
		print_stmt_error(prepared_stmt, NULL);
	else
		print_color("   Successfully generated!", "light blue", ' ', false, true, false, true);
```

```
        mysql_stmt_close(prepared_stmt);
}


void collect_report() {
        MYSQL_STMT *prepared_stmt;
        MYSQL_BIND param[3];

        char ucc_username[MAX_LENGHT_USERNAME], report_code_string[10];
        int report_code;

        print_color("Username: ", "yellow", ' ', false, false, false, false);
        getInput(MAX_LENGHT_USERNAME, ucc_username, false);

        print_color("Report code: ", "light cyan", ' ', false, false, false, false);
        getInput(10, report_code_string, false);
        report_code = atoi(report_code_string);

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

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

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

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

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

```
        finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize report statement", true);

    if (mysql_stmt_bind_param(prepared_stmt, param) != 0)
        finish_with_stmt_error(conn, prepared_stmt, "Unable to bind parameters to collect the report", true);

    if (mysql_stmt_execute(prepared_stmt) != 0)
        print_stmt_error(prepared_stmt, NULL);
    else
        print_color("   Successfully collected!", "light blue", ' ', false, true, false, true);

    mysql_stmt_close(prepared_stmt);
    return;
}

void view_report() {
    MYSQL_STMT *prepared_stmt;
    MYSQL_BIND param[2];

    char ucc_username[MAX_LENGHT_USERNAME], report_code_string[10];
    int report_code;

    bool request, is_null;

    is_null = 1;

    print_color("Username: ", "yellow", ' ', false, false, false, false);
    getInput(MAX_LENGHT_USERNAME, ucc_username, false);

    request = yesOrNo("Do you know the report code?", 'y', 'n');

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

```
        if(request) {
                print_color("Report code: ", "light cyan", ' ', false, false, false, false);
                getInput(10, report_code_string, false);
                report_code = atoi(report_code_string);
        } else
                param[0].is_null = &is_null;


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

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

        if (!setup_prepared_stmt(&prepared_stmt, "call visualizza_report(?, ?)", conn))
                finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize report statement", true);

        if (mysql_stmt_bind_param(prepared_stmt, param) != 0)
                finish_with_stmt_error(conn, prepared_stmt, "Unable to bind parameters to collect the report", true);

        if (mysql_stmt_execute(prepared_stmt) != 0)
                print_stmt_error(prepared_stmt, NULL);
        else
                dump_result_set(conn, prepared_stmt, "Reports\n");

        mysql_stmt_close(prepared_stmt);
        return;
}

int run_as_administrator(MYSQL *main_conn, struct configuration main_conf){
```

```
        conn = main_conn;
        conf = main_conf;

        int num_list = 7, chosen_num;
                                                        // length of list
        char *list[] = {"1","2","3","4","5","6","7"};
                                        // list of choice
        char option;

        print_color("Welcome ", "cyan", ' ', true, false, true, false);
        print_color(conf.username, "cyan", ' ', false, true, true, false);

        if(!parse_config("Users/Administrator.json", &conf)) {
                print_color("  Unable to load administrator configuration", "red", ' ', false, true, true,
true);
                exit(EXIT_FAILURE);
        }

        if(mysql_change_user(conn, conf.db_username, conf.db_password, conf.database)) {
                print_color("  mysql_change_user() failed", "red", ' ', false, true, true, true);
                exit(EXIT_FAILURE);
        }

        while(1){

                option = ' ';
                chosen_num = -1;

                // Sleep process for 3 seconds so that the new instructions can be read by the user
                //poll(0, 0, 290);

                print_color("  What would do you want to do? ", "white", ' ', true, true, false, false);
```

```
        print_color(list[0], "light blue", ' ', true, false, false, false); print_color('') Insert a new category", "orange", ' ', false, false, false, false);
        print_color(list[1], "light blue", ' ', true, false, false, false); print_color('') View category names online", "light cyan", ' ', false, false, false, false);
        print_color(list[2], "light blue", ' ', true, false, false, false); print_color('') View ad", "orange", ' ', false, false, false, false);
        print_color(list[3], "light blue", ' ', true, false, false, false); print_color('') Generate a report", "light cyan", ' ', false, false, false, false);
        print_color(list[4], "light blue", ' ', true, false, false, false); print_color('') Collect reports", "orange", ' ', false, false, false, false);
        print_color(list[5], "light blue", ' ', true, false, false, false); print_color('') View reports", "light cyan", ' ', false, false, false, false);
        print_color(list[6], "light red", ' ', true, false, false, false); print_color('') QUIT", "light red", ' ', false, true, false, false);

        multiChoice("Which do you choose?", list, num_list, &chosen_num, &option);

        if(option == '#') {
                print_color("  Number doesn't exists", "red", ' ', false, true, false, true);
                continue;
        }

        if(chosen_num == num_list-1) {
                print_color("  Goodbye ", "orange", ' ', true, false, true, true);
                print_color(conf.username, "orange", ' ', false, true, true, true);
                printf("\n");
                break;
        }

        switch(chosen_num) {
                case 0:
                        new_category();
                        break;
                case 1:
```

```
                    view_category();
                    break;
                case 2:
                    view_ad_administrator();
                    break;
                case 3:
                    generate_report();
                    break;
                case 4:
                    collect_report();
                    break;
                case 5:
                    view_report();
                    break;
                default:
                    print_color("  Error to choose a number", "red", ' ', false, true, false, true);
            }
        }

        return 0;
}
```

- **defines.h**

```
#pragma once

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

#define MAX_AD_CODE_LENGHT 10
#define MAX_COMMENT_ID_LENGHT 10
```

```
#define MAX_NOTE_ID_LENGHT 10
#define MAX_CF_LENGHT 17
#define MAX_CODE_CONVERSATION 10
#define MAX_LENGHT_USERNAME 45

struct configuration {
        char *host;
        char *db_username;
        char *db_password;
        unsigned int port;
        char *database;

        char username[45];
        char password[45];
};

extern struct configuration conf;

extern int parse_config(char *path, struct configuration *conf);
extern char *getInput(unsigned int lung, char *stringa, bool hide);
extern char *getInputScanf(char *domanda, char *stringa, int length_max);
extern bool yesOrNo(char *domanda, char yes, char no);
extern void print_color(char *stringa, char *colore_scelto, char c, bool first_space, bool last_space,
bool bold, bool blink);
extern void multiChoice(char *domanda, char *choices[], int num, int *chosen_num, char *option);
extern void print_error (MYSQL *conn, char *message);
extern void print_stmt_error (MYSQL_STMT *stmt, char *message);
extern void finish_with_error(MYSQL *conn, char *message);
extern void finish_with_stmt_error(MYSQL *conn, MYSQL_STMT *stmt, char *message, bool
close_stmt);
extern bool setup_prepared_stmt(MYSQL_STMT **stmt, char *statement, MYSQL *conn);
extern void dump_result_set(MYSQL *conn, MYSQL_STMT *stmt, char *title);
extern int run_as_administrator(MYSQL *conn, struct configuration conf);
extern int run_as_ucc(MYSQL *conn, struct configuration conf);
```

```
extern int run_as_uscc(MYSQL *conn, struct configuration conf);
```

- **inout.c**

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

#include "defines.h"

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


void handler_ign(){
        signal(SIGALRM, handler);
        signal(SIGINT, handler);
        signal(SIGHUP, handler);
        signal(SIGQUIT, handler);
        signal(SIGTERM, handler);
        signal(SIGTSTP, handler);
        signal(SIGTTIN, handler);
        signal(SIGTTOU, handler);
}
```

```
void handler_dfl(){
        signal(SIGALRM, SIG_DFL);
        signal(SIGINT, SIG_DFL);
        signal(SIGHUP, SIG_DFL);
        signal(SIGQUIT, SIG_DFL);
        signal(SIGTERM, SIG_DFL);
        signal(SIGTSTP, SIG_DFL);
        signal(SIGTTIN, SIG_DFL);
        signal(SIGTTOU, SIG_DFL);
}


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

        struct termios term, oterm;

        handler_ign(); // Ignora eventuali segnali

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

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

```
	}

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

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

	// 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');
	}

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

	if(hide) {
		//Va a capo dopo l'input
```

```
                (void) write(fileno(stdout), "\n", 1);

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

                handler_dfl();  // Ripristina la gestione dei segnali

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

        return stringa;
}

// Scanf alternativo
char *getInputScanf(char *domanda, char *stringa, int length_max) {
        int length, i=0;
        char c;

        printf("%s", domanda);

        handler_ign();          // Ignora eventuali segnali

        length = 0;
        stringa = (char *) malloc(length_max * sizeof(char *));

        while(1) {
                c = getchar();
                stringa[i] = c;
                if(c == '\n') break;
                length++;
                i++;
        }
```

```
        stringa[length] = '\0';

        if(length > length_max)
                stringa[length_max-1] = '\0';

        handler_dfl();          // Reset di default dei segnali

        // Rilancio dei segnali pendenti
        if(signo)
                (void) raise(signo);

        return stringa;
}

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

void print_color(char *colorless, char *colore_scelto, char c, bool first_space, bool last_space, bool bold, bool blink) {
        char *stringa;
        char *list_color[] = {"orange", "blue", "light blue", "red", "light red", "white", "purple", "cyan", "light cyan", "yellow", "pink"};
        int i, length_list_color;

        // Controllo dei parametri
        if(strlen(colorless) == 0 && c == ' ') {
                print_error(NULL, "Error Parameters");
                return;
        }

        // Il colore deve essere minuscolo
```

```
char *colore = (char *) malloc(strlen(colore_scelto) * sizeof(char *));
for(i=0; colore[i]; i++){
        colore[i] = tolower(colore_scelto[i]);
}
colore[i] = '\0';

// Controllo che la variabile passata sia una stringa o un char
if(strcmp(colorless, "") == 0) {
        stringa = (char *) malloc(2);
        stringa[0] = c;
        stringa[1] = '\0';
} else {
        stringa = (char *) malloc(strlen(colorless)*sizeof(char *));
        strcpy(stringa, colorless);
        stringa[strlen(stringa)] = '\0';
}

// Cerco il colore
length_list_color = 11; i = 0;
while(1) {
        if (i == length_list_color) {break;}
        if (strcmp(colore_scelto, list_color[i]) == 0) {break;}
        i++;
}
free(colore);

// Stampa spazio
if(first_space)
        printf("\n");

// Stampa blink
if(blink)
        printf("\e[1m\e[5m");
```

```
        // Stampa in grassetto
        if(bold)
                printf("\e[1m");


        // Stampa della stringa
        switch(i) {
        case 0:
                printf("\033[40m\033[1;32m");
                printf("%s", stringa);
                break;
        case 1:
                printf("\033[40m\033[34m");
                printf("%s", stringa);
                break;
        case 2:
                printf("\033[40m\033[1;34m");
                printf("%s", stringa);
                break;
        case 3:
                printf("\033[40m\033[31m");
                printf("%s", stringa);
                break;
        case 4:
                printf("\033[40m\033[1;31m");
                printf("%s", stringa);
                break;
        case 5:
                printf("\033[40m\033[1;37m");
                printf("%s", stringa);
                break;
        case 6:
                printf("\033[40m\033[35m");
                printf("%s", stringa);
```

```
            break;
        case 7:
            printf("\033[40m\033[36m");
            printf("%s", stringa);
            break;
        case 8:
            printf("\033[40m\033[1;36m");
            printf("%s", stringa);
            break;
        case 9:
            printf("\033[40m\033[1;33m");
            printf("%s", stringa);
            break;
        case 10:
            printf("\033[40m\033[1;35m");
            printf("%s", stringa);
            break;
        default:
            printf("%s", stringa);
            break;
        }

        // Delimiter color
        if(i<length_list_color)
            printf("\033[0m");

        // Fine stampa in grassetto
        if(bold)
            printf("\e[22m");

        // Fine stampa blink
        if(blink)
            printf("\e[25m\e[22m");
```

```
        // Stampa spazio
        if(last_space)
                printf("\n");

        free(stringa);   // Deallocazione spazio nell'heap
        return;
}

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

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

        // Richiesta della risposta
        while(true) {
                print_color(domanda, "white", ' ', false, false, true, false);    // Mostra la domanda
                printf(" [");
                print_color("", "light blue", yes, false, false, false, false);
                printf("/");
                print_color("", "light red", no, false, false, false, false);
                printf("]: ");

                //printf("\e[1m%s\e[22m   [\033[40m\033[1;34m%c\033[0m/\033[40m\033[1;31m%c\
033[0m]: ", domanda, yes, no);          // Mostra la domanda

                char c[2];
                getInput(2, c, false);

                // Controlla quale risposta è stata data
                if(c[0] == yes || c[0] == toupper(yes))
                        return true;
                else if(c[0] == no || c[0] == toupper(no))
                        return false;
```

```
        }
}

void multiChoice(char *domanda, char *choices[], int num, int *chosen_num, char *option)
{
        int i = 0, j = 0, lenght = 0;

        while(i<num)
                lenght += (int)strlen(choices[i++]);

        char *possib = (char *) malloc(lenght * num * sizeof(char *));        // Genera la stringa delle possibilità

        // Copia tutti i valori in un unica stringa
        for(i = 0; i < num; i++) {
                for(int z=0; z<(int)strlen(choices[i]); z++) {
                        possib[j++] = choices[i][z];
                }
                possib[j++] = '/';
        }
        possib[j-1] = '\0';         // Per eliminare l'ultima '/'

        // Chiede la risposta
        print_color(domanda, "white", ' ', true, false, true, false);      // Mostra la domanda
        printf(" [");
        print_color(possib, "light blue", ' ', false, false, false, false);
        printf("]: ");

        char c[3];
        sprintf(c, "%s", getInput(3, c, false));

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

```
                        *chosen_num = i;      // Imposta l'indice all'interno della lista tramite dereferenziamento di puntatore
                        return;
                }
        }

        *option = '#';   // numero non trovato
        return;

}
```

- **parse.c**

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

#include "defines.h"

#define BUFF_SIZE 4096

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

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

```
        JSMN_STRING = 3,
        JSMN_PRIMITIVE = 4
} jsmntype_t;

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

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

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

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

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

/**
 * Fills token type and boundaries.
 */
static void jsmn_fill_token(jsmntok_t *token, jsmntype_t type,
			int start, int end) {
	token->type = type;
	token->start = start;
	token->end = end;
	token->size = 0;
}

/**
```

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

	start = parser->pos;

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

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

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

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

		int start = parser->pos;

		parser->pos++;

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

			/* Quote: end of string */
			if (c == '\"') {
				if (tokens == NULL) {
					return 0;
				}
```

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

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

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

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

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

                        c = js[parser->pos];
                        switch (c) {
                                case '{': case '[':
```

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

```
                }
                if (token->parent == -1) {
                    if(token->type != type || parser->toksuper == -1) {
                        return JSMN_ERROR_INVAL;
                    }
                    break;
                }
                token = &tokens[token->parent];
            }
#else
            for (i = parser->toknext - 1; i >= 0; i--) {
                token = &tokens[i];
                if (token->start != -1 && token->end == -1) {
                    if (token->type != type) {
                        return JSMN_ERROR_INVAL;
                    }
                    parser->toksuper = -1;
                    token->end = parser->pos + 1;
                    break;
                }
            }
            /* Error if unmatched closing bracket */
            if (i == -1) return JSMN_ERROR_INVAL;
            for (; i >= 0; i--) {
                token = &tokens[i];
                if (token->start != -1 && token->end == -1) {
                    parser->toksuper = i;
                    break;
                }
            }
#endif
            break;
        case '\"':
            r = jsmn_parse_string(parser, js, len, tokens, num_tokens);
```

```
                        if (r < 0) return r;
                        count++;
                        if (parser->toksuper != -1 && tokens != NULL)
                                tokens[parser->toksuper].size++;
                        break;
                case '\t' : case '\r' : case '\n' : case ' ':
                        break;
                case ':':
                        parser->toksuper = parser->toknext - 1;
                        break;
                case ',':
                        if (tokens != NULL && parser->toksuper != -1 &&
                                        tokens[parser->toksuper].type != JSMN_ARRAY &&
                                        tokens[parser->toksuper].type != JSMN_OBJECT) {
#ifdef JSMN_PARENT_LINKS
                                parser->toksuper = tokens[parser->toksuper].parent;
#else
                                for (i = parser->toknext - 1; i >= 0; i--) {
                                        if (tokens[i].type == JSMN_ARRAY || tokens[i].type
== JSMN_OBJECT) {
                                                if (tokens[i].start != -1 && tokens[i].end == -1)
{
                                                        parser->toksuper = i;
                                                        break;
                                                }
                                        }
                                }
#endif
                        }
                        break;
#ifdef JSMN_STRICT
                /* In strict mode primitives are: numbers and booleans */
                case '-': case '0': case '1' : case '2': case '3' : case '4':
                case '5': case '6': case '7' : case '8': case '9':
```

```
                        case 't': case 'f': case 'n' :
                                /* And they must not be keys of the object */
                                if (tokens != NULL && parser->toksuper != -1) {
                                        jsmntok_t *t = &tokens[parser->toksuper];
                                        if (t->type == JSMN_OBJECT ||
                                                        (t->type == JSMN_STRING && t->size != 0)) {
                                                return JSMN_ERROR_INVAL;
                                        }
                                }
#else
                        /* In non-strict mode every unquoted value is a primitive */
                        default:
#endif
                                r = jsmn_parse_primitive(parser, js, len, tokens, num_tokens);
                                if (r < 0) return r;
                                count++;
                                if (parser->toksuper != -1 && tokens != NULL)
                                        tokens[parser->toksuper].size++;
                                break;

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

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

```
                }
        }

        return count;
}

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

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

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

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

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

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

        config[fsize] = 0;
        return fsize;
}

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

        load_file(path);

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

        /* Assume the top-level element is an object */
        if (r < 1 || t[0].type != JSMN_OBJECT) {
```

```
            printf("Object expected\n");
            return 0;
        }

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

- **utilis.h**

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

#include "defines.h"
```

```
void print_stmt_error (MYSQL_STMT *stmt, char *message)
{
	if (message)
		fprintf (stderr, "\e[1m\e[5m\033[40m\033[31m%s\033[0m\e[25m\e[22m\n", message);

	if (stmt != NULL) {
		fprintf (stderr, "\e[1m\e[5m\033[40m\033[31mError %u (%s): %s\033[0m\e[25m\e[22m\n",
			mysql_stmt_errno (stmt), // returns SQLSTATE value
			mysql_stmt_sqlstate(stmt), // returns MYSQL_ERRNO value
			mysql_stmt_error (stmt)); // returns MESSAGE_TEXT value
	}
}

void print_error(MYSQL *conn, char *message)
{
	fprintf (stderr, "\e[1m\e[5m\033[40m\033[31m%s\033[0m\e[25m\e[22m\n", message);
	if (conn != NULL) {
		#if MYSQL_VERSION_ID >= 40101
		fprintf (stderr, "\e[1m\e[5m\033[40m\033[31mError %u (%s): %s\033[0m\e[25m\e[22m\n",
		mysql_errno (conn), mysql_sqlstate(conn), mysql_error (conn));
		#else
		fprintf (stderr, "\e[1m\e[5m\033[40m\033[31mError %u (%s): %s\033[0m\e[25m\e[22m\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;
 5              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 */
        }

10      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);
15              printf(" %-*s |", (int)field->max_length, field->name);
        }
        putchar('\n');

        print_dashes(res_set);
20  }

    void dump_result_set(MYSQL *conn, MYSQL_STMT *stmt, char *title)
    {
        int i;
25      int status;
        int num_fields;       /* number of columns in result */
        MYSQL_FIELD *fields;  /* for result set metadata */
        MYSQL_BIND *rs_bind;  /* for output buffers */
        MYSQL_RES *rs_metadata;
30      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);

        bool is_null[num_fields];

        if (num_fields > 0) {
                /* there is a result set to fetch */
                printf("\n   ");
                print_color(title, "light cyan", ' ', false, true, true, true);


                if((rs_metadata = mysql_stmt_result_metadata(stmt)) == NULL) {
                        finish_with_stmt_error(conn, stmt, "Unable to retrieve result metadata\n",
true);
                }

                dump_result_set_header(rs_metadata);

                fields = mysql_fetch_fields(rs_metadata);

                rs_bind = (MYSQL_BIND *)malloc(sizeof (MYSQL_BIND) * num_fields);
                if (!rs_bind) {
                        finish_with_stmt_error(conn, stmt, "Cannot allocate output buffers\n", true);
```

```
        }
        memset(rs_bind, 0, sizeof (MYSQL_BIND) * num_fields);

        /* set up and bind result set output buffers */
        for (i = 0; i < num_fields; ++i) {

                // Properly size the parameter buffer
                switch(fields[i].type) {
                        case MYSQL_TYPE_DATE:
                        case MYSQL_TYPE_TIMESTAMP:
                        case MYSQL_TYPE_DATETIME:
                        case MYSQL_TYPE_TIME:
                                attr_size = sizeof(MYSQL_TIME);
                                break;
                        case MYSQL_TYPE_FLOAT:
                                attr_size = sizeof(float);
                                break;
                        case MYSQL_TYPE_DOUBLE:
                                attr_size = sizeof(double);
                                break;
                        case MYSQL_TYPE_TINY:
                                attr_size = sizeof(signed char);
                                break;
                        case MYSQL_TYPE_SHORT:
                        case MYSQL_TYPE_YEAR:
                                attr_size = sizeof(short int);
                                break;
                        case MYSQL_TYPE_LONG:
                        case MYSQL_TYPE_INT24:
                                attr_size = sizeof(int);
                                break;
                        case MYSQL_TYPE_LONGLONG:
                                attr_size = sizeof(int);
                                break;
```

```
                default:
                        attr_size = fields[i].max_length;
                        break;
                }

                // Setup the binding for the current parameter
                rs_bind[i].buffer_type = fields[i].type;
                rs_bind[i].buffer = malloc(attr_size + 1);
                rs_bind[i].buffer_length = attr_size + 1;
                rs_bind[i].is_null = &is_null[i];

                if(rs_bind[i].buffer == NULL) {
                        finish_with_stmt_error(conn, stmt, "Cannot allocate output buffers\n",
true);
                }
        }

        if(mysql_stmt_bind_result(stmt, rs_bind)) {
                finish_with_stmt_error(conn, stmt, "Unable to bind output parameters\n",
true);
        }

        /* fetch and display result set rows */
        while (true) {
                status = mysql_stmt_fetch(stmt);

                if (status == 1 || status == MYSQL_NO_DATA)
                        break;

                putchar('|');

                for (i = 0; i < num_fields; i++) {

                        if (rs_bind[i].is_null_value) {
```

```
                printf (" %-*s |", (int)fields[i].max_length, "NULL");
                continue;
            }

            if(*rs_bind[i].is_null){
                printf(" %-*s |", (int)fields[i].max_length, "NULL");
                continue;
            }


            if(rs_bind[i].buffer_type == MYSQL_TYPE_TINY) {
                if (*(char *)rs_bind[i].buffer == 0)
                    printf(" %-*s |", (int)fields[i].max_length+
((int)strlen("FALSE")), "FALSE");
                if (*(char *)rs_bind[i].buffer == 1)
                    printf(" %-*s |", (int)fields[i].max_length+
((int)strlen("TRUE")), "TRUE");
            } else {
                switch (rs_bind[i].buffer_type) {

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

                    case MYSQL_TYPE_DATETIME:
                        date = (MYSQL_TIME *)rs_bind[i].buffer;
                        printf("%02d/%02d/%4d %02d:%02d:%02d %-
*s |", date->day, date->month, date->year,date->hour,date->minute,date->second,
((int)fields[i].max_length)-19, " ");
                        break;

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

```
                printf(" %d-%02d-%02d %-*s |", date->year,
date->month, date->day, 1, "");
                break;

        case MYSQL_TYPE_TIMESTAMP:
                date = (MYSQL_TIME *)rs_bind[i].buffer;
                printf(" %d-%02d-%02d %-*s |", date->year,
date->month, date->day, 1, "");
                break;

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

        case MYSQL_TYPE_FLOAT:
        case MYSQL_TYPE_DOUBLE:
                printf(" %.02f |", *(float *)rs_bind[i].buffer);
                break;

        case MYSQL_TYPE_LONG:
        case MYSQL_TYPE_SHORT:
        case MYSQL_TYPE_TINY:
                printf(" %-*d |", (int)fields[i].max_length, *(int
*)rs_bind[i].buffer);
                break;

        case MYSQL_TYPE_NEWDECIMAL:
                printf(" %-*.02lf |", (int)fields[i].max_length,
*(float*) rs_bind[i].buffer);
                break;

        default:
```

```
                                                printf("  \e[1m\e[5m\033[40m\033[31mERROR:
Unhandled type (%d)\033[0m\e[25m\e[22m\n", rs_bind[i].buffer_type);
                                                abort();
                                        }
                                }
                        }
                        putchar('\n');
                        print_dashes(rs_metadata);
                }

                mysql_stmt_next_result(stmt);

                mysql_free_result(rs_metadata); /* free metadata */

                /* free output buffers */
                for (i = 0; i < num_fields; i++) {
                        free(rs_bind[i].buffer);
                }
                free(rs_bind);
        }
}
```

- **Makefile**

```
CC = gcc
OTPS = `mysql_config --cflags --include --libs`

all:
        $(CC) -Wall -Wextra *.c -o client $(OTPS)

clean:
        rm *.o client
```

- **UCC.json**

```
{
        "host": "localhost",
```

```
    "username" : "USCC",
    "password" : "usccpassword",
    "port" : 3306,
    "database" : "BachecaElettronicadb"
}
```

- **USCC.json**

```
{
    "host": "localhost",
    "username" : "USCC",
    "password" : "usccpassword",
    "port" : 3306,
    "database" : "BachecaElettronicadb"
}
```

- **Administrator.json**

```
{
    "host": "localhost",
    "username" : "USCC",
    "password" : "usccpassword",
    "port" : 3306,
    "database" : "BachecaElettronicadb"
}
```

- **Login.json**

```
{
    "host": "localhost",
    "username" : "USCC",
    "password" : "usccpassword",
    "port" : 3306,
    "database" : "BachecaElettronicadb"
}
```