Basi di Dati

Progetto A.A. 2021/2022

SISTEMA DI ASTE ONLINE

0253129

Daniele d’Errico

Indice

1. Descrizione del Minimondo

Sistema di aste online

Una casa d'aste intende realizzare un sistema online di aste. Il sistema deve consentire agli amministratori la gestione degli oggetti che si vogliono pubblicare e tutto il ciclo di vita delle aste. Gli utenti del sistema, previa registrazione, hanno la possibilità di fare offerte su un qualsiasi oggetto. Al termine dell'asta, l'offerta maggiore sarà quella che avrà vinto l'asta. Alla registrazione, gli utenti devono comunicare il codice fiscale, il nome, il cognome, la data di nascita, la città di nascita, le informazioni sulla propria carta di credito (intestatario, numero, data di scadenza, codice CVV). Inoltre, essi devono fornire un indirizzo cui consegnare eventuali oggetti acquistati.

Gli amministratori gestiscono l'inserimento degli oggetti. Ogni oggetto è caratterizzato da un codice alfanumerico univoco, da una descrizione, da uno stato (ad esempio "come nuovo", "in buone condizioni", "non funzionante", ecc.), da un prezzo di base d'asta, da una descrizione delle dimensioni e da un attributo colore. Quando viene inserito un nuovo oggetto nel sistema, gli amministratori possono decidere la durata dell'asta, da un minimo di un giorno ad un massimo di sette giorni. Inoltre, a ciascuna asta viene associata una categoria. Le categorie appartengono ad un titolario gerarchico, organizzato su un massimo di tre livelli. La gestione delle categorie degli oggetti afferisce sempre agli amministratori del sistema.

Gli utenti del sistema possono visualizzare in qualsiasi momento tutte le aste aperte. Quando un'asta viene visualizzata, gli utenti ottengono tutte le informazioni legate allo stato attuale della stessa, tra cui il tempo mancante alla chiusura, il numero di offerte fatte, l'importo dell'offerta massima attuale. Non possono però visualizzare chi è che ha effettuato l'offerta massima.

Dato un oggetto in asta, gli utenti possono fare un'offerta, maggiore del valore attuale di offerta. La granularità di incremento delle offerte è di multipli di 50 centesimi di euro. Inoltre, un utente che ha attualmente piazzato l'offerta massima, può sfruttare la funzionalità di "controfferta automatica". Tale funzionalità permette all'utente di indicare un importo massimo con cui si intende rilanciare l'offerta, qualora un altro utente faccia

un’offerta maggiore. La gestione delle offerte pertanto funziona nel modo seguente. L’utente A indica un importo I con cui vuole rilanciare l’offerta nei confronti dell’utente B che è attualmente il migliore offerente. L’utente B ha anche indicato un importo di controfferta C. Se C > I, il sistema indicherà come miglior offerente l’utente A, con importo temporaneo I, ma immediatamente dopo indicherà nuovamente l’utente B come migliore offerente, con un importo di I + 0,50€.

Il sistema tiene traccia, per ogni oggetto, di tutte le offerte che sono state fatte e dell’instante temporale in cui queste sono state inserite nel sistema. Ciò significa che tutte le transazioni automatiche generate dal sistema di controfferta automatica devono essere registrate nel sistema. Gli amministratori, in ogni momento, possono generare un report che, dato un oggetto, mostri lo storico delle offerte, indicante anche quali sono state generate dal sistema di controfferta automatica.

Gli utenti, in ogni momento, possono visualizzare l’elenco degli oggetti aggiudicati e l’elenco degli oggetti per i quali è presente un’asta in corso cui hanno fatto almeno un’offerta.

2. Analisi dei Requisiti

Identificazione dei termini ambigui e correzioni possibili

Linea	Termine	Nuovo termine	Motivo correzione
4	Amministratore	Utente Amministratore	Il concetto di Amministratore è un'estensione di quello di Utente, pertanto si ritiene opportuno ridefinire entrambi per distinguere meglio i due concetti.
5	Utente del Sistema	Utente Base	Si vuole evidenziare il fatto che un Utende, rispetto ad un Amministratore , può usufruire ad un diverso tipo di servizi.
10	Indirizzo cui consegnare eventuali oggetti acquistati	Indirizzo di consegna	Si esprime lo stesso concetto in maniera più semplice.
13	Codice alfanumerico univoco	Identificatore	Si esprime lo stesso concetto in forma più compatta.
16	Decidere la durata dell'asta	Impostare la durata dell'asta	Il nuovo termine è meno ambiguo.
22	Aste aperte	Aste attive	Il nuovo termine è meno ambiguo.
28	Valore attuale di offerta	Offerta massima attuale	Terminologia più precisa.
48	Asta in corso	Asta attiva	Il nuovo termine è meno ambiguo.

Specifica disambiguata

Sistema di aste online

Una casa d'aste intende realizzare un sistema online di aste. Il sistema deve consentire agli Utenti Amministratori la gestione degli oggetti che si vogliono pubblicare e tutto il ciclo di vita delle aste. Gli Utenti Base, previa registrazione, hanno la possibilità di fare offerte su un qualsiasi oggetto. Al termine dell'asta, l'offerta maggiore sarà quella che avrà vinto l'asta. Alla registrazione, gli Utenti Base devono comunicare il codice fiscale, il nome, il cognome, la data di nascita, la città di nascita, le informazioni sulla propria carta di credito (intestatario, numero, data di scadenza, codice CVV). Inoltre, essi devono fornire un indirizzo di consegna.

Gli Utenti Amministratori gestiscono l'inserimento degli oggetti. Ogni oggetto è caratterizzato da un identificatore, da una descrizione, da uno stato (ad esempio "come nuovo", "in buone condizioni", "non funzionante", ecc.), da un prezzo di base d'asta, da una descrizione delle dimensioni e da un attributo colore. Quando viene inserito un nuovo oggetto nel sistema, gli Utenti Amministratori possono impostare la durata dell'asta, da un minimo di un giorno ad un massimo di sette giorni. Inoltre, a ciascuna asta viene associata una categoria. Le categorie appartengono ad un titolario gerarchico, organizzato su un massimo di tre livelli. La gestione delle categorie degli oggetti afferisce sempre agli Utenti Amministratori.

Gli Utenti Base del sistema possono visualizzare in qualsiasi momento tutte le aste attive. Quando un'asta viene visualizzata, gli utenti ottengono tutte le informazioni legate allo stato attuale della stessa, tra cui il tempo mancante alla chiusura, il numero di offerte fatte, l'importo dell'offerta massima attuale. Non possono però visualizzare chi è che ha effettuato l'offerta massima.

Dato un oggetto in asta, gli utenti possono fare un'offerta, maggiore dell'offerta massima attuale. La granularità di incremento delle offerte è di multipli di 50 centesimi di euro. Inoltre, un utente che ha attualmente piazzato l'offerta massima, può sfruttare la funzionalità di "controfferta automatica". Tale funzionalità permette all'utente di indicare un importo massimo con cui si intende rilanciare l'offerta, qualora un altro utente faccia un'offerta maggiore. La gestione delle offerte pertanto funziona nel modo seguente. L'utente A indica un importo I con cui vuole rilanciare l'offerta nei confronti dell'utente B che è attualmente il migliore offerente. L'utente B ha anche indicato un importo di controfferta C. Se C > I, il sistema indicherà come miglior offerente l'utente A, con importo temporaneo I, ma immediatamente dopo indicherà nuovamente l'utente B come migliore offerente, con un importo di I + 0,50€.

Il sistema tiene traccia, per ogni oggetto, di tutte le offerte che sono state fatte e dell'instante temporale in cui queste sono state inserite nel sistema. Ciò significa che tutte le transazioni automatiche generate dal sistema di controfferta automatica devono essere registrate nel sistema. Gli Utenti Amministratori, in ogni momento, possono generare un report che, dato un oggetto, mostri lo storico delle offerte, indicante anche quali sono state generate dal sistema di controfferta automatica.

Gli utenti, in ogni momento, possono visualizzare l'elenco degli oggetti aggiudicati e l'elenco degli

oggetti per i quali è presente un’asta attiva cui hanno fatto almeno un’offerta.

Glossario dei Termini

Termine	Descrizione	Sinonimi	Collegamenti
Utente Amministratore	Attore che si occupa della gestione degli oggetti che vuole pubblicare.	Amministratore, Amministratore del Sistema.	Oggetto, Asta, Categoria.
Utente Base	Attore che si collega con il sistema per partecipare alle aste.	Utente, Utente del Sistema.	Oggetto, Asta, Offerta, Controfferta automatica.
Oggetto	Elemento fisico che si intende vendere o comprare tramite un’asta.		Utente Amministratore, Utente Base, Asta.
Asta	Competizione tra Utenti Base, I quali cercano di aggiudicarsi un oggetto, rilanciando sull’offerta massima.		Utente Base, Utente Amministratore, Oggetto, Offerta, Controfferta automatica.
Offerta	Importo in denaro che un Utente Base intende offrire, rilanciando sull’offerta massima attuale.		Oggetto, Asta, Utente Base.
Controfferta automatica	Automatizzazione del concetto di Offerta.	Rilancio automatico, Rilancio.	Oggetto, Asta, Utente Base.
Categoria	Categoria di appartenenza dell’Oggetto.	Tipo, Tipologia.	Oggetto, Asta, Utente Amministratore.

Raggruppamento dei requisiti in insiemi omogenei

Frasi relative all’Utente Base
L’Utente Base è l’attore che si interfaccia al sistema per concorrere alle aste. Per utilizzare il sistema, ogni Utente Base deve registrarsi, fornendo i seguenti dati: codice fiscale, nome, cognome, data di nascita, città di nascita, informazioni sulla propria carta di credito (intestatario, numero, data di scadenza, codice CVV), indirizzo di consegna. Un Utente Base può: 1) Visualizzare in qualsiasi momento tutte le aste attive. 2) Visualizzare, per ogni asta, lo stato attuale di essa (i.e. il tempo mancante alla chiusura, il

numero di offerte fatte, l'importo dell'offerta massima attuale…)
3) Concorrere in qualsiasi momento in tutte le aste attive.
4) Una volta piazzata un'offerta massima per un oggetto all'asta, sfruttare la funzionalità di "controfferta automatica" (funzionalità che permette all'utente di indicare un importo massimo con cui si intende rilanciare l'offerta, qualora un altro utente faccia un'offerta maggiore).
5) Visualizzare l'elenco degli oggetti aggiudicati.
6) Visualizzare l'elenco degli oggetti per i quali è attiva un'asta cui hanno fatto almeno un'offerta.

Un Utente Base non può:

1) Visualizzare l'Utente che ha effettuato l'offerta massima su un Oggetto all'asta.

Frasi relative all'Utente Amministratore

L'Utente Amministratore si occupa della gestione degli oggetti che vuole pubblicare, e del ciclo di vita
delle Aste.

Un Utente Amministratore può:

1) Inserire un nuovo Oggetto nel Sistema.
2) All'inserimento di un nuovo Oggetto, impostare la durata dell'Asta (minimo 1 giorno, massimo 7 giorni).
3) Assegnare la Categoria di riferimento per un Oggetto.
4) Generare un report che mostri lo storico delle offerte per un Oggetto, distinguendo le offerte dalle "controfferte automatiche".

Frasi relative all'Oggetto

Un Oggetto è ciò che l'Utente Base intende acquistare partecipando ad un'asta.

Ogni oggetto è caratterizzato da:

1) un identificatore.
2) un nome.
3) una descrizione.
4) una categoria di appartenenza.

5) uno stato (ad esempio "come nuovo", "in buone condizioni", "non funzionante", ecc.).
6) un prezzo di base d'asta.
7) una descrizione delle dimensioni.
8) un attributo colore.

Il Sistema tiene traccia, per ogni oggetto, di tutte le offerte che sono state fatte e dell'instante temporale in cui queste sono state inserite nel sistema.

Frasi relative all'Asta

Un'Asta è una competizione tra Utenti Base, i quali cercano di aggiudicarsi un Oggetto di interesse, rilanciando sull'offerta massima attuale.
L'asta è vinta, allo scadere della sua durata, dall'Utente Base che ha piazzato l'offerta massima.

In ogni momento, lo stato di un'asta è definito da:

1) tempo mancante alla chiusura.
2) Oggetto associato all'asta.
3) numero di offerte fatte.
4) importo dell'offerta massima attuale.

Frasi relative all'Offerta

Un'Offerta è l'importo in denaro che un Utente Base intende offrire, a rilancio sull'offerta massima attualmente raggiunta per l'oggetto di interesse, per vincere l'asta e aggiudicarsi tale oggetto.
Al termine dell'asta, l'offerta massima sarà quella che avrà vinto l'asta.
Il massimo incremento delle offerte è di multipli di 50 centesimi di euro.
La gestione delle offerte pertanto funziona nel modo seguente:
L'Utente A indica un importo I con cui vuole rilanciare l'offerta nei confronti dell'Utente B, che è attualmente il migliore offerente.
L'Utente B ha anche indicato un importo di controfferta C.
Se C > I, il sistema indicherà come miglior offerente l'utente A, con importo temporaneo I, ma immediatamente dopo indicherà nuovamente l'utente B come migliore offerente, con un importo di I + 0,50€.
Il sistema tiene traccia, per ogni oggetto, di tutte le offerte che sono state fatte e dell'instante temporale in cui queste sono state inserite nel sistema.

Frasi relative alla Controfferta Automatica

Automatizzazione del concetto di Offerta.
La funzionalità di Controfferta Automatica permette all'Utente Base di indicare un importo massimo con cui si intende rilanciare l'offerta, qualora un altro utente faccia un'offerta maggiore.
Tutte le transazioni automatiche generate dal sistema di controfferta automatica devono essere registrate nel sistema.

Frasi relative alla Categoria

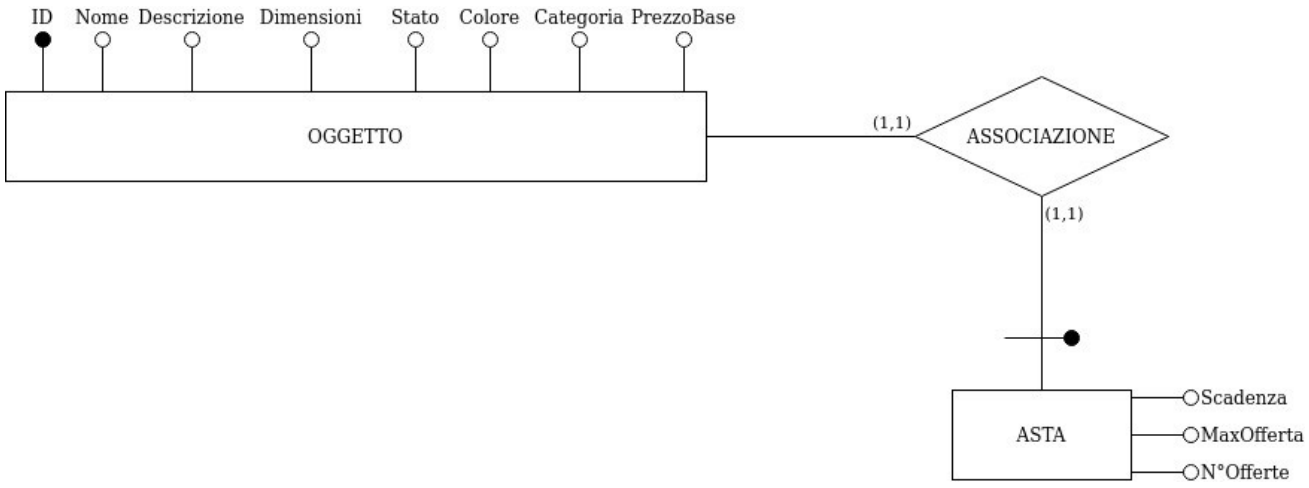
A ciascuna asta viene associata una categoria, quella dell'Oggetto all'asta.
Le categorie appartengono ad un titolario gerarchico, organizzato su un massimo di tre livelli.

3. Progettazione concettuale

Costruzione dello schema E-R

STEP 1

Per la schematizzazione E-R si decide di utilizzare un approccio inside-out, ovveor partire dal concetto principale del sistema e costruirci intorno una sorta di base su cui costruire l’applicazione.
Il concetto principale su cui si basa l’applicazione è quello di Asta; inoltre quest’ultimo viene legato in modo stretto al concetto di Oggetto.
La corrispondenza biunivoca tra questi due concetti fa dedurre che non può esistere un’Asta che non sia associata ad un’Oggetto; per questo motivo l’entià Asta viene resa entità debole, identificata dall’oggetto cui è associata.
Si può pensare di inserire in concetto di Asta Attiva tramite Design Pattern (instance-of), diversificando il concetto di Asta generica con quello di Asta attiva; questo però risulta superfluo in quenato un Utente Amministratore al momento della pubblicazione dell’Oggetto specifica la durata dell’Asta e così da quel momento l’asta risulterà attiva; quindi risulta superfluo mantenere una rappresentazione di Asta generica.
Si può definire l’attributo “scadenza” per l’entità Asta, in questo modo, al momento della pubblicazione, da parte di un Utente Amministratore, di un Oggetto da mettere all’Asta, verrà stabilito l’istante temporale della scadenza dell’Asta.
Raggiunto il termine di scadenza, allora l’asta sarà automaticamente considerata conclusa, l’oggetto aggiudicato, e non sarà possibile effettuare offerte ulteriori.
Questa scelta permette di semplificare notevolmente la schematizzazione, perché non ci sarà bisogno di mantenere Entità separate che distinguano i concetti di Asta, Asta attiva e Asta conclusa.



STEP 2

Si estende lo schema in direzione degli attori del Sistema, ovvero l'Utente Base e l'Utente Amministratore.

a) L'Utente Amministratore espone uno o più Oggetti, ognuno di essi implica l'inizio del ciclo di vita di un'asta.
b) In ogni momento, l'Utente Amministratore può documentare in un report le offerte che sono state fatte su una delle aste di sua competenza, quindi sui rispettivi oggetti, distinguendo le offerte dalle controfferte automatiche.
c) L'Utente Base fa le sue offerte, in qualsiasi momento, su qualsiasi Asta disponibile nel Sistema.
d) Secondo lo schema proposto, l'unico vincolo che si dovrebbe aggiungere è che se la scadenza di un'Asta è verificata, allora non sono più accettate offerte da parte di Utenti.

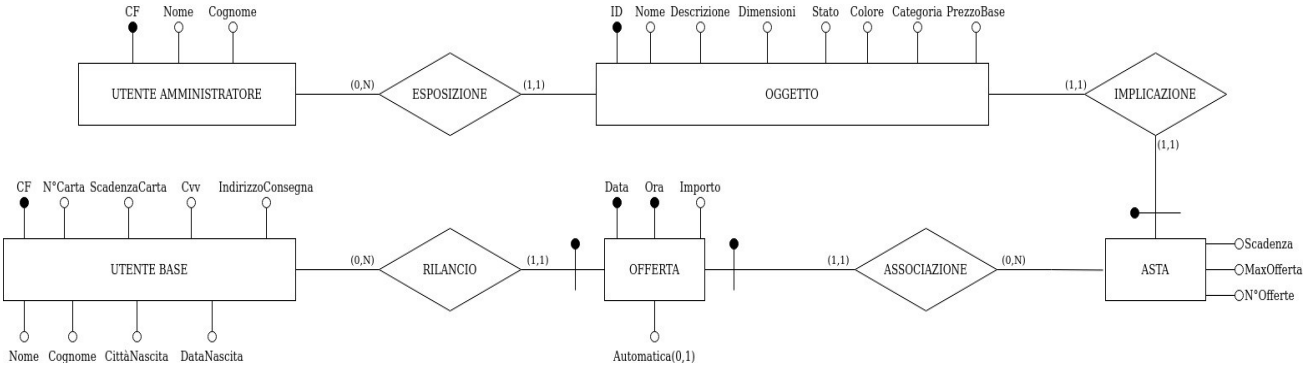
   Senza scrivere questo vincolo, sarebbe opportuno storicizzare il concetto di Asta aggiungendo una generalizzazione totale che divida le aste in Aste Attive e Aste concluse: il problema è che il concetto di "Asta Conclusa" non aggiunge ulteriori informazioni rispetto al generico concetto di Asta.
e) L'Offerta è un'entità "molto debole", in quanto risulta definita in riferimento all'Utente Base e all'Oggetto di interesse, oltre che dall'istante temporale in cui viene lanciata.

   L'Offerta non può essere intesa come una relazione binaria tra Utente Base ed Asta, poiché ciò implicherebbe che ogni Utente avrebbe a disposizione una sola offerta per ognuna delle aste disponibili.

   È stato dunque applicato in questo caso un design pattern di "reificazione di relazione",

essendo per giunta l'Offerta un concetto ricco di attributi caratterizzanti.

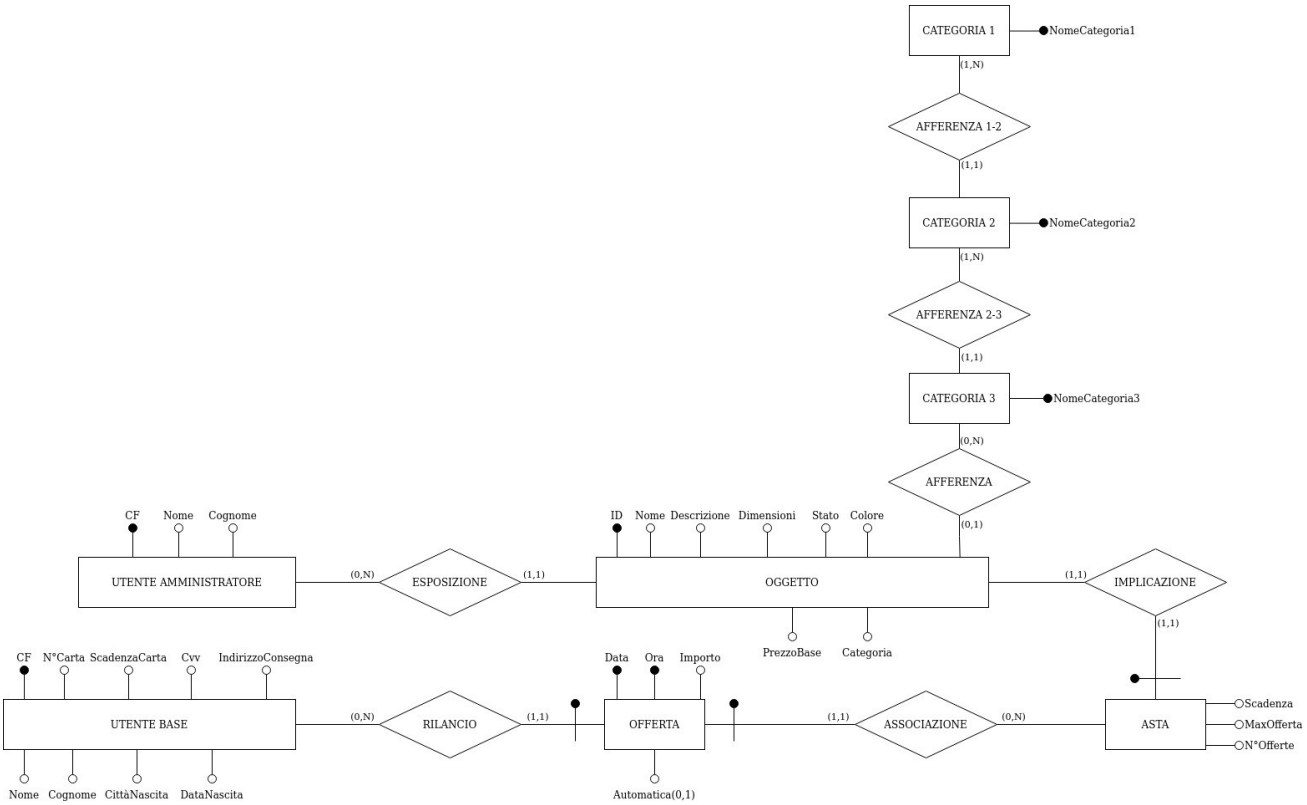
La Controfferta Automatica è semplicemente rappresentabile come un caso particolare di offerta, ovvero il caso in cui un'Offerta sia automatizzata: questo può essere semplicemente schematizzato come un attributo opzionale ("Automatica") dell'entità Offerta.



STEP 4

Ciò che rimane da rappresentare è l'afferenza a una categoria di un Oggetto.

Le categorie sono organizzate in tre livelli su un titolario gerarchico, e la gestione di queste è riservata agli Utenti Amministratori.



Integrazione finale

È necessario testare i requisiti di buona progettazione dello schema, quali correttezza, completezza, leggibilità e minimalità.

Correttezza: lo schema è sintatticamente e semanticamente corretto.

Completezza: lo schema non presenta nessuma informazione sul funzionamento della controfferta automatica (aspetto importante dle sistema); è necessario aggiungere l’attributo “ImportoControffertaMax” sull’entità Asta in quanto in ogni momento, possa esistere al più una Controfferta Automatica, qualora sia impostata dal miglior offerente attuale.

Riportando un possibile scenario: Se un Utente Base piazza un’offerta massima su una certa Asta, questo sarà il miglior offerente (attuale) di quell’Asta.

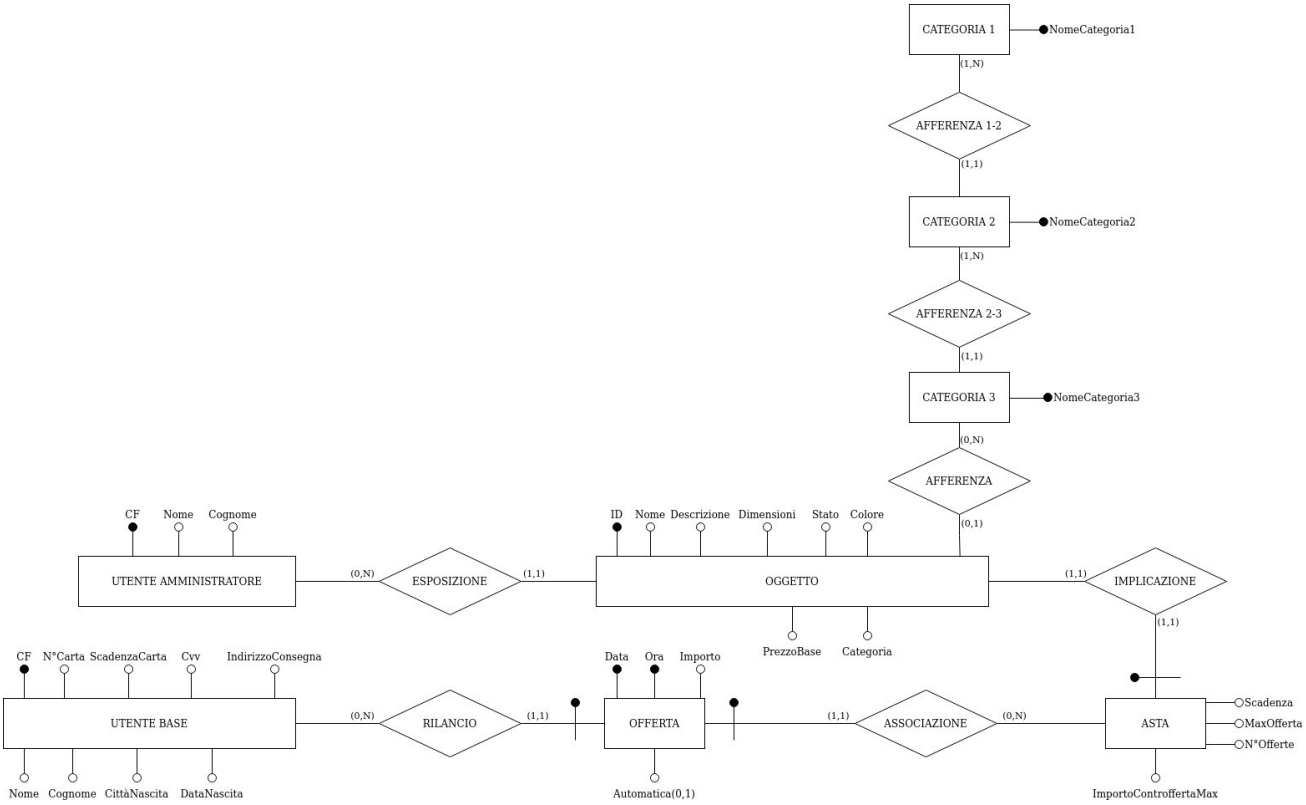
L’attributo opzionale “ImportoControffertaMax” sull’entità Asta permette a un Utente in vantaggio di indicare l’importo massimo con cui, a partire dalla sua offerta, rilanciare un’eventuale offerta proveniente da un altro Utente.

Se la controfferta va a buon fine, ma il valore effettivo rilanciato è minore del valore massimo di controfferta che il miglior offerente aveva impostato, allora il valore effettivo rilanciato sarà sottratto al valore massimo di controfferta, e il Sistema potrà:

a) Registrare, tra le offerte dell’Utente in vantaggio, l’offerta generata dalla funzionalità di controfferta automatica, segnalata dalla partecipazione dell’attributo “Automatica” sull’entità Offerta.
b) Mantenere lo stato del Miglior Offerente, aggiornando il valore dell’importo massimo di controfferta.

Leggibilità: lo schema risulta leggibile per quato riguarda le associazioni in entrambi i versi.

Minimalità: lo schema risulta minimale.



Regole aziendali

È necessaroi aggiungere alcune regole che non si è potuto esprimere attraverso lo schema.

1. Un Utente Base non deve fare offerte su Aste il cui attributo "Scadenza" indica che tale Asta è terminata.
2. Un Utente Amministratore non deve documentare le Offerte fatte su un'Asta associata ad un Oggetto non esposto da lui.

Dizionario dei dati

Entità	Descrizione	Attributi	Identificatori
Utente Base		CF, Nome, Cognome, DataNascita, CittàNascita, IndirizzoConsegna, N°Carta, ScadenzaCarta, Cvv	CF

Utente Amministratore		CF, Nome, Cognome	CF
Oggetto		ID, Nome, Descrizione, Dimensioni, Stato, Colore, Categoria, PrezzoBase	ID
Asta		N°Offerte, Scadenza, MaxOfferta, ImportoControffertaMax	ID (in riferimento all’ oggetto)
Offerta		Data, Ora, Importo, Automatica	CF (in riferimento all’Utente Base), ID (in riferimento all’oggetto), Data, Ora
Categoria1		NomeCategoria1	NomeCategoria1
Categoria2		NomeCategoria2	NomeCategoria2
Categoria3		NomeCategoria3	NomeCategoria2

4. Progettazione logica

Volume dei dati

Per iniziare, è di buona utilità porre in relazione Entità e Associazioni per calcolare il loro rapporto in termini di Volume:

1) Per ogni Utente Amministratore, considero 20 Utenti Base;
2) Per ogni Utente Amministratore, considero una media di 20 Oggetti esposti, quindi 20 Aste;
3) Per ogni Asta, considero una media di 1 offerta per ogni Utente Base, tenendo presente che per numeri ragionevolmente grandi, cerchie ristrette di Utenti Base puntano ad una specifica Asta;
4) Un’occorrenza di Esposizione è data da una coppia Utente Amministratore-Oggetto. Considerando che ogni Oggetto è associato ad uno e un solo Utente Amministratore, il volume della relazione Esposizione sarà lo stesso dell’entità Oggetto.
5) Allo stesso modo, il volume della relazione Implicazione sarà uguale al volume dell’entità Oggetto.

Concetto nello schema	Tipo[1]	Volume atteso
Utente Amministratore	E	20
Utente Base	E	400
Asta	E	400
Offerta	E	160.000
Oggetto	E	400
Miglior Offerente	E	400
Esposizione	R	400
Implicazione	R	400
Rilancio	R	160.000
Associazione	R	160.000
Categoria1	E	5
Categoria2	E	20
Categoria3	E	60

1 Indicare con E le entità, con R le relazioni

Tavola delle operazioni

Cod.	Descrizione	Frequenza attesa
1	Registrazione nuovo Utente nel Sistema	2 a settimana
2	Login	150 al giorno
3	Inizializzazone Asta per un nuovo Oggetto	10 al giorno
4	Visualizzazione di tutte le Aste attive	200 al giorno
5	Aggiornamento di un’Offerta da parte di un’Utente	250 al giorno
6	Visualizza lo stato di un’Asta	500 al giorno
7	Visualizza tutti gli oggetti aggiudicati da un’Utente Base	100 al giorno
8	Trova tutte le Aste per cui un Utente ha proposto almeno un’Offerta	350 al giorno
9	Visualizza le Aste associate ad una Categoria	250 al giorno
10	Visualizza tutte le Offerte fatte in un’Asta	75 al giorno
11	Visualizza le Aste indette da un Utente Amministratore	25 al giorno
12	Trova tutte le Offerte generate in un’Asta dal sistema di controfferta automatica	30 al giorno
13	Visualizza le Aste attive associate al nome di un Oggetto	500 al giorno
14	Inserisci una nuova Categoria	1 ogni due mesi
15	Aggiorna una Categoria	1 ogni due mesi
16	Elimina una Categoria	1 ogni due mesi

Questa tipologia di Sistema prevede una continua espansione pertanto sarebbe opportuno stimare le tavole dei volumi e delle operationi in funzione del tempo.

Costo delle operazioni

Si decide di riportare per ognioperazione la relativa tavola degli accessi.

Operazione 1: Registrazione nuovo Utente nel Sistema.

Concetto	Costrutto	Accessi	Tipo
Utente Base/ Utente Amministratore	E	1	S

Operazione 2: Login.

Concetto	Costrutto	Accessi	Tipo
Utente Base/ Utente Amministratore	E	1	L

Operazione 3: Inizializzazone Asta per un nuovo Oggetto.

Concetto	Costrutto	Accessi	Tipo
Asta	E	1	S
Oggetto	E	1	L
Esposizione	R	1	L

Operazione 4: Visualizzazione di tutte le Aste attive.

Si sono considerate attive la metà delle aste (400/2).

Concetto	Costrutto	Accessi	Tipo
Asta	E	200	L

Operazione 5: Aggiornamento di un’Offerta da parte di un’Utente.

Concetto	Costrutto	Accessi	Tipo
Offerta	E	1	S
Rilancio	R	1	S
Associazione	R	1	S
Miglior Offerente	E	1	S

Operazione 6: Visualizza lo stato di un’Asta.

Concetto	Costrutto	Accessi	Tipo
Asta	E	1	L

Operazione 7: Visualizza tutti gli oggetti aggiudicati da un’Utente Base.

Considerando che gli Oggetti aggiudicati è la stessa delle Aste concluse aggiudicate da uno stesso Utente; che il numero di aste concluse è pari alla metà delle Aste (400/2) e quindi in media un Utente si aggiudica 200/400 = 0,5 oggetti.

Concetto	Costrutto	Accessi	Tipo
Oggetto	E	1	L
Vantaggio	R	1	L

Operazione 8: Trova tutte le Aste per cui un Utente ha proposto almeno un’Offerta.

Considerando che in media un Utente partecipa ad almeno 2 aste.

Concetto	Costrutto	Accessi	Tipo
Utente Base	E	1	L
Asta	E	2	L
Rilancio	R	2	L

Operazione 9: Visualizza le Aste associate ad una Categoria.

Considerando che in media gli oggetti sono suddivisi in 5 categorie.

Concetto	Costrutto	Accessi	Tipo
Asta	E	80	L
Oggetto	E	80	L
Implicazione	R	80	L

Operazione 10: Visualizza tutte le Offerte fatte in un'Asta.

Considerando che in media ogni asta riceve 1 offerta per ogni Utente.

Concetto	Costrutto	Accessi	Tipo
Asta	E	1	L
Offerta	E	400	L
Associazione	R	400	L

Operazione 11: Visualizza le Aste indette da un Utente Amministratore.

Concetto	Costrutto	Accessi	Tipo
Utente Amministratore	E	1	L
Asta	E	20	L
Esposizione	R	20	L
Implicazione	R	20	L

Operazione 12: Trova tutte le Offerte generate in un'Asta dal sistema di controfferta automatica.

Consideranto che in media il 20% delle offerte è generato dal sistema di controfferta automatica.

Concetto	Costrutto	Accessi	Tipo
Asta	E	1	L
Offerta	E	80	L
Associazione	R	80	L

Operazione 13: Visualizza le Aste attive associate al nome di un Oggetto.

Consideranto in media la presenza di 2 omonimi per uno stesso oggetto.

Concetto	Costrutto	Accessi	Tipo
Asta	E	3	L
Implicazione	R	3	L
Oggetto	E	3	L

Operazione 14: Inserisci una nuova Categoria.

Concetto	Costrutto	Accessi	Tipo
Categoria3	E	1	S
Categoria2	E	1	L

Operazione 15: Aggiorna una Categoria.

Concetto	Costrutto	Accessi	Tipo
Categoria3 (2 o 1)	E	1	S

Operazione 16: Elimina una Categoria.

Concetto	Costrutto	Accessi	Tipo
Categoria3 (2 o 1)	E	1	L

Ristrutturazione dello schema E-R

Lo schema concettuale presentato è funzionante e navigabile, tuttavia attraverso un'analisi dei costi risulta evidente ottimizzare alcuni aspetti del Sistema.
Analizzando il costo dele operazioni risulta rispecchiata seppur approssimativamente la regola "dell'ottanta-venti", secondo cui il'80% del carico applicativo è generato dal 20% delle operazioni.
In particolare si possono migliorare due fattori per ottimizzare il Sistema:

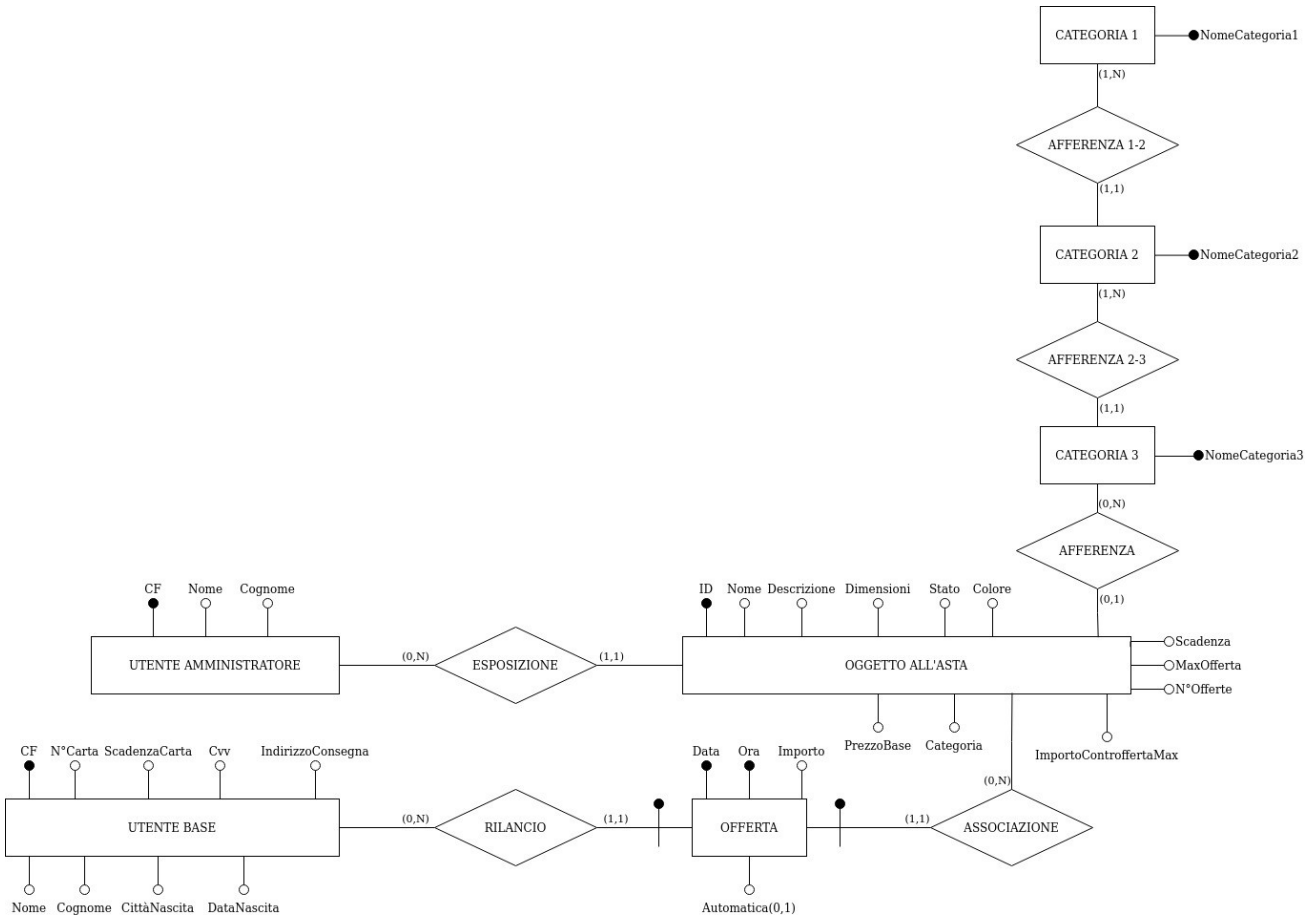
1) È presente un ridondanza forte tra le entità Oggetto e Asta, che inoltre sono legate da una relazione 1 a 1; l'esistenza in contemporanea delle due incrementa il numero di accessi in lettura.
2) È presente un grande incremento in termine di volume di dati per mantenere lo storico di tutte le offerte che sono state fatte per ogni asta (attiva o conclusa).

Analisi delle ridondanze

Come introdotto le entità Oggetto ed Asta generano una forte ridondanza in quanto possono rappresentare lo stesso concetto all'interno del Sistema; non risulta conveniente mantenere i due concetti separati in quanto l'inserimento di un Oggetto nel Sistema implica l'esposizione dello stessa in un'Asta e vincere l'Asta implica aggiudicarsi l'Oggetto.
Inoltre l'Applicazione lavora molto in lettura e quindi risulterebbe poco funzionale mantenere una relazione 1 a 1 che causerebbe una duplicazione degli accessi per recuperare informazioni che invece potrebbro essere unificate senza problemi.
Come soluzione si decide di utilizzare "l'Accorpamento di Concetti"; ovvero unire i due concetti

Oggetto ed Asta in “Oggetto all’Asta” con in conseguente accorpamento di tutti gli attributi come motrato nello schema.

Da notare che questa aggiunta riunifica le operazioni “Inserisci un nuovo Oggetto nel Sistema” e “Inizializza l’asta per un nuovo Oggetto”, che non avevano molto senso prese singolarmente.



Eliminazione delle Generalizzazioni

Non sono presenti generalizzazioni all’interno dello schema.

Scelta degli identificatori primari

Per ciò che riguarda la scelta degli identificatori principali, lo schema è corretto.

Trasformazione di attributi e identificatori

Gli identificatori esterni presenti per l’entità Offerta sono necessari, in quanto ogni offerta può essere identificata solo se si conoscono l’offerente, l’oggetto di interesse, la data e l’ora in cui essa viene

effettuata.
Si potrebbe pensare di aggiungere un codice identificativo univoco per ogni offerta, ma sarebbe uno spreco di risorse hardware (in termini di volume dei dati).
Inoltre, è da notare che, considerando un modello relazionale dei dati, il doppio identificatore esterno implica che la relazione Offerta rappresenterà obbligatoriamente anche le associazioni “Rilancio” e “Associazione”, che non avranno dunque una propria rappresentazione tabellare.

Traduzione di entità e associazioni

Schema relazionale:

UTENTE BASE (**CF**, Nome, Cognome, DataNascita, CittàNascita, N°Carta, ScadenzaCarta, Cvv, IndirizzoConsegna)

UTENTE AMMINISTRATORE (**CF**, Nome, Cognome)

OGGETTO ALL'ASTA (**ID**, Nome, Descrizione, Dimensioni*, Espositore, Stato, Colore*, Categoria*, PrezzoBase, N°Offerte,MaxOfferta*, MigliorOfferente*, ImportoControffertaMax*, Scadenza)

OFFERTA (**Utente**, **Oggetto**, **Istante**, Automatica, Importo, ImportoControfferta*)

CATEGORIA1 (**Nome**)

CATEGORIA2 (**Nome**, NomeCategoria1*)

CATEGORIA3 (**Nome**, NomeCategoria2*)

Schema referenziale con vincoli di integrità referenziale:

Normalizzazione del modello relazionale

Consideriamo separatamente le varie relazioni.
Per ognuna delle relazioni, bisogna determinare le dipendenze funzionali tra gli attributi costituenti, e verificare se sono soddisfatte le forme normali.
La soddisfazione delle forme normali determina la qualità dello schema relazionale, in termini di ridondanze e anomalie.
Di fatto, lo schema relazionale non contiene, nelle sue relazioni, alcun tipo di dipendenza funzionale.
Ciò implica inequivocabilmente che la soluzione è già ottimale e in particolare non ridondante: ci troviamo nel caso in cui la forma normale di “Boyce e Codd” è soddisfatta, e di conseguenza lo sono anche 1NF, 2NF, 3NF.

5. Progettazione fisica

Utenti e privilegi

Gli Utenti previsti all'interno dell'applicazione sono l'Utente Base e l'Utente Amministratore.

1) Privilegi concessi agli Utenti Login, stando alle specifiche dei requisiti:
   - Operazione 2: login (EXECUTE).

2) Privilegi concessi agli Utenti Base, stando alle specifiche dei requisiti:
   - Operazione 1: registrare un nuovo utente base nel sistema (EXECUTE).
   - Operazione 4: visualizza tutte le aste attive (EXECUTE).
   - Operazione 5: registra l'offerta di un utente per un'asta (rilancia un'offerta) (EXECUTE).
   - Operazione 6: visualizza lo stato di un'asta (EXECUTE).
   - Operazione 7: visualizza tutti gli oggetti aggiudicati (EXECUTE).
   - Operazione 8: trova tutte le aste attive in cui l'utente ha lanciato almeno un'offerta (EXECUTE).
   - Operazione 9: visualizza le aste attive associate ad una categoria (EXECUTE).
   - Operazione 11: visualizza le aste indette da un utente amministratore (EXECUTE).
   - Operazione 13: visualizza le aste attive associate al nome di un oggetto (EXECUTE).

2) Privilegi concessi all'Utente Amministratore, stando alle specifiche dei requisiti:
   - Operazione 1: registrare un nuovo utente amministratore nel sistema (EXECUTE).
   - Operazione 3: inizializza l'asta per un nuovo oggetto (EXECUTE).
   - Operazione 6: visualizza lo stato di un'asta (EXECUTE).
   - Operazione 10: trova tutte le offerte fatte in un'asta (EXECUTE).
   - Operazione 11: visualizza le aste indette (EXECUTE).
   - Operazione 12: trova tutte le offerte generate in un'asta dal sistema di controfferta automatica (EXECUTE).
   - Operazione 14: inserisci una nuova categoria (EXECUTE).
   - Operazione 15: aggiorna una categoria (EXECUTE).
   - Operazione 16: elimina una categoria (EXECUTE).

Strutture di memorizzazione

Tabella UTENTE BASE		
Colonna	**Tipo di dato**	**Attributi**[2]
CF	CHAR(16)	PK, NN
Nome	VARCHAR(20)	NN
Cognome	VARCHAR(20)	NN
DataNascita	DATE	NN
CittàNascita	VARCHAR(20)	NN
N°Carta	VARCHAR(20)	UQ, NN
ScadenzaCarta	CHAR(16)	NN
Cvv	CHAR(16)	NN
IndirizzoConsegna	VARCHAR(30)	NN

Tabella UTENTE AMMINISTRATORE		
Colonna	**Tipo di dato**	**Attributi**[3]
CF	CHAR(16)	PK, NN
Nome	VARCHAR(20)	NN
Cognome	VARCHAR(20)	NN

Tabella OFFERTA		
Colonna	**Tipo di dato**	**Attributi**[4]
Utente	CHAR(16)	PK, NN
Oggetto	INT	PK, NN
Istante	TIMESTAMP(5)	PK, NN
Importo	FLOAT(5,2)	NN
Automatica	INT	NN
ImportoControfferta	FLOAT(5,2)	

Tabella OGGETTO ALL'ASTA		
Colonna	**Tipo di dato**	**Attributi**[5]

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

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

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

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

ID	INT	PK, NN, AI
Nome	VARCHAR(20)	NN
Descrizione	VARCHAR(30)	
Categoria	VARCHAR(20)	
Stato	VARCHAR(20)	NN
Colore	VARCHAR(16)	
Espositore	CHAR(16)	NN
Dimensioni	VARCHAR(10)	
PrezzoBase	FLOAT(7,2)	NN
N°Offerte	INT	NN
MaxOfferta	FLOAT(5,2)	
MigliorOfferente	CHAR(16)	
ImpostoControffertaMax	FLOAT(5,2)	
Scadenza	TIMESTAMP(5)	NN

Tabella CATEGORIA3		
Colonna	**Tipo di dato**	**Attributi**[6]
Nome	VARCHAR(20)	PK, NN
NomeCategoria2	VARCHAR(20)	

Tabella CATEGORIA2		
Colonna	**Tipo di dato**	**Attributi**[7]
Nome	VARCHAR(20)	PK, NN
NomeCategoria1	VARCHAR(20)	

Tabella CATEGORIA1		
Colonna	**Tipo di dato**	**Attributi**[8]
Nome	VARCHAR(20)	PK, NN

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

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

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

Indici

È stato utilizzato l'indice per la data di 'utente_base_idx' per velocizzare la ricerca delle offerte fatte da un determinato utente.

Tabella OFFERTA	
Indice utente_base_idx	**Tipo[9]: IDX**
Colonna 1	Utente

Trigger

L'utilizzo del seguente trigger per la tabella 'oggettoAllAsta' rende più veloce l'inserimento di una categoria di livello 3 da parte di un utente Amministratore in quanto non è necessario che quest'ultimo inserisca la categoria di appartenenza di un oggetto prima dell'inserimento dell'oggetto stesso all'asta ma grazie al trigger questo può verificarsi direttamente quando un utente amministratore inserisce un nuovo oggetto.
Quindi all'inserimento di un nuovo oggetto all'asta, verrà controllato se la categoria dell'oggetto è già presente nella tabella 'Categoria3'; se risulta già presente non verrà fatto nulla altrimenti verrà inserita direttamente la nuova categoria nella tabella di appartenenza.

```
CREATE TRIGGER `sistemaAsteOnline`.`oggettoAllAsta_BEFORE_INSERT` BEFORE
INSERT ON `oggettoAllAsta` FOR EACH ROW
BEGIN
    DECLARE var_categoria varchar(40);
    SELECT Nome FROM Categoria3 WHERE Nome = NEW.Categoria INTO var_categoria;
    IF var_categoria IS NULL THEN
    INSERT INTO Categoria3(Nome,NomeCategoria2) VALUES (NEW.Categoria,NULL);
    END IF;
END
```

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

Eventi

Non sono stati usati eventi nella applicazione.

Viste

Le due viste `titolario_gerarchico_lower` e `titolario_gerarchico_upper` relative alle categorie vengono utilizzaze per create un titolario gerarchico corretto.

```
CREATE VIEW `sistemaAsteOnline`.`titolario_gerarchico_lower` (Nome, Padre) AS
    SELECT c3.Nome AS Nome, c2.Nome AS Padre
    FROM (Categoria3 AS c3 LEFT JOIN Categoria2 AS c2 ON((c3.NomeCategoria2 =
    c2.Nome)))
    GROUP BY c2.Nome, c3.Nome
    UNION
    SELECT c3.Nome AS Nome, c2.Nome AS Padre
    FROM (Categoria2 AS c2 LEFT JOIN Categoria3 AS c3 ON((c3.NomeCategoria2 =
    c2.Nome)))
    GROUP BY c2.Nome, c3.Nome
```

```
CREATE VIEW `sistemaAsteOnline`.`titolario_gerarchico_upper` (Nome, Padre) AS
    SELECT c2.Nome AS Nome, c1.Nome AS Padre FROM (Categoria2 AS c2 LEFT JOIN
    Categoria1 AS c1 ON((c2.NomeCategoria1 = c1.Nome)))
    GROUP BY c1.Nome, c2.Nome
    UNION
    SELECT c2.Nome AS Nome, c1.Nome AS Padre FROM (Categoria1 AS c1 LEFT JOIN
    Categoria2 AS c2 ON((c2.NomeCategoria1 = c1.Nome)))
    GROUP BY c1.Nome, c2.Nome
```

Stored Procedures e transazioni

Per quanto riguarda le procedure implementate, queste sono chiaramente in relazione uno ad uno con le operazioni precedentemente introdotte.
Una suddivisione elementare tra le tipologie di stored procedure è data dalla definizione di:
1) Operazioni di visualizzazione sui dati
2) Operazioni complesse, caratterizzanti il corpo dell’applicazione, ovvero il ciclo di vita delle aste.

Operazioni di visualizzazione:

- visualizza_aste_attive
- visualizza_aste_per_categoria
- visualizzazione_titolario_gerarchico
- visualizza_aste_per_espositore
- visualizza_aste_per_nome_oggetto
- visualizza_oggetti_aggiudicati
- visualizza_partecipazione_aste
- visualizza_stato_asta
- report_asta

Logica delle Aste:

- inizializzazione_asta
- registra_offerta
- inserimento_categoria1
- inserimento_categoria2
- inserimento_categoria3
- aggiornamento_categoria
- cancellazione_categoria

3. Accesso

- registrazione_utente_amm
- registrazione_utente_base
- validazione_accesso

registrazione_utente_base

```
CREATE PROCEDURE `registrazione_utente_base`(in cf char(16), in nome_utente varchar(20), in cognome_utente varchar(20), in birth date, in birth_place varchar(20), in num_carta char(16), in scad_carta char(16), in cvv char(3), in indirizzo_consegna varchar(30), in psw varchar(30))
BEGIN
    INSERT INTO `utenteBase`(`CF`, `Nome`, `Cognome`, `DataNascita`, `CittaNascita`,
                `Ncarta`, `ScadenzaCarta`, `CVV`, `IndirizzoConsegna`) values (`cf`,
                `nome_utente`, `cognome_utente`, `birth`, `birth_place`, `num_carta`,
                `scad_carta`, `cvv`, `indirizzo_consegna`);
    INSERT INTO `login`(`CF`,`psw`,`user`) values (`cf`,`psw`,'base');
END
```

registrazione_utente_amm

```
CREATE PROCEDURE `registrazione_utente_amm`(in cf char(16), in nome_utente varchar(20), in cognome_utente varchar(20), in psw varchar(30))
BEGIN
    INSERT INTO `utenteAmministratore`(`CF`, `Nome`, `Cognome`) values (`cf`,
                `nome_utente`, `cognome_utente`);
    INSERT INTO `login`(`CF`,`psw`,`user`) values (`cf`,`psw`,'amministratore');
END
```

validazione_accesso

```
CREATE PROCEDURE `validazione_accesso`(in var_cf char(30), in var_psw varchar(30), out var_user INT)
BEGIN
    declare var_user_role ENUM('amministratore', 'base');
    set transaction isolation level REPEATABLE READ;

    SELECT `user` FROM `login` WHERE `CF` = var_cf AND `psw` = var_psw INTO
    var_user_role;
    IF var_user_role = 'amministratore' THEN SET var_user = 0;
    ELSEIF var_user_role = 'base' THEN SET var_user = 1;
    ELSE SET var_user = 2;
```

```
    END IF;
END
```

inizializzazione_asta

```
CREATE PROCEDURE `inizializzazione_asta`(in nome_ogg varchar(20), in cat varchar(20), in descrizione_ogg longtext, in stato_ogg varchar(30), in color_ogg varchar(20), in espositore_ogg char(20), in dim varchar(20), in prezzo_base decimal(7,2), in scad_asta int)
BEGIN
    declare exit handler for sqlexception
    BEGIN
        rollback;
        resignal;
    END;
    start transaction;

    IF nome_ogg = '' THEN
        signal sqlstate '45008'
        set message_text = "Attenzione: Non è specificato il nome dell'oggetto.";
    END IF;

    IF (`scad_asta` < 1 OR `scad_asta`> 7 ) THEN
        signal sqlstate '45008'
        set message_text = "Attenzione: L'asta puo avere durata da un minimo di 1 a un
                massimo di 7 giorni.";
    END IF;
    IF `stato_ogg` = '' THEN
        signal sqlstate '45009'
        set message_text = "Attenzione: Non è specificato lo stato dell'oggetto.";
    END IF;
    IF `cat` IS NOT NULL THEN
        INSERT INTO `oggettoAllAsta` (`Nome`, `Categoria`, `Descrizione`, `Stato`,
            `Colore`, `Espositore`, `Dimensioni`, `PrezzoBase`, `Scadenza`,
            `MAXOfferta`) values (`nome_ogg`, `cat`, `descrizione_ogg`,
            `stato_ogg`, `color_ogg`, `espositore_ogg`, `dim`,`prezzo_base`,
```

```
                    DATE_ADD(NOW(), interval scad_asta day), '0');
    END IF;
    COMMIT;
END
```

registra_offerta

```
CREATE PROCEDURE `registra_offerta`(in cf_offerente char(20), in oggetto_asta int(11), in importo_rilancio float(7,2), in importo_controfferta_max float (7,2) )
BEGIN
    declare scadenza_asta timestamp;
    declare ultimo_offerente char(20);
    declare var_importo_controff float(7,2);
    declare var_compara_offerta float(7,2);
    declare var_resto float(7,2);

    declare var_offerta_max float(5,2);
    declare var_miglior_offerente char(20);
    declare var_importo_controff_max float(7,2);
    #declare var_offerente char(20);
    #declare var_importo float(7,2);


    declare exit handler for sqlexception
    BEGIN
        rollback;
        resignal;
    END;

  set transaction isolation level SERIALIZABLE;

    start transaction;
    SELECT `Scadenza` FROM `oggettoAllAsta` WHERE `ID` = oggetto_asta INTO
    scadenza_asta;
```

```
IF scadenza_asta < current_timestamp() THEN
        signal sqlstate '45017'
        set message_text = "Asta conclusa. Oggetto aggiudicato. Non è possibile rilanciare
                            offerte!";
END IF;

#controllo sull'ultimo offerente dell'asta in questione
SELECT `MigliorOfferente` FROM `oggettoAllAsta` WHERE `ID` = `oggetto_asta`
INTO ultimo_offerente;
IF ultimo_offerente = `cf_offerente` THEN
        signal sqlstate '45016'
        set message_text = "Attenzione: non è possibile rilanciare la propria offerta!";
END IF;
IF `importo_controfferta_max` IS NULL THEN
        set var_importo_controff = 0.00;
END IF;
IF `importo_controfferta_max` IS NOT NULL THEN
        set var_importo_controff = `importo_controfferta_max`;
END IF;
IF `importo_rilancio` IS NULL THEN
        signal sqlstate '45013'
        set message_text = "Attenzione, non è stato indicato un importo di rilancio sull'asta!";
END IF;
SELECT MOD(`importo_rilancio`, 0.5) INTO var_resto;
IF var_resto <> 0 THEN
        signal sqlstate '45013'
        set message_text = "Attenzione, la granularità di incremento deve essere di 0.5
                            centesimi per ogni offerta!";
END IF;

#controllo che l'importo di rilancio sia maggiore dell'attuale offerta massima
SELECT `MaxOfferta` FROM `oggettoAllAsta` WHERE `ID` = `oggetto_asta` INTO
var_compara_offerta;
IF var_compara_offerta IS NOT NULL THEN #se esiste già sull'oggetto all'asta un'offerta
```

massima

```
IF `importo_rilancio` <= var_compara_offerta THEN
        signal sqlstate '45012'
        set message_text = "Attenzione, l'importo dell'offerta non è sufficiente a
                rilanciare l'offerta massima precedente";
END IF;
INSERT INTO `offerta`(`Utente`,`Oggetto`,`Importo`,`ImportoControfferta`) values
(`cf_offerente`,`oggetto_asta`,`importo_rilancio`, var_importo_controff);
```

#controllo che non ci sia un altro utente con un importo di controfferta maggiore dell'offerta attuale

```
SELECT `MaxOfferta`, `MigliorOfferente`, `ImportoControffertaMax` FROM
`oggettoAllAsta` WHERE `ID` = `oggetto_asta` INTO var_offerta_max,
var_miglior_offerente, var_importo_controff_max;
        IF var_importo_controff_max > `importo_controfferta_max` + 0.5 AND
        var_importo_controff_max > `importo_rilancio`- var_offerta_max AND
        var_importo_controff_max IS NOT NULL THEN
        INSERT INTO `offerta`(`Utente`, `Oggetto`, `Importo`, `Automatica`,
        `ImportoControfferta`) values (var_miglior_offerente, `oggetto_asta`,
        `importo_rilancio` +0.5, '1', var_importo_controff_max-((`importo_rilancio`-
        var_offerta_max)+0.5));
        UPDATE `oggettoAllAsta` set `NOfferte` = `NOfferte` + 2, `Maxofferta` =
        `Maxofferta` + (`importo_rilancio` - var_offerta_max)+0.5,
        `ImportoControffertaMax` = var_importo_controff_max-((`importo_rilancio`-
        var_offerta_max)+0.5)
        WHERE `ID` = `oggetto_asta`;
        ELSE
        UPDATE `oggettoAllAsta` set `NOfferte` = `NOfferte` + 1, `Maxofferta` =
        `importo_rilancio`, `MigliorOfferente` = `cf_offerente`,
        `ImportoControffertaMax` = var_importo_controff
        WHERE `ID` = `oggetto_asta`;
        END IF;
```

```
        END IF;

        IF var_compara_offerta IS NULL THEN #se questa è la prima offerta sull'oggetto all'asta

    #controllo che l'importo di rilancio non sia nullo e che sia maggiore dell'attuale offerta massima
                SELECT `PrezzoBase` FROM `oggettoAllAsta` WHERE `ID` = `oggetto_asta`
                INTO var_compara_offerta;
                IF `importo_rilancio` <= var_compara_offerta THEN
                        signal sqlstate '45012'
                        set message_text = "Attenzione, l'importo dell'offerta non è sufficiente a
                                        rilanciare l'offerta massima precedente";
                END IF;
                INSERT INTO `offerta`(`Utente`,`Oggetto`,`Importo`,`ImportoControfferta`) values
                (`cf_offerente`,`oggetto_asta`,`importo_rilancio`,var_importo_controff);
                UPDATE `oggettoAllAsta` set `NOfferte` = `NOfferte` + 1, `Maxofferta` =
                `importo_rilancio`, `MigliorOfferente` = `cf_offerente`, `ImportoControffertaMax` =
                var_importo_controff
                WHERE `ID` = `oggetto_asta`;
        END IF;
        COMMIT;
END
```

inserimento_categoria1

```
CREATE PROCEDURE `inserimento_categoria1`(in nome_categoria varchar(20))
BEGIN
        declare var_compara_nome varchar(20);
        declare exit handler for sqlexception
        BEGIN
                rollback;
                resignal;
        END;
        start transaction;
```

```
        IF nome_categoria = '' THEN
                signal sqlstate '45001'
                set message_text = "Categoria non specificata.";
        END IF;

    # controlla se esiste gia la categoria
        SELECT `Nome` FROM `Categoria1` WHERE `Nome` = `nome_categoria` INTO
        var_compara_nome;
        IF var_compara_nome IS NOT NULL THEN
                signal sqlstate '45001'
                set message_text = "Categoria gia esistente.";
        END IF;
        INSERT INTO `Categoria1`(`Nome`) values (`nome_categoria`);
        COMMIT;
END
```

inserimento_categoria2

```
CREATE PROCEDURE `inserimento_categoria2`(in nome_categoria varchar(20), in
padre_categoria varchar(20))
BEGIN
        declare var_compara_nome varchar(20);
        declare var_compara_nome_padre varchar(20);
        declare exit handler for sqlexception

        BEGIN
                rollback;
                resignal;
        END;
        start transaction;

        IF nome_categoria = '' OR padre_categoria = '' THEN
                signal sqlstate '45001'
                set message_text = "Categoria/e non specificata/e.";
```

```
        END IF;

        # controlla se esiste gia la categoria
        SELECT `Nome` FROM `Categoria2` WHERE `Nome` = `nome_categoria` INTO
        var_compara_nome;

        IF var_compara_nome IS NOT NULL THEN
                signal sqlstate '45001'
                set message_text = "Categoria gia esistente.";
        END IF;
        IF `padre_categoria` IS NULL THEN
                INSERT INTO `Categoria2`(`Nome`) values (`nome_categoria`);
        END IF;
        IF `padre_categoria` IS NOT NULL THEN
                SELECT `Nome` FROM `Categoria1` WHERE `Nome` = `padre_categoria` INTO
                var_compara_nome_padre;

        # se non esiste questa categoria-padre viene mostrato un messaggio di errore
        IF var_compara_nome_padre IS NULL THEN
                signal sqlstate '45007'
                set message_text = "Attenzione, non esiste la categoria padre indicata al livello 1 del
                                titolario!";
        END IF;
        INSERT INTO `Categoria2`(`Nome`, `NomeCategoria1`) values  (`nome_categoria`,
        `padre_categoria`);
        END IF;
        COMMIT;
END
```

inserimento_categoria3

```
CREATE PROCEDURE `inserimento_categoria3` (in nome_categoria varchar(20), in
padre_categoria varchar(20))
BEGIN
```

```
declare var_compara_nome varchar(20);
declare var_compara_nome_padre varchar(20);
declare exit handler for sqlexception

BEGIN
        rollback;
        resignal;
END;
start transaction;

IF nome_categoria = '' OR padre_categoria = '' THEN
        signal sqlstate '45001'
        set message_text = "Categoria/e non specificata/e.";
END IF;

#controlla se la categoria è già esistente
SELECT `Nome` FROM `Categoria3` WHERE `Nome` = `nome_categoria` INTO
var_compara_nome;
IF var_compara_nome IS NOT NULL THEN
        signal sqlstate '45001'
        set message_text = "Categoria gia esistente.";
END IF;
IF `padre_categoria` IS NULL THEN
        INSERT INTO `Categoria3`(`Nome`) values (`nome_categoria`);
END IF;
IF `padre_categoria` IS NOT NULL THEN
        SELECT `Nome` FROM `Categoria2` WHERE `Nome` = `padre_categoria` INTO
        var_compara_nome_padre;

# se non esiste questa categoria-padre viene mostrato un messaggio di errore
        IF var_compara_nome_padre IS NULL THEN
                signal sqlstate '45007'
                set message_text = "Attenzione, non esiste la categoria padre indicata al
                                livello 2 del titolario!";
```

```
            END IF;
            INSERT INTO `Categoria3` (`Nome`, `NomeCategoria2`) values (`nome_categoria`,
            `padre_categoria`);
        END IF;
        COMMIT;
END
```

aggiornamento_categoria

```
CREATE PROCEDURE `aggiornamento_categoria`(in old_nome_categoria varchar(20), in
new_nome_categoria varchar(20), in old_nome_padre varchar(20), in new_nome_padre varchar (20)
)
BEGIN
        declare var_compara_nome varchar(20);
        declare var_compara_nomepadre varchar(20);
        declare exit handler for sqlexception
        BEGIN
            rollback;
            resignal;
        END;

        start transaction;

        # check sulle variabili di ingresso indispensabili all'esecuzione della procedura
        IF `old_nome_categoria` = '' OR  `old_nome_categoria` IS NULL THEN
            signal sqlstate '45005'
            set message_text = "E' necessario fornire il nome della categoria per cercarla nel
                                titolario.";
        END IF;

        #verifica l'esistenza della categoria da modificare, cercandola nei tre livelli del titolario.
        #LIVELLO 1, RICERCA.
        SELECT `Nome` FROM `Categoria1` WHERE `Nome` = `old_nome_categoria` INTO
        var_compara_nome;
```

```
IF var_compara_nome IS NULL THEN # chiave non trovata al primo livello!
    #LIVELLO2, RICERCA.
    SELECT `Nome` FROM `Categoria2` WHERE `Nome` = `old_nome_categoria`
    INTO var_compara_nome;
    IF var_compara_nome IS NULL THEN #chiave non trovata al secondo livello!
        #LIVELLO3, RICERCA.
        SELECT `Nome` FROM `Categoria3` WHERE `Nome` =
        `old_nome_categoria` INTO var_compara_nome;
        IF var_compara_nome IS NULL THEN #chiave non presente nel titolario!
            signal sqlstate '45002'
            set message_text = "Questa categoria non esiste.";
        END IF;

        IF var_compara_nome IS NOT NULL THEN #chiave trovata al terzo livello.
            IF `new_nome_padre` IS NULL OR `new_nome_padre` = '' THEN
                IF `new_nome_categoria` = '' OR `new_nome_categoria` IS
                NULL THEN
                    signal sqlstate '45004'
                    set message_text = "Non si può aggiornare a valore
                                        nullo il nome di una categoria.";
                END IF;

                #si vuole aggiornare solo il nome della categoria.
                UPDATE `Categoria3` set `Nome` = `new_nome_categoria`
                WHERE `Nome` = `old_nome_categoria`;
            END IF;

            IF `new_nome_padre` IS NOT NULL THEN

                #se non esiste la categoria di livello 2,ERRORE
                SELECT `Nome` FROM `Categoria2` WHERE
                `Nome`=`new_nome_padre` INTO var_compara_nomepadre;
                IF var_compara_nomepadre IS NULL THEN
                    signal sqlstate '45007'
```

```
                    set message_text = "Attenzione, non esiste la categoria
                                        padre indicata al livello 2 del
                                        titolario!";
                END IF;

                IF `new_nome_categoria` IS NULL THEN
                    IF `old_nome_padre` IS NULL THEN
                        UPDATE `Categoria3` set `NomeCategoria2` =
                        `new_nome_padre` WHERE `Nome` =
                        `old_nome_categoria`;
                    END IF;
                    IF `old_nome_padre` IS NOT NULL THEN
                        UPDATE `Categoria3` set `NomeCategoria2` =
                        `new_nome_padre` WHERE `Nome` =
                        `old_nome_categoria` AND `NomeCategoria2`
                        = `old_nome_padre`;
                    END IF;
                END IF;
            END IF;
            set var_compara_nome = null;
        END IF;
    END IF;
    IF var_compara_nome IS NOT NULL THEN #chiave trovata al secondo livello!
        IF `new_nome_padre` IS NULL OR `new_nome_padre` = '' THEN
            IF `new_nome_categoria` IS NULL OR `new_nome_categoria` = ''
            THEN
                signal sqlstate '45004'
                set message_text = "Non si può aggiornare a valore nullo il
                                    nome di una categoria.";
            END IF;

            #si vuole aggiornare solo il nome della categoria.
            UPDATE `Categoria2` set `Nome` = `new_nome_categoria` WHERE
            `Nome` = `old_nome_categoria`;
```

```
                END IF;
                IF `new_nome_padre` IS NOT NULL THEN

                        #se non esiste la categoria di livello 1, ERRORE
                        SELECT `Nome` FROM `Categoria1` WHERE
                        `Nome`=`new_nome_padre` INTO var_compara_nomepadre;
                        IF var_compara_nomepadre IS NULL THEN
                                signal sqlstate '45007'
                                set message_text = "Attenzione, non esiste la categoria padre
                                                indicata al livello 1 del titolario!";
                        END IF;
                        IF `new_nome_categoria` IS NULL THEN
                                IF `old_nome_padre` IS NULL THEN
                                        UPDATE `Categoria2` set `NomeCategoria1` =
                                        `new_nome_padre` WHERE `Nome` =
                                        `old_nome_categoria`;
                                END IF;
                                IF `old_nome_padre` IS NOT NULL THEN
                                        UPDATE `Categoria2` set `NomeCategoria1` =
                                        `new_nome_padre` WHERE `Nome` =
                                        `old_nome_categoria` AND `NomeCategoria1` =
                                        `old_nome_padre`;
                                END IF;
                        END IF;
                END IF;
                set var_compara_nome = null;
        END IF;
END IF;
IF var_compara_nome IS NOT NULL THEN #chiave trovata al primo livello!
        IF `new_nome_categoria` IS NULL OR `new_nome_categoria` = '' THEN
                signal sqlstate '45004'
                set message_text = "Non si può aggiornare a valore nullo il nome di una
                                categoria.";
        END IF;
```

```
            UPDATE `Categoria1` set `Nome` = `new_nome_categoria` WHERE `Nome` =
            `old_nome_categoria`;
        END IF;
        COMMIT;

END
```

cancellazione_categoria

```
CREATE PROCEDURE `cancellazione_categoria`(in nome_categoria varchar(20))
BEGIN
        declare var_compara_nome varchar(20);
        declare exit handler for sqlexception
        BEGIN
            rollback;
            resignal;
        END;
        start transaction;
        SELECT `Categoria` FROM `oggettoAllAsta` WHERE `Categoria` = `nome_categoria`
        AND  `Scadenza` > current_timestamp() INTO var_compara_nome;
        IF var_compara_nome IS NOT NULL THEN
            signal sqlstate '45002'
            set message_text = "Non si può eliminare una categoria a cui afferisce un oggetto
                                correntemente all'asta.";
        END IF;
        SELECT `Nome` FROM `Categoria1` WHERE `Nome` = `nome_categoria` INTO
        var_compara_nome;
        IF var_compara_nome IS NULL THEN
            SELECT `Nome` FROM `Categoria2` WHERE `Nome` = `nome_categoria` INTO
            var_compara_nome;
            IF var_compara_nome IS NULL THEN
                SELECT `Nome` FROM `Categoria3` WHERE `Nome` = `nome_categoria`
                INTO var_compara_nome;
```

```
                IF var_compara_nome IS NULL THEN
                    signal sqlstate '45002'
                    set message_text = "Questa categoria non esiste.";
                END IF;
                IF var_compara_nome IS NOT NULL THEN
                    DELETE FROM `Categoria3` WHERE `Nome` = `nome_categoria`;
                END IF;
            END IF;
            IF var_compara_nome IS NOT NULL THEN
                DELETE FROM `Categoria2` WHERE `Nome` = `nome_categoria`;
            END IF;
        END IF;
        IF var_compara_nome IS NOT NULL THEN
            DELETE FROM `Categoria1` WHERE `Nome` = `nome_categoria`;
        END IF;
        COMMIT;
END
```

visualizza_aste_attive

```
CREATE PROCEDURE `visualizza_aste_attive`()
BEGIN

    set transaction READ ONLY;
    set transaction isolation level READ COMMITTED;

    SELECT `ID`, `Nome`, `Espositore`, `MAXOfferta` FROM `oggettoAllAsta` WHERE
    (Scadenza > current_timestamp());
END
```

visualizza_aste_per_categoria

```
CREATE PROCEDURE `visualizza_aste_per_categoria` (in categoria varchar(20))
BEGIN
```

```
    declare var_categoria varchar(20);
    set var_categoria = concat( "%", categoria, "%");

    set transaction READ ONLY;
    set transaction isolation level READ COMMITTED;

    SELECT `ID`, aste.`Nome`, Categoria, `MaxOfferta` FROM `oggettoAllAsta` AS aste
    LEFT JOIN `Categoria3` AS cat3 ON aste.`Categoria` = cat3.`Nome`
    LEFT JOIN `Categoria2` AS cat2 ON cat3.`NomeCategoria2` = cat2.`Nome`
    LEFT JOIN `Categoria1` AS cat1 ON cat2.`NomeCategoria1` = cat1.`Nome`
    WHERE (Scadenza > current_timestamp()) AND (cat3.`Nome` like var_categoria OR
    cat2.`Nome` like var_categoria OR cat1.`Nome` like var_categoria);
END
```

visualizzazione_titolario_gerarchico

```
CREATE PROCEDURE `visualizzazione_titolario_gerarchico`()
BEGIN

    set transaction READ ONLY;
    set transaction isolation level READ COMMITTED;

    SELECT t_low.`Nome` AS cat3, t_low.`Padre` AS cat2, t_up.`Padre` AS cat1
    FROM `titolario_gerarchico_lower` AS t_low LEFT JOIN `titolario_gerarchico_upper` AS
    t_up
    ON t_low.`Padre` = t_up.`Nome`
    GROUP BY t_up.`Padre`, t_low.`Padre`, t_low.`Nome`
    UNION
    SELECT t_low.`Nome` AS cat3, t_low.`Padre` AS cat2, t_up.`Padre` AS cat1
    FROM `titolario_gerarchico_lower` AS t_low RIGHT JOIN `titolario_gerarchico_upper`
    AS t_up
    ON t_low.`Padre` = t_up.`Nome`
    GROUP BY t_up.`Padre`, t_low.`Padre`, t_low.`Nome`;
```

```
END
```

visualizza_aste_per_espositore

```
CREATE PROCEDURE `visualizza_aste_per_espositore`(in var_espositore char(20))
BEGIN

    set transaction READ ONLY;
    set transaction isolation level READ COMMITTED;

    SELECT `Espositore`, `ID`, `Nome`, `Categoria`, `MaxOfferta`, `Scadenza`
    FROM `oggettoAllAsta`
    WHERE `Espositore` = var_espositore
    ORDER BY `ID` ASC;
END
```

visualizza_aste_per_nome_oggetto

```
CREATE PROCEDURE `visualizza_aste_per_nome_oggetto`(in nomeoggetto varchar(20))
BEGIN

    declare var_nomeoggetto varchar(20);
    set var_nomeoggetto = concat( "%", nomeoggetto, "%");

     set transaction READ ONLY;
    set transaction isolation level READ COMMITTED;

    SELECT `ID`, `Nome`, `Espositore`, `MAXOfferta`
    FROM `oggettoAllAsta`
    WHERE (`Scadenza` > current_timestamp()) AND `Nome` like  var_nomeoggetto;
END
```

visualizza_oggetti_aggiudicati

```
CREATE PROCEDURE `visualizza_oggetti_aggiudicati`(in utente char(16))
BEGIN

    set transaction READ ONLY;
    set transaction isolation level READ COMMITTED;

    SELECT `Nome`, `Descrizione`, `MaxOfferta`, `Scadenza`
    FROM `oggettoAllAsta`
    WHERE `MigliorOfferente` = utente AND `Scadenza` < current_timestamp();
END
```

visualizza_partecipazione_aste

```
CREATE PROCEDURE `visualizza_partecipazione_aste`(in utente char(16))
BEGIN
    set transaction READ ONLY;
    set transaction isolation level READ COMMITTED;

    SELECT DISTINCT `ID`, `Nome`, `Descrizione`, `Scadenza`
    FROM`offerta` USE INDEX (utente_base_idx)
    JOIN `oggettoAllAsta` ON `Oggetto` = `ID`
    WHERE `Utente` = `utente`;
END
```

visualizza_stato_asta

```
CREATE PROCEDURE `visualizza_stato_asta`(in asta_id int)
BEGIN
    set transaction READ ONLY;
    set transaction isolation level READ COMMITTED;

    SELECT `Espositore`, `Nome`, `Descrizione`, `Categoria`, `Stato`, `Colore`, `Dimensioni`,
    `NOfferte`, `MAXOfferta`, `Scadenza`
    FROM `oggettoAllAsta`
```

```
    WHERE (`ID` = asta_id);
END
```

report_asta

```
CREATE PROCEDURE `report_asta`(in id_asta int)
BEGIN

    set transaction READ ONLY;
    set transaction isolation level READ COMMITTED;

    SELECT `Utente`, `Importo`, `Istante`, `Automatica`
    FROM `oggettoAllAsta` JOIN `offerta` ON `Oggetto` = `ID`
    WHERE`Oggetto` = id_asta
    ORDER BY `Importo` DESC;
END
```

Appendice: Implementazione

Codice SQL per instanziare il database

```
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 sistemaAsteOnline
-- -----------------------------------------------------
DROP SCHEMA IF EXISTS `sistemaAsteOnline` ;

-- -----------------------------------------------------
-- SCHEMA sistemaAsteOnline
-- -----------------------------------------------------
CREATE SCHEMA IF NOT EXISTS `sistemaAsteOnline` ;
USE `sistemaAsteOnline` ;

-- -----------------------------------------------------
-- TABLE `sistemaAsteOnline`.`Categoria1`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `sistemaAsteOnline`.`Categoria1` ;

CREATE TABLE IF NOT EXISTS `sistemaAsteOnline`.`Categoria1` (
  `Nome` VARCHAR(20) CHARACTER SET 'utf8mb4' COLLATE 'utf8mb4_0900_ai_ci' NOT NULL,
  PRIMARY KEY (`Nome`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
```

```
COLLATE = utf8mb4_0900_ai_ci;


-- -----------------------------------------------------
-- TABLE `sistemaAsteOnline`.`Categoria2`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `sistemaAsteOnline`.`Categoria2` ;

CREATE TABLE IF NOT EXISTS `sistemaAsteOnline`.`Categoria2` (
  `Nome` VARCHAR(20) CHARACTER SET 'utf8mb4' COLLATE 'utf8mb4_0900_ai_ci' NOT NULL,
    `NomeCategoria1` VARCHAR(20) CHARACTER SET 'utf8mb4' COLLATE 'utf8mb4_0900_ai_ci' NULL DEFAULT NULL,
 PRIMARY KEY (`Nome`),
 CONSTRAINT `fk_Categoria2_Categoria1`
   FOREIGN KEY (`NomeCategoria1`)
   REFERENCES `sistemaAsteOnline`.`Categoria1` (`Nome`)
   ON DELETE SET NULL
   ON UPDATE NO ACTION)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;

CREATE INDEX `fk_Categoria2_Categoria1_idx` ON `sistemaAsteOnline`.`Categoria2` (`NomeCategoria1` ASC) VISIBLE;


-- -----------------------------------------------------
-- TABLE `sistemaAsteOnline`.`Categoria3`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `sistemaAsteOnline`.`Categoria3` ;

CREATE TABLE IF NOT EXISTS `sistemaAsteOnline`.`Categoria3` (
  `Nome` VARCHAR(20) CHARACTER SET 'utf8mb4' COLLATE 'utf8mb4_0900_ai_ci' NOT
```

```
NULL,
        `NomeCategoria2` VARCHAR(20) CHARACTER SET 'utf8mb4' COLLATE 'utf8mb4_0900_ai_ci' NULL DEFAULT NULL,
  PRIMARY KEY (`Nome`),
  CONSTRAINT `fk_Categoria3_Categoria2`
    FOREIGN KEY (`NomeCategoria2`)
    REFERENCES `sistemaAsteOnline`.`Categoria2` (`Nome`)
    ON DELETE SET NULL
    ON UPDATE CASCADE)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;

CREATE INDEX `fk_Categoria3_Categoria2_idx` ON `sistemaAsteOnline`.`Categoria3` (`NomeCategoria2` ASC) VISIBLE;



-- -----------------------------------------------------
-- TABLE `sistemaAsteOnline`.`login`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `sistemaAsteOnline`.`login` ;

CREATE TABLE IF NOT EXISTS `sistemaAsteOnline`.`login` (
  `CF` CHAR(16) CHARACTER SET 'utf8mb4' COLLATE 'utf8mb4_0900_ai_ci' NOT NULL,
    `psw` VARCHAR(20) CHARACTER SET 'utf8mb4' COLLATE 'utf8mb4_0900_ai_ci' NOT NULL,
        `user` ENUM('amministratore', 'base') CHARACTER SET 'utf8mb4' COLLATE 'utf8mb4_0900_ai_ci' NOT NULL,
  PRIMARY KEY (`CF`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;
```

```
-- -----------------------------------------------------
-- TABLE `sistemaAsteOnline`.`utenteAmministratore`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `sistemaAsteOnline`.`utenteAmministratore` ;

CREATE TABLE IF NOT EXISTS `sistemaAsteOnline`.`utenteAmministratore` (
  `CF` CHAR(16) CHARACTER SET 'utf8mb4' COLLATE 'utf8mb4_0900_ai_ci' NOT NULL,
  `Nome` VARCHAR(20) CHARACTER SET 'utf8mb3' NOT NULL,
  `Cognome` VARCHAR(20) CHARACTER SET 'utf8mb3' NOT NULL,
  PRIMARY KEY (`CF`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;


-- -----------------------------------------------------
-- TABLE `sistemaAsteOnline`.`oggettoAllAsta`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `sistemaAsteOnline`.`oggettoAllAsta` ;

CREATE TABLE IF NOT EXISTS `sistemaAsteOnline`.`oggettoAllAsta` (
  `ID` INT NOT NULL AUTO_INCREMENT,
   `Descrizione` VARCHAR(30) CHARACTER SET 'utf8mb4' COLLATE 'utf8mb4_0900_ai_ci' NULL DEFAULT NULL,
   `Nome` VARCHAR(20) CHARACTER SET 'utf8mb4' COLLATE 'utf8mb4_0900_ai_ci' NOT NULL,
    `Categoria` VARCHAR(20) CHARACTER SET 'utf8mb4' COLLATE 'utf8mb4_0900_ai_ci' NULL DEFAULT NULL,
   `Stato` VARCHAR(20) CHARACTER SET 'utf8mb4' COLLATE 'utf8mb4_0900_ai_ci' NOT NULL,
  `Colore` VARCHAR(16) CHARACTER SET 'utf8mb4' COLLATE 'utf8mb4_0900_ai_ci' NULL DEFAULT NULL,
   `Espositore` CHAR(16) CHARACTER SET 'utf8mb4' COLLATE 'utf8mb4_0900_ai_ci' NOT NULL,
```

```
  `Dimensioni` VARCHAR(10) CHARACTER SET 'utf8mb4' COLLATE 'utf8mb4_0900_ai_ci' NULL DEFAULT NULL,
  `PrezzoBase` FLOAT(7,2) NOT NULL,
  `NOfferte` INT NOT NULL DEFAULT '0',
  `MaxOfferta` FLOAT(5,2) NULL DEFAULT 0.00,
  `MigliorOfferente` CHAR(16) CHARACTER SET 'utf8mb4' COLLATE 'utf8mb4_0900_ai_ci' NULL DEFAULT NULL,
  `ImportoControffertaMax` FLOAT(5,2) NULL DEFAULT '0.00',
  `Scadenza` TIMESTAMP(5) NOT NULL,
  PRIMARY KEY (`ID`),
  CONSTRAINT `fk_Oggetto all'Asta_Categoria_3`
    FOREIGN KEY (`Categoria`)
    REFERENCES `sistemaAsteOnline`.`Categoria3` (`Nome`)
    ON DELETE SET NULL
    ON UPDATE CASCADE,
  CONSTRAINT `fk_OggettoAllAsta_Espositore`
    FOREIGN KEY (`Espositore`)
    REFERENCES `sistemaAsteOnline`.`utenteAmministratore` (`CF`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;

CREATE INDEX `fk_categoria_3_idx` ON `sistemaAsteOnline`.`oggettoAllAsta` (`Categoria` ASC) VISIBLE;

CREATE INDEX `fk_oggettoAllAsta_Espositore_1_idx` ON `sistemaAsteOnline`.`oggettoAllAsta` (`Espositore` ASC) VISIBLE;


-- -----------------------------------------------------
-- TABLE `sistemaAsteOnline`.`utenteBase`
-- -----------------------------------------------------
```

```
DROP TABLE IF EXISTS `sistemaAsteOnline`.`utenteBase` ;

CREATE TABLE IF NOT EXISTS `sistemaAsteOnline`.`utenteBase` (
  `CF` CHAR(16) CHARACTER SET 'utf8mb4' COLLATE 'utf8mb4_0900_ai_ci' NOT NULL,
   `Nome` VARCHAR(20) CHARACTER SET 'utf8mb4' COLLATE 'utf8mb4_0900_ai_ci' NOT NULL,
  `Cognome` VARCHAR(20) CHARACTER SET 'utf8mb4' COLLATE 'utf8mb4_0900_ai_ci' NOT NULL,
  `DataNascita` DATE NOT NULL,
   `CittaNascita` VARCHAR(20) CHARACTER SET 'utf8mb4' COLLATE 'utf8mb4_0900_ai_ci' NOT NULL,
   `NCarta` VARCHAR(20) CHARACTER SET 'utf8mb4' COLLATE 'utf8mb4_0900_ai_ci' NOT NULL,
  `ScadenzaCarta` CHAR(16) CHARACTER SET 'utf8mb4' COLLATE 'utf8mb4_0900_ai_ci' NOT NULL,
  `Cvv` CHAR(16) CHARACTER SET 'utf8mb4' COLLATE 'utf8mb4_0900_ai_ci' NOT NULL,
        `IndirizzoConsegna` VARCHAR(30) CHARACTER SET 'utf8mb4' COLLATE 'utf8mb4_0900_ai_ci' NOT NULL,
  PRIMARY KEY (`CF`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;


-- -----------------------------------------------------
-- TABLE `sistemaAsteOnline`.`offerta`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `sistemaAsteOnline`.`offerta` ;

CREATE TABLE IF NOT EXISTS `sistemaAsteOnline`.`offerta` (
  `Utente` CHAR(16) CHARACTER SET 'utf8mb4' COLLATE 'utf8mb4_0900_ai_ci' NOT NULL,
  `Oggetto` INT NOT NULL,
   `Istante` TIMESTAMP(5) NOT NULL DEFAULT CURRENT_TIMESTAMP(5) ON UPDATE CURRENT_TIMESTAMP(5),
```

```
  `Importo` FLOAT(5,2) NOT NULL,
  `Automatica` INT NOT NULL DEFAULT '0',
  `ImportoControfferta` FLOAT(5,2) NULL DEFAULT NULL,
  PRIMARY KEY (`Utente`, `Oggetto`, `Istante`),
  CONSTRAINT `fk_oggetto_1`
    FOREIGN KEY (`Oggetto`)
    REFERENCES `sistemaAsteOnline`.`oggettoAllAsta` (`ID`)
    ON DELETE RESTRICT
    ON UPDATE RESTRICT,
  CONSTRAINT `fk_utente_1`
    FOREIGN KEY (`Utente`)
    REFERENCES `sistemaAsteOnline`.`utenteBase` (`CF`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci
KEY_BLOCK_SIZE = 1;

CREATE INDEX `utente_base_idx` ON `sistemaAsteOnline`.`offerta` (`Utente` ASC) VISIBLE;

CREATE INDEX `oggetto_idx` ON `sistemaAsteOnline`.`offerta` (`Oggetto` ASC) VISIBLE;

SET SQL_MODE = '';
DROP USER IF EXISTS utentebase;
SET
SQL_MODE='ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,N
O_ZERO_DATE,ERROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION';
CREATE USER 'utentebase' IDENTIFIED BY 'Utentebase!0';

GRANT EXECUTE ON procedure `sistemaAsteOnline`.`registra_offerta` TO 'utentebase';
GRANT EXECUTE ON procedure `sistemaAsteOnline`.`visualizza_aste_attive` TO 'utentebase';
GRANT EXECUTE ON procedure `sistemaAsteOnline`.`visualizza_aste_per_categoria` TO
'utentebase';
```

```
GRANT EXECUTE ON procedure `sistemaAsteOnline`.`visualizza_aste_per_espositore` TO 'utentebase';
GRANT EXECUTE ON procedure `sistemaAsteOnline`.`visualizza_aste_per_nome_oggetto` TO 'utentebase';
GRANT EXECUTE ON procedure `sistemaAsteOnline`.`visualizza_stato_asta` TO 'utentebase';
GRANT EXECUTE ON procedure `sistemaAsteOnline`.`visualizza_oggetti_aggiudicati` TO 'utentebase';
GRANT EXECUTE ON procedure `sistemaAsteOnline`.`visualizza_partecipazione_aste` TO 'utentebase';
GRANT EXECUTE ON procedure `sistemaAsteOnline`.`registrazione_utente_base` TO 'utentebase';
GRANT EXECUTE ON procedure `sistemaAsteOnline`.`visualizzazione_titolario_gerarchico` TO 'utentebase';
SET SQL_MODE = '';
DROP USER IF EXISTS utenteamm;
SET SQL_MODE='ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZERO_DATE,ERROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION';
CREATE USER 'utenteamm' IDENTIFIED BY 'Utenteamm!0';

GRANT INSERT ON TABLE `sistemaAsteOnline`.`oggettoAllAsta` TO 'utenteamm';
GRANT EXECUTE ON procedure `sistemaAsteOnline`.`aggiornamento_categoria` TO 'utenteamm';
GRANT EXECUTE ON procedure `sistemaAsteOnline`.`cancellazione_categoria` TO 'utenteamm';
GRANT EXECUTE ON procedure `sistemaAsteOnline`.`inizializzazione_asta` TO 'utenteamm';
GRANT EXECUTE ON procedure `sistemaAsteOnline`.`inserimento_categoria1` TO 'utenteamm';
GRANT EXECUTE ON procedure `sistemaAsteOnline`.`inserimento_categoria2` TO 'utenteamm';
GRANT EXECUTE ON procedure `sistemaAsteOnline`.`inserimento_categoria3` TO 'utenteamm';
GRANT EXECUTE ON procedure `sistemaAsteOnline`.`report_asta` TO 'utenteamm';
GRANT EXECUTE ON procedure `sistemaAsteOnline`.`visualizzazione_titolario_gerarchico` TO 'utenteamm';
GRANT EXECUTE ON procedure `sistemaAsteOnline`.`visualizza_aste_per_espositore` TO 'utenteamm';
GRANT EXECUTE ON procedure `sistemaAsteOnline`.`visualizza_stato_asta` TO 'utenteamm';
```

```
GRANT EXECUTE ON procedure `sistemaAsteOnline`.`registrazione_utente_amm` TO 'utenteamm';
GRANT UPDATE, SELECT, INSERT, DELETE ON TABLE `sistemaAsteOnline`.`Categoria1` TO 'utenteamm';
GRANT UPDATE, SELECT, INSERT, DELETE ON TABLE `sistemaAsteOnline`.`Categoria2` TO 'utenteamm';
GRANT DELETE, INSERT, SELECT, UPDATE ON TABLE `sistemaAsteOnline`.`Categoria3` TO 'utenteamm';
SET SQL_MODE = '';
DROP USER IF EXISTS login;
SET
SQL_MODE='ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZERO_DATE,ERROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION';
CREATE USER 'login' IDENTIFIED BY 'Login!00';

GRANT EXECUTE ON procedure `sistemaAsteOnline`.`validazione_accesso` TO 'login';

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

Codice del Front-End

main.c

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

#include "defines.h"
```

```
#define LOGIN "login"

static MYSQL *con;

typedef enum {
	ADMIN = 1,
	STANDARD,
	FAILED_LOGIN
}role_t;

role_t attempt_login(MYSQL *conn, char *cf, char *password){

	MYSQL_STMT *login_procedure;
	MYSQL_BIND param[3];

	int role = 0;

	if(!setup_prepared_stmt(&login_procedure, "call validazione_accesso(?, ?, ?)", conn)){
		print_stmt_error(login_procedure, "Unable to initialize login statement\n");
		goto err2;
	}

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

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

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

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

```
        print_stmt_error(login_procedure, "Could not bind parameters for login");
        goto err;
}

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

//Prepare output parameters
memset(param, 0, sizeof(param));
param[0].buffer_type = MYSQL_TYPE_LONG;  //OUT
param[0].buffer = &role;
param[0].buffer_length = sizeof(role);


if(mysql_stmt_bind_result(login_procedure, param)){
        print_stmt_error(login_procedure, "Could not retrieve output parameters");
        goto err;
}

//Retrieve output parameter
if(mysql_stmt_fetch(login_procedure)){
        print_stmt_error(login_procedure, "Could not buffer results");
        goto err;
}

mysql_stmt_close(login_procedure);
return role;

err:
mysql_stmt_close(login_procedure);
```

```
        err2:
        return FAILED_LOGIN;

}

void main(int argc, char **argv){

        char operation[20];
        char cf[128];
        char password[128];

        top:

        printf("\033[2J\033[H");
        printf("..........\n");
        printf("..........\n");
        printf(".....__ __ __ __ __ __ ..... _ __ __ _ .....\n");
        printf(".....|              |.....|        |.....\n");
        printf(".....| LOGIN UTENTE: 1 |.....| EXIT: 0 |.....\n");
        printf(".....|_ __ __ __ __ __ |.....|__ __ __ |.....\n");
        printf("..........\n");
        printf("..........\n\n");
        printf("   Inserisci il codice dell'operazione : ");

        getInput(64, operation, false);

        if(!strcmp(operation, "0")){
                printf(ANSI_COLOR_RED"\n   Exit...\n\n" ANSI_COLOR_RESET);
                return;
        }
        else if(!strcmp(operation, "1")){

                //Inizializzazione della prima connessione (phantom user), per connettersi al db ->
                //tabella Login.
```

```
con = mysql_init(NULL);
if (con == NULL) {
        fprintf(stderr, "Initilization error: %s\n", mysql_error(con));
        exit(EXIT_FAILURE);
}

if (mysql_real_connect(con, getJson(LOGIN, HOST), getJson(LOGIN,
USERNAME), getJson(LOGIN, PASSWORD), getJson(LOGIN, DATABASE),
atoi(getJson(LOGIN, PORT)), NULL,
 CLIENT_MULTI_STATEMENTS | CLIENT_MULTI_RESULTS) ==
NULL){
        fprintf(stderr, "Connection error: %s\n", mysql_error(con));
        mysql_close(con);
        exit(1);
}

printf("\033[2J\033[H");
printf(".....\n");
printf(".....\n");
printf(".....__ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ _.....\
n");
printf(".....|                                                  |.....\n");
printf(".....|              LOGIN UTENTE                        |.....\n");
printf(".....|__ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ _|.....\
n");
printf(".....\n");
printf(".....\n\n");
printf("                        CODICE FISCALE : ");
getInput(128, cf, false);
printf("\n\n                         PASSWORD : ");
getInput(128, password, true);

int role = attempt_login(con, cf, password);
```

```
            switch(role){
                case 0:
                    admin(con, cf);
                    break;
                case 1:
                    standard(con, cf);
                    break;
                default:
                    printf(ANSI_COLOR_RED"\n\n                    Credenziali
                    errate!\n\n\n" ANSI_COLOR_RESET);
                    break;
            }

            mysql_close(con);
        }
        else{
            printf(ANSI_COLOR_RED"\n   Comando non riconosciuto\n\n"
            ANSI_COLOR_RESET);
        }

        goto top;
}
```

standard.c

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

#include "defines.h"

#define STANDARD "standard"
```

```
static UTENTE logged_user;

/* ----------------------------SEZIONE DEDICATA ALL'UTENTE
BASE---------------------------------- */


//RICERCA ASTA PER NOME OGGETTO
void op_1(MYSQL *conn){

	MYSQL_STMT *stmt;
	MYSQL_BIND ps_params[1];  // input parameter buffers

	top:

	//titolo : operazione
	printf("\033[2J\033[H");
	printf("\n\n**** VISUALIZZAZIONE ASTE: RICERCA PER OGGETTO ****\n\n");

	//define input variable
	char nome[64];
	printf("Ricerca aste per nome: ");
	getInput(64, nome, false);
	printf("\n");

	//inizializzazione statement procedurale.
	if(!setup_prepared_stmt(&stmt , "call visualizza_aste_per_nome_oggetto(?)", conn)){
		finish_with_stmt_error(conn, stmt, "Impossibile inizializzare lo statement.\n", false);
		printf("\nPremi invio per continuare...\n");
		while(getchar()!= '\n'){}
		goto top;
	}

	// initialize parameters
```

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

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

        // bind input parameters
        if(mysql_stmt_bind_param(stmt, ps_params) != 0){
                finish_with_stmt_error(conn, stmt, "Errore nel binding dei parametri.\n", true);
                printf("\nPremi invio per continuare...\n");
                while(getchar()!= '\n'){}
                goto top;
        }

        // Run the stored procedure
        if(mysql_stmt_execute(stmt) != 0){
                print_stmt_error(stmt, "\nErrore nella visualizzazione dello stato dell'asta\n");
                return;
        }

        procedure_output(conn, stmt);
        mysql_stmt_close(stmt);
}

//VISUALIZZAZIONE ASTE ATTIVE
void op_2(MYSQL *conn){

        MYSQL_STMT *stmt;

        top:

        //titolo : operazione
        printf("\033[2J\033[H");
        printf("\n\n**** VISUALIZZAZIONE ASTE ATTIVE ****\n\n");
```

```
	//inizializzazione statement procedurale.
	if(!setup_prepared_stmt(&stmt , "call visualizza_aste_attive()", conn)){
		finish_with_stmt_error(conn, stmt, "Impossibile inizializzare lo statement.\n", false);
		printf("\nPremi invio per continuare...\n");
		while(getchar()!= '\n'){}
		goto top;
	}

	// Run the stored procedure
	if(mysql_stmt_execute(stmt) != 0){
		print_stmt_error(stmt, "\nErrore nella visualizzazione delle aste attive\n");
		return;
	}

	procedure_output(conn, stmt);
	mysql_stmt_close(stmt);
}


//VISUALIZZAZIONE STATO DI UN ASTA
void op_3(MYSQL *conn){

	MYSQL_STMT *stmt;
	MYSQL_BIND ps_params[1];  // input parameter buffers

	top:

	//titolo : operazione
	printf("\033[2J\033[H");
	printf("\n\n**** VISUALIZZAZIONE ASTE: VISUALIZZA LO STATO DI UN'ASTA
****\n\n");

	//define input variable
```

```
int *id = malloc(sizeof(int));
char* _id = malloc(sizeof(char)*64);
printf("inserire il codice dell'asta: ");
getInput(64, _id, false);
printf("\n");
*id =  atoi(_id);
free(_id);

//inizializzazione statement procedurale.
if(!setup_prepared_stmt(&stmt , "call visualizza_stato_asta(?)", conn)){
        finish_with_stmt_error(conn, stmt, "Impossibile inizializzare lo statement.\n", false);
        printf("\nPremi invio per continuare...\n");
        while(getchar()!= '\n'){}
        free(id);
        goto top;
}

// initialize parameters
memset(ps_params, 0, sizeof(ps_params));

ps_params[0].buffer_type = MYSQL_TYPE_LONG;
ps_params[0].buffer = id;
ps_params[0].buffer_length = sizeof(int);

// bind input parameters
if(mysql_stmt_bind_param(stmt, ps_params) != 0){
        finish_with_stmt_error(conn, stmt, "Errore nel binding dei parametri.\n", true);
        printf("\nPremi invio per continuare...\n");
        while(getchar()!= '\n'){}
        free(id);
        goto top;
}

// Run the stored procedure
```

```
	if(mysql_stmt_execute(stmt) != 0){
		print_stmt_error(stmt, "\nErrore nella visualizzazione delle aste attive\n");
		free(id);
		return;
	}

	procedure_output(conn, stmt);
	free(id);
	mysql_stmt_close(stmt);

}

//VISUALIZZAZIONE OGGETTI AGGIUDICATI
void op_4(MYSQL *conn){

	MYSQL_STMT *stmt;
	MYSQL_BIND ps_params[1];  // input parameter buffers

	top:

	//titolo : operazione
	printf("\033[2J\033[H");
	printf("\n\n**** VISUALIZZAZIONE OGGETTI AGGIUDICATI ****\n\n");

	//inizializzazione statement procedurale.
	if(!setup_prepared_stmt(&stmt , "call visualizza_oggetti_aggiudicati(?)", conn)){
		finish_with_stmt_error(conn, stmt, "Impossibile inizializzare lo statement.\n", false);
		printf("\nPremi invio per continuare...\n");
		while(getchar()!= '\n'){}
		goto top;
	}

	// initialize parameters
	memset(ps_params, 0, sizeof(ps_params));
```

```
        ps_params[0].buffer_type = MYSQL_TYPE_STRING;
        ps_params[0].buffer = logged_user.cf;
        ps_params[0].buffer_length = strlen(logged_user.cf);

        // bind input parameters
        if(mysql_stmt_bind_param(stmt, ps_params) != 0){
                finish_with_stmt_error(conn, stmt, "Errore nel binding dei parametri.\n", true);
                printf("\nPremi invio per continuare...\n");
                while(getchar()!= '\n'){}
                goto top;
        }

        // Run the stored procedure
        if(mysql_stmt_execute(stmt) != 0){
                print_stmt_error(stmt, "\nErrore nella visualizzazione degli oggetti aggiudicati\n");
                return;
        }

        procedure_output(conn, stmt);
        mysql_stmt_close(stmt);
}

//REGISTRAZIONE OFFERTA SU UN'ASTA
void op_5(MYSQL *conn){

        MYSQL_STMT *stmt;
        MYSQL_BIND ps_params[4];  // input parameter buffers

        top:

        //titolo : operazione
        printf("\033[2J\033[H");
        printf("\n\n**** REGISTRAZIONE OFFERTA SU UN'ASTA ****\n\n");
```

```
//define input variable
char* _id = malloc(sizeof(char)*64);
printf("Inserire il codice dell'asta: ");
getInput(64, _id, false);
int *id = malloc(sizeof(int));
*id = (int) atoi(_id);
free(_id);

char* _offerta = malloc(sizeof(char)*64);
printf("\nInserire l'importo dell'offerta: ");
getInput(64, _offerta, false);
float *offerta = malloc(sizeof(float));
*offerta = (float) atof(_offerta);
free(_offerta);

char* _importo_controff = malloc(sizeof(char)*64);
printf("\nInserire un importo massimo di controfferta: ");
getInput(64, _importo_controff, false);
float *importo_controff = malloc(sizeof(float));
*importo_controff = (float) atof(_offerta);
free(_importo_controff);

//inizializzazione statement procedurale.
if(!setup_prepared_stmt(&stmt , "call registra_offerta(?, ?, ?, ?)", conn)){
        finish_with_stmt_error(conn, stmt, "Impossibile inizializzare lo statement.\n", false);
        printf("\nPremi invio per continuare...\n");
        while(getchar()!= '\n'){}
        free(id);
        free(offerta);
        free(importo_controff);
        goto top;
}
```

```
// initialize parameters
memset(ps_params, 0, sizeof(ps_params));

ps_params[0].buffer_type = MYSQL_TYPE_STRING;
ps_params[0].buffer = logged_user.cf;
ps_params[0].buffer_length = strlen(logged_user.cf);

ps_params[1].buffer_type = MYSQL_TYPE_LONG;
ps_params[1].buffer = id;
ps_params[1].buffer_length = sizeof(int);

ps_params[2].buffer_type = MYSQL_TYPE_FLOAT;
ps_params[2].buffer = offerta;
ps_params[2].buffer_length = sizeof(float);

ps_params[3].buffer_type = MYSQL_TYPE_FLOAT;
ps_params[3].buffer = importo_controff;
ps_params[3].buffer_length = sizeof(float);

// bind input parameters
if(mysql_stmt_bind_param(stmt, ps_params) != 0){
        finish_with_stmt_error(conn, stmt, "Errore nel binding dei parametri.\n", true);
        printf("\nPremi invio per continuare...\n");
        while(getchar()!= '\n'){}
        free(id);
        free(offerta);
        free(importo_controff);
        goto top;
}

// Run the stored procedure
if(mysql_stmt_execute(stmt) != 0){
        print_stmt_error(stmt, "\nErrore nella registrazione dell'offerta\n");
```

```
            free(id);
            free(offerta);
            free(importo_controff);
            return;
        }

        printf(ANSI_COLOR_GREEN"\nL'offerta è stata regolarmente registrata.\n"
ANSI_COLOR_RESET);

        free(id);
        free(offerta);
        free(importo_controff);

        mysql_stmt_close(stmt);
}

//RICERCA ASTE PER ESPOSITORE
void op_6(MYSQL *conn){

        MYSQL_STMT *stmt;
        MYSQL_BIND ps_params[1];  // input parameter buffers

        top:

        //titolo : operazione
        printf("\033[2J\033[H");
        printf("\n\n**** RICERCA ASTE PER ESPOSITORE ****\n\n");


        //define input variable
        char* cf_esp = malloc(sizeof(char)*16);
        printf("Inserire il codice fiscale dell'espositore: ");
        getInput(16, cf_esp, false);
        printf("\n");
```

```
//inizializzazione statement procedurale.
if(!setup_prepared_stmt(&stmt , "call visualizza_aste_per_espositore(?)", conn)){
        finish_with_stmt_error(conn, stmt, "Impossibile inizializzare lo statement.\n", false);
        printf("\nPremi invio per continuare...\n");
        while(getchar()!= '\n'){}
        free(cf_esp);
        goto top;
}

// initialize parameters
memset(ps_params, 0, sizeof(ps_params));

ps_params[0].buffer_type = MYSQL_TYPE_STRING;
ps_params[0].buffer = cf_esp;
ps_params[0].buffer_length = strlen(cf_esp);

// bind input parameters
if(mysql_stmt_bind_param(stmt, ps_params) != 0){
        finish_with_stmt_error(conn, stmt, "Errore nel binding dei parametri.\n", true);
        printf("\nPremi invio per continuare...\n");
        while(getchar()!= '\n'){}
        free(cf_esp);
        goto top;
}

// Run the stored procedure
if(mysql_stmt_execute(stmt) != 0){
        print_stmt_error(stmt, "\nErrore nella visualizzazione delle aste per espositore\n");
        free(cf_esp);
        return;
}

procedure_output(conn, stmt);
```

```
        free(cf_esp);
        mysql_stmt_close(stmt);

}

//RICERCA ASTE PER CATEGORIA
void op_7(MYSQL *conn){

        MYSQL_STMT *stmt;
        MYSQL_BIND ps_params[1];  // input parameter buffers

        top:

        //titolo : operazione
        printf("\033[2J\033[H");
        printf("\n\n**** VISUALIZZAZIONE ASTE: RICERCA PER CATEGORIA ****\n\n");

        //define input variable
        char* categoria = malloc(sizeof(char)*64);
        printf("inserire il nome di una categoria esistente (o parte di esso): ");
        getInput(64, categoria, false);
        printf("\n");

        //inizializzazione statement procedurale.
        if(!setup_prepared_stmt(&stmt , "call visualizza_aste_per_categoria(?)", conn)){
                finish_with_stmt_error(conn, stmt, "Impossibile inizializzare lo statement.\n", false);
                printf("\nPremi invio per continuare...\n");
                while(getchar()!= '\n'){}
                free(categoria);
                goto top;
        }

        // initialize parameters
        memset(ps_params, 0, sizeof(ps_params));
```

```
        ps_params[0].buffer_type = MYSQL_TYPE_VAR_STRING;
        ps_params[0].buffer = categoria;
        ps_params[0].buffer_length = strlen(categoria);

        // bind input parameters
        if(mysql_stmt_bind_param(stmt, ps_params) != 0){
                finish_with_stmt_error(conn, stmt, "Errore nel binding dei parametri.\n", true);
                printf("\nPremi invio per continuare...\n");
                while(getchar()!= '\n'){}
                free(categoria);
                goto top;
        }

        // Run the stored procedure
        if(mysql_stmt_execute(stmt) != 0){
                print_stmt_error(stmt, "\nErrore nella visualizzazione delle aste per categoria\n");
                free(categoria);
                return;
        }

        procedure_output(conn, stmt);
        free(categoria);
        mysql_stmt_close(stmt);
}

//RICERCA ASTE PER PARTECIPAZIONE
void op_9(MYSQL *conn){

        MYSQL_STMT *stmt;
        MYSQL_BIND ps_params[1];  // input parameter buffers

        top:
```

```
        //titolo : operazione
        printf("\033[2J\033[H");
        printf("\n\n**** RICERCA ASTE PER PARTECIPAZIONE ****\n\n");

        //inizializzazione statement procedurale.
        if(!setup_prepared_stmt(&stmt , "call visualizza_partecipazione_aste(?)", conn)){
                finish_with_stmt_error(conn, stmt, "Impossibile inizializzare lo statement.\n", false);
                printf("\nPremi invio per continuare...\n");
                while(getchar()!= '\n'){}
                goto top;
        }

        // initialize parameters
        memset(ps_params, 0, sizeof(ps_params));

        ps_params[0].buffer_type = MYSQL_TYPE_STRING;
        ps_params[0].buffer = logged_user.cf;
        ps_params[0].buffer_length = strlen(logged_user.cf);

        //printf("Utente %s\n", logged_user.cf);



        // bind input parameters
        if(mysql_stmt_bind_param(stmt, ps_params) != 0){
                finish_with_stmt_error(conn, stmt, "Errore nel binding dei parametri.\n", true);
                printf("\nPremi invio per continuare...\n");
                while(getchar()!= '\n'){}
                goto top;
        }

        // Run the stored procedure
        if(mysql_stmt_execute(stmt) != 0){
                print_stmt_error(stmt, "\nErrore nella visualizzazione delle partecipazioni alle aste\
n");
```

```
            return;
        }

        procedure_output(conn, stmt);
        mysql_stmt_close(stmt);
}


//AGGIUNGI UTENTE BASE
void op_10(MYSQL *conn){

        MYSQL_STMT *stmt;
        MYSQL_BIND ps_params[10];  // input parameter buffers

        MYSQL_TIME *date, *scad_carta;
        date = malloc(sizeof(MYSQL_TIME));
        scad_carta = malloc(sizeof(MYSQL_TIME));

        top:

        //titolo : operazione
        printf("\033[2J\033[H");
        printf("\n\n**** AGGIUNGI UTENTE BASE ****\n\n");

        //define input variable
        char *cf = malloc(sizeof(char)*16);
        printf("\nCodice fiscale : ");
        getInput(16, cf, false);

        char *nome = malloc(sizeof(char)*32);
        printf("\nNome utente : ");
        getInput(32, nome, false);

        char *cognome = malloc(sizeof(char)*32);
```

```
printf("\nCognome : ");
getInput(32, cognome, false);

char *pswd = malloc(sizeof(char)*32);
printf("\nPassword di accesso : ");
getInput(32, pswd, false);

char *_anno = malloc(sizeof(char)*32);
printf("\nAnno di nascita : ");
getInput(32, _anno, false);
date->year = atoi(_anno);
free(_anno);

char *_mese = malloc(sizeof(char)*32);
printf("\nMese di nascita : ");
getInput(32, _mese, false);
date->month = atoi(_mese);
free(_mese);

char *_giorno = malloc(sizeof(char)*32);
printf("\nGiorno di nascita : ");
getInput(32, _giorno, false);
date->day = atoi(_giorno);
free(_giorno);

//length[4] = sizeof(MYSQL_TIME);

//--------------------

char *birthplace = malloc(sizeof(char)*32);
printf("\nLuogo di nascita : ");
getInput(32, birthplace, false);

char *numcarta = malloc(sizeof(char)*32);
```

```
        printf("\nNumero di carta : ");
        getInput(32, numcarta, false);

        //scadenza carta suddivisa in 2 sezioni!!!!--

        char *_scadanno = malloc(sizeof(char)*32);
        printf("\nAnno di scadenza : ");
        getInput(32, _scadanno, false);
        scad_carta->year = atoi(_scadanno);
        free(_scadanno);

        char *_scadmese = malloc(sizeof(char)*32);
        printf("\nMese di scadenza : ");
        getInput(32, _scadmese, false);
        scad_carta->month = atoi(_scadmese);
        free(_scadmese);

        //length[7] = sizeof(MYSQL_TIME);

        //-------------------

        char *cvv = malloc(sizeof(char)*10);
        printf("\nCvv : ");
        getInput(10, cvv, false);

        char *indirizzo = malloc(sizeof(char)*32);
        printf("\nIndirizzo di consegna : ");
        getInput(32, indirizzo, false);

        //inizializzazione statement procedurale.
        if(!setup_prepared_stmt(&stmt, "call registrazione_utente_base(?, ?, ?, ?, ?, ?, ?, ?, ?, ?)",
conn)) {
                finish_with_stmt_error(conn, stmt, "Impossibile inizializzare lo statement.\n", false);
                printf("\nPremi invio per continuare...\n");
```

```
            while(getchar()!= '\n'){}
            //free(type);
            free(cf);
            free(nome);
            free(cognome);
            free(pswd);
            free(birthplace);
            free(numcarta);
            free(cvv);
            free(indirizzo);
            free(date);
            free(scad_carta);
            goto top;
}

// initialize parameters
memset(ps_params, 0, sizeof(ps_params));

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

ps_params[1].buffer_type = MYSQL_TYPE_VAR_STRING;
ps_params[1].buffer = nome;
ps_params[1].buffer_length = strlen(nome);

ps_params[2].buffer_type = MYSQL_TYPE_VAR_STRING;
ps_params[2].buffer = cognome;
ps_params[2].buffer_length = strlen(cognome);

ps_params[3].buffer_type = MYSQL_TYPE_DATE;
ps_params[3].buffer = date;
ps_params[3].buffer_length = sizeof(date);
```

```
ps_params[4].buffer_type =  MYSQL_TYPE_VAR_STRING;
ps_params[4].buffer = birthplace;
ps_params[4].buffer_length = strlen(birthplace);

ps_params[5].buffer_type = MYSQL_TYPE_VAR_STRING;
ps_params[5].buffer = numcarta;
ps_params[5].buffer_length = strlen(numcarta);

ps_params[6].buffer_type = MYSQL_TYPE_DATE;
ps_params[6].buffer = scad_carta;
ps_params[6].buffer_length = sizeof(scad_carta);

ps_params[7].buffer_type = MYSQL_TYPE_VAR_STRING;
ps_params[7].buffer = cvv;
ps_params[7].buffer_length = strlen(cvv);

ps_params[8].buffer_type = MYSQL_TYPE_VAR_STRING;
ps_params[8].buffer = indirizzo;
ps_params[8].buffer_length = strlen(indirizzo);

ps_params[9].buffer_type = MYSQL_TYPE_VAR_STRING;
ps_params[9].buffer = pswd;
ps_params[9].buffer_length = strlen(pswd);

// bind input parameters
if(mysql_stmt_bind_param(stmt, ps_params) != 0){
        finish_with_stmt_error(conn, stmt, "Errore nel binding dei parametri.\n", true);
        printf("\nPremi invio per continuare...\n");
        while(getchar()!= '\n'){}
        free(cf);
        free(nome);
        free(cognome);
        free(pswd);
        free(birthplace);
```

```
            free(numcarta);
            free(cvv);
            free(indirizzo);
            free(date);
            free(scad_carta);
            goto top;
    }

    // Run the stored procedure
    if(mysql_stmt_execute(stmt) != 0){
            print_stmt_error(stmt, "\nErrore nella registrazione dell'utente Base.\n");
            free(cf);
            free(nome);
            free(cognome);
            free(pswd);
            free(birthplace);
            free(numcarta);
            free(cvv);
            free(indirizzo);
            free(date);
            free(scad_carta);
            return;
    }

    printf(ANSI_COLOR_GREEN"\nUtente Base registrato\n" ANSI_COLOR_RESET);

    free(cf);
    free(nome);
    free(cognome);
    free(pswd);
    free(birthplace);
    free(numcarta);
    free(cvv);
    free(indirizzo);
```

```
        free(date);
        free(scad_carta);

        mysql_stmt_close(stmt);

}

void standard(MYSQL *conn, char *cf){

        char operation[20];

        if(mysql_change_user(conn, getJson(STANDARD, USERNAME), getJson(STANDARD,
        PASSWORD), getJson(STANDARD, DATABASE))){
                fprintf(stderr, "mysql_change_user() failed\n");
                exit(EXIT_FAILURE);
        }

        memcpy(logged_user.cf, cf, 64);

        start:

        printf("\033[2J\033[H");
        printf("Connected as");
        printf(ANSI_COLOR_BOLD" %s"ANSI_COLOR_OFF , cf);
        printf("-standard\n\n");

        printf(".....\n");
        printf(".....\n");
        printf(".....|      MENU UTENTE      |.....\n");
        printf(".....\n");
        printf(".....\n\n\n");

        printf(" _____ _________ _________ _____Operazioni Disponibili_____ _________ _________
_______ _\n");
```

```
        printf("|                                                                    |\n");
        printf("|                                                  [LOG OUT: 0]    |\n");
        printf("|                                                                    |\n");
        printf("|   OPERAZIONE 1  : Ricerca aste per nome dell' oggetto                    |\n");
        printf("|   OPERAZIONE 2  : Ricerca aste attive                                |\n");
        printf("|   OPERAZIONE 3  : Visualizzazione dello stato di un'asta                 |\n");
        printf("|   OPERAZIONE 4  : Visualizzazione oggetti aggiudicati                    |\n");
        printf("|   OPERAZIONE 5  : Registrazione offerta per un'asta                      |\n");
        printf("|   OPERAZIONE 6  : Ricerca aste per espositore                            |\n");
        printf("|   OPERAZIONE 7  : Ricerca aste attive per categoria di afferenza         |\n");
        printf("|   OPERAZIONE 8  : Visualizzazione categorie                              |\n");
        printf("|   OPERAZIONE 9  : Visualizzazione delle aste a cui ho partecipato        |\n");
        printf("|   OPERAZIONE 10 : Aggiungi utente Base
|\n");
        printf("|____ ________ ________ ________ ________ ________ ________ ________
________ ______ |\n\n");
        printf("   Inserisci il codice dell'operazione : ");

        getInput(64, operation, false);

        if(!strcmp(operation, "0")){
                printf(ANSI_COLOR_RED"\n   Logout...\n\n" ANSI_COLOR_RESET);
                printf("\n\n Premi invio per tornare al Menù Login...\n");
        while(getchar() != '\n'){}
                return;
        }
        else if(!strcmp(operation, "1")){
                op_1(conn);
        }
        else if(!strcmp(operation, "2")){
                op_2(conn);
        }
        else if(!strcmp(operation, "3")){
                op_3(conn);
```

```
        }
        else if(!strcmp(operation, "4")){
                op_4(conn);
        }
        else if(!strcmp(operation, "5")){
                op_5(conn);
        }
        else if(!strcmp(operation, "6")){
                op_6(conn);
        }
        else if(!strcmp(operation, "7")){
                op_7(conn);
        }
        else if(!strcmp(operation, "8")){
                op_a1(conn);
        }
        else if(!strcmp(operation, "9")){
                op_9(conn);
        }
        else if(!strcmp(operation, "10")){
                op_10(conn);
        }
        else{
                printf(ANSI_COLOR_RED"\n   Comando non riconosciuto"
                ANSI_COLOR_RESET);
        }

        printf("\n\n Premi invio per tornare al Menù Utente...\n");
    while(getchar() != '\n'){}
    goto start;

}
```

admin.c

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

#include "defines.h"

#define ADMIN "admin"


char command[20];
static UTENTE logged_user;

/* ----------------------SEZIONE DEDICATA ALL'UTENTE
AMMINISTRATORE---------------------------- */

/*                      *** Gestione categorie ***                              */

//VISUALIZZA TITOLARIO GERARCHICO
void op_a1(MYSQL *conn){

        MYSQL_STMT *stmt;
        int status;

        top:

        //titolo : operazione
        printf("\033[2J\033[H");
        printf("\n\n    **** VISUALIZZAZIONE CATEGORIE : TITOLARIO GERARCHICO
****\n\n");

        //inizializzazione statement procedurale.
        if(!setup_prepared_stmt(&stmt , "call visualizzazione_titolario_gerarchico()", conn)){
                finish_with_stmt_error(conn, stmt, "Impossibile inizializzare lo statement.\n", false);
                printf("\nPremi invio per continuare...\n");
```

```
            while(getchar()!= '\n'){}
            goto top;
        }

        // Run the stored procedure
        if(mysql_stmt_execute(stmt) != 0){
            print_stmt_error(stmt, "Errore nella visualizzazione del titolario gerarchico.\n");
            printf("\nPremi invio per continuare...\n");
            while(getchar()!= '\n'){}
            return;
        }

        procedure_output(conn, stmt);
        mysql_stmt_close(stmt);

}


//INSERIMENTO CATEGORIA
void op_a2(MYSQL *conn){

        MYSQL_STMT *stmt;

        top:

        printf("\033[2J\033[H");
        printf("\n\n          **** INSERIMENTO CATEGORIA ****\n\n");
        printf(" _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ \n");
        printf("|                                                    |\n");
        printf("| In questa sezione è possibile inserire nuove categorie      |\n");
        printf("| nel titolario corrente. Il titolario è strutturato in       |\n");
        printf("| una gerarchia a tre livelli:                              |\n");
        printf("|                                                    |\n");
        printf("| Lv 1 : categorie genitori di categorie di Lv (1)           |\n");
```

```
printf("| Lv 2 : categorie genitori di categorie di Lv (2)           |\n");
printf("| Lv 3 : categorie effettivamente associabili ad oggetti Lv(3) |\n");
printf("|                                                            |\n");
printf("| E' possibile inserire una nuova categoria in uno           |\n");
printf("| qualsiasi di questi livelli, indicando il suo nome         |\n");
printf("| e quello della categoria genitore, se esistente.           |\n");
printf("| _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _| \n");
printf("\nDigitare il livello del titolario in cui si desidera inserire la nuova categoria.\n");
printf("\nAltrimenti, digitare il codice '0' per visionare il titolario corrente... ");

char *_lv = malloc(sizeof(char) * 32);
int lv;
getInput(32, _lv, false);
lv = atoi(_lv);
free(_lv);

int answer;

switch(lv){

        case 0:
                op_a1(conn);
                printf("\n\n Premi invio per tornare alla scelta del livello...\n");
                while(getchar() != '\n'){}
                        printf("\033[2J\033[H");
                goto top;
                break;

        case 1:

                printf("\033[2J\033[H");
                printf("INSERIMENTO CATEGORIA: LIVELLO 1 \n");

                MYSQL_BIND ps_params[1];  // input parameter buffers
```

```
                    //define input variable
                    char nome[64];
                    printf("\nInserire il nome della nuova categoria: ");
                    getInput(64, nome, false);

                    //inizializzazione statement procedurale.
                    if(!setup_prepared_stmt(&stmt, "call inserimento_categoria1(?)", conn)) {
                            finish_with_stmt_error(conn, stmt, "Impossibile inizializzare lo
statement.\n", false);
                            printf("\nPremi invio per continuare...\n");
                            while(getchar()!= '\n'){}
                            goto top;
                    }

                    // initialize parameters
                    memset(ps_params, 0, sizeof(ps_params));

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

                    // bind input parameters
                    if(mysql_stmt_bind_param(stmt, ps_params) != 0){
                                    finish_with_stmt_error(conn, stmt, "Errore nel binding dei
parametri.\n", true);
                                    printf("\nPremi invio per continuare...\n");
                                    while(getchar()!= '\n'){}
                                    goto top;
                            }


                    // Run the stored procedure
                    if(mysql_stmt_execute(stmt) != 0){
```

```
                print_stmt_error(stmt, "\nErrore nell'inserimento della categoria di
livello 1.\n");
                return;
            }

            printf(ANSI_COLOR_GREEN "\nCategoria inserita correttamente\n"
ANSI_COLOR_RESET);
            break;


        case 2:

            printf("\033[2J\033[H");
            printf("INSERIMENTO CATEGORIA: LIVELLO 2 \n");

            printf("Si desidera associare la categoria a un genitore al livello 1 ?\n");
            printf("Digitare il codice di risposta [ 0 = sì  |  1 = no ] : ");

            char* _answer2 = malloc(sizeof(char)*32);
            getInput(32, _answer2, false);
            answer = atoi(_answer2);
            free(_answer2);

            if(answer == 0){

                MYSQL_BIND ps_params[2];  // input parameter buffers

                //define input variable
                char nome[64];
                printf("\nInserire il nome della nuova categoria: ");
                getInput(64, nome, false);

                char nomePadre[64];
                printf("\nInserire il nome della categoria genitore: ");
```

```
                    getInput(64, nomePadre, false);


                    //inizializzazione statement procedurale.
                    if(!setup_prepared_stmt(&stmt, "call inserimento_categoria2(?, ?)",
conn)) {
                        finish_with_stmt_error(conn, stmt, "Impossibile inizializzare lo
statement.\n", false);
                        printf("\nPremi invio per continuare...\n");
                        while(getchar()!= '\n'){}
                        goto top;
                    }


                    // initialize parameters
                    memset(ps_params, 0, sizeof(ps_params));

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

                    ps_params[1].buffer_type = MYSQL_TYPE_VAR_STRING;
                    ps_params[1].buffer = nomePadre;
                    ps_params[1].buffer_length = strlen(nomePadre);

                    // bind input parameters
                    if(mysql_stmt_bind_param(stmt, ps_params) != 0){
                        finish_with_stmt_error(conn, stmt, "Errore nel binding dei
parametri.\n", true);
                        printf("\nPremi invio per continuare...\n");
                        while(getchar()!= '\n'){}
                        goto top;
                    }
```

```
                    // Run the stored procedure
                    if(mysql_stmt_execute(stmt) != 0){
                        print_stmt_error(stmt, "\nErrore nell'inserimento della categoria
di livello 2.\n");
                        return;
                    }

                    printf(ANSI_COLOR_GREEN "\nCategoria inserita correttamente\n"
ANSI_COLOR_RESET);

                } else{

                    MYSQL_BIND ps_params[1];  // input parameter buffers

                    //define input variable
                    char nome[64];
                    printf("\nInserire il nome della nuova categoria: ");
                    getInput(64, nome, false);


                    //inizializzazione statement procedurale.
                    if(!setup_prepared_stmt(&stmt, "call inserimento_categoria2(?, ?)",
conn)) {
                        finish_with_stmt_error(conn, stmt, "Impossibile inizializzare lo
statement.\n", false);
                        printf("\nPremi invio per continuare...\n");
                        while(getchar()!= '\n'){}
                        goto top;
                    }


                    // initialize parameters
                    memset(ps_params, 0, sizeof(ps_params));
```

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


                    // bind input parameters
                    if(mysql_stmt_bind_param(stmt, ps_params) != 0){
                        finish_with_stmt_error(conn, stmt, "Errore nel binding dei
parametri.\n", true);
                        printf("\nPremi invio per continuare...\n");
                        while(getchar()!= '\n'){}
                        goto top;
                    }

                    // Run the stored procedure
                    if(mysql_stmt_execute(stmt) != 0){
                        print_stmt_error(stmt, "\nErrore nell'inserimento della categoria
di livello 2.\n");
                        return;
                    }

                    printf(ANSI_COLOR_GREEN "\nCategoria inserita correttamente\n"
ANSI_COLOR_RESET);

                }

                break;

            case 3:
                printf("\033[2J\033[H");
                printf("INSERIMENTO CATEGORIA: LIVELLO 3 \n");

                printf("Si desidera associare la categoria a un genitore al livello 2 ?\n");
                printf("Digitare il codice di risposta [ 0 = sì  |  1 = no ] : ");
```

```
char* _answer = malloc(sizeof(char)*32);
getInput(32, _answer, false);
answer = atoi(_answer);
free(_answer);

if(answer == 0){

        MYSQL_BIND ps_params[2];          // input parameter buffers

        //define input variable
        char nome[64];
        printf("\nInserire il nome della nuova categoria: ");
        getInput(64, nome, false);

        char nomePadre[64];
        printf("\nInserire il nome della categoria genitore: ");
        getInput(64, nomePadre, false);


        //inizializzazione statement procedurale.
        if(!setup_prepared_stmt(&stmt, "call inserimento_categoria3(?, ?)",
conn)) {
                finish_with_stmt_error(conn, stmt, "Impossibile inizializzare lo
statement.\n", false);
                printf("\nPremi invio per continuare...\n");
                while(getchar()!= '\n'){}
                goto top;
        }


        // prepare parameters
        memset(ps_params, 0, sizeof(ps_params));
```

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

                    ps_params[1].buffer_type = MYSQL_TYPE_VAR_STRING;
                    ps_params[1].buffer = nomePadre;
                    ps_params[1].buffer_length = strlen(nomePadre);

                    // bind input parameters
                    if(mysql_stmt_bind_param(stmt, ps_params) != 0){
                        finish_with_stmt_error(conn, stmt, "Errore nel binding dei
parametri.\n", true);
                        printf("\nPremi invio per continuare...\n");
                        while(getchar()!= '\n'){}
                        goto top;
                    }

                    // Run the stored procedure
                    if(mysql_stmt_execute(stmt) != 0){
                        print_stmt_error(stmt, "\nErrore nell'inserimento della categoria
di livello 3.\n");
                        return;
                    }

                    printf(ANSI_COLOR_GREEN "\nCategoria inserita correttamente\n"
ANSI_COLOR_RESET);

                } else{

                    MYSQL_BIND ps_params[1];  // input parameter buffers

                    //define input variable
                    char nome[64];
                    printf("\nInserire il nome della nuova categoria: ");
```

```
                    getInput(64, nome, false);


                    //inizializzazione statement procedurale.
                    if(!setup_prepared_stmt(&stmt, "call inserimento_categoria3(?, ?)",
conn)) {
                        finish_with_stmt_error(conn, stmt, "Impossibile inizializzare lo
statement.\n", false);
                        printf("\nPremi invio per continuare...\n");
                        while(getchar()!= '\n'){}
                        goto top;
                    }


                    // initialize parameters
                    memset(ps_params, 0, sizeof(ps_params));

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

                    // bind input parameters
                    if(mysql_stmt_bind_param(stmt, ps_params) != 0){
                        finish_with_stmt_error(conn, stmt, "Errore nel binding dei
parametri.\n", true);
                        printf("\nPremi invio per continuare...\n");
                        while(getchar()!= '\n'){}
                        goto top;
                    }

                    // Run the stored procedure
                    if(mysql_stmt_execute(stmt) != 0){
                        print_stmt_error(stmt, "\nErrore nell'inserimento della categoria
di livello 3.\n");
```

```
                                return;
                        }

                                printf(ANSI_COLOR_GREEN "\nCategoria inserita correttamente\n"
ANSI_COLOR_RESET);

                        }

                        break;


                default:
                        goto top;
                        break;
        }

        mysql_stmt_close(stmt);
}


//AGGIORNAMENTO CATEGORIA
void op_a3(MYSQL *conn){

        MYSQL_STMT *stmt;

        top:

        printf("\033[2J\033[H");
        printf("\n\n          **** AGGIORNAMENTO CATEGORIA ****\n\n");
        printf(" _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _\n");
        printf("|                                                |\n");
        printf("| In questa sezione è possibile modificare il nome di    |\n");
        printf("| una categoria e/o associarla ad una categoria genitore,|\n");
        printf("| scelta tra quelle esistenti al livello superiore del   |\n");
```

```
printf("| titolario. Si scelga la tipologia di aggiornamento     |\n");
printf("| tra le seguenti:                                       |\n");
printf("|                                                        |\n");
printf("| 1. Modifica del nome di una categoria                  |\n");
printf("| 2. Associa un padre ad una categoria (orfana)          |\n");
printf("| 3. Cambia padre ad una categoria                       |\n");
printf("| _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _|\n");
printf("\nDigitare il codice relativo alla tipologia di aggiornamento desiderata.\n");
printf("\nAltrimenti, digitare il codice '0' per visionare il titolario corrente... ");

char *_cod = malloc(sizeof(char) * 32);
int cod;
getInput(32, _cod, false);
cod = atoi(_cod);
free(_cod);

MYSQL_BIND ps_params[3];  // input parameter buffers
char nome[64];
char nuovo_nome[64];
char nomePadre[64];
char new_nomePadre[64];

switch(cod){

        case 0:
                op_a1(conn);
                printf("\n\n Premi invio per tornare alla scelta del livello...\n");
                while(getchar() != '\n'){}
                printf("\033[2J\033[H");
                goto top;
                break;

        case 1:
```

```
                    //define input variable
                    printf("\nInserire il nome attuale della categoria : ");
                    getInput(64, nome, false);

                    printf("\nInserire il nuovo nome : ");
                    getInput(64, nuovo_nome, false);

                    //inizializzazione statement procedurale.
                    if(!setup_prepared_stmt(&stmt, "call aggiornamento_categoria(?, ?, null,
null)", conn)) {
                            finish_with_stmt_error(conn, stmt, "Impossibile inizializzare lo
statement.\n", false);
                            printf("\nPremi invio per continuare...\n");
                            while(getchar()!= '\n'){}
                            goto top;
                    }

                    // initialize parameters
                    memset(ps_params, 0, sizeof(ps_params));

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

                    ps_params[1].buffer_type = MYSQL_TYPE_VAR_STRING;
                    ps_params[1].buffer = nuovo_nome;
                    ps_params[1].buffer_length = strlen(nuovo_nome);

                    // bind input parameters
                    if(mysql_stmt_bind_param(stmt, ps_params) != 0){
                            finish_with_stmt_error(conn, stmt, "Errore nel binding dei parametri.\
n", true);
                            printf("\nPremi invio per continuare...\n");
                            while(getchar()!= '\n'){}
```

```
                goto top;
            }

            // Run the stored procedure
            if(mysql_stmt_execute(stmt) != 0){
                print_stmt_error(stmt, "\nErrore nell'aggiornamento della categoria.\
n");
                return;
            }

            printf(ANSI_COLOR_GREEN "\nNome modificato correttamente\n"
ANSI_COLOR_RESET);
            break;

        case 2:
            //define input variable
            printf("\nInserire il nome della categoria : ");
            getInput(64, nome, false);

            printf("\nInserire il nome della categoria che si vuole rendere padre : ");
            getInput(64, nomePadre, false);

            //inizializzazione statement procedurale.
            if(!setup_prepared_stmt(&stmt, "call aggiornamento_categoria(?, null,
null, ?)", conn)) {
                finish_with_stmt_error(conn, stmt, "Impossibile inizializzare lo
statement.\n", false);
                printf("\nPremi invio per continuare...\n");
                while(getchar()!= '\n'){}
                goto top;
            }

            // initialize parameters
            memset(ps_params, 0, sizeof(ps_params));
```

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

                ps_params[1].buffer_type = MYSQL_TYPE_VAR_STRING;
                ps_params[1].buffer = nomePadre;
                ps_params[1].buffer_length = strlen(nomePadre);

                // bind input parameters
                if(mysql_stmt_bind_param(stmt, ps_params) != 0){
                        finish_with_stmt_error(conn, stmt, "Errore nel binding dei parametri.\
n", true);
                        printf("\nPremi invio per continuare...\n");
                        while(getchar()!= '\n'){}
                        goto top;
                }

                // Run the stored procedure
                if(mysql_stmt_execute(stmt) != 0){
                        print_stmt_error(stmt, "\nErrore nell'associazione della categoria.\n");
                        return;
                }

                printf(ANSI_COLOR_GREEN "\nAssociazione completata\n"
ANSI_COLOR_RESET);
                break;

        case 3:

                //define input variable
                printf("\nInserire il nome della categoria : ");
                getInput(64, nome, false);
```

```
                printf("\nInserire il nome della attuale categoria padre : ");
                getInput(64, nomePadre, false);

                printf("\nInserire il nome della categoria che si vuole rendere padre : ");
                getInput(64, new_nomePadre, false);

                //inizializzazione statement procedurale.
                if(!setup_prepared_stmt(&stmt, "call aggiornamento_categoria(?, null, ?, ?)",
conn)) {
                        finish_with_stmt_error(conn, stmt, "Impossibile inizializzare lo
statement.\n", false);
                        printf("\nPremi invio per continuare...\n");
                        while(getchar()!= '\n'){}
                        goto top;
                }

                // initialize parameters
                memset(ps_params, 0, sizeof(ps_params));

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

                ps_params[1].buffer_type = MYSQL_TYPE_VAR_STRING;
                ps_params[1].buffer = nomePadre;
                ps_params[1].buffer_length = strlen(nomePadre);

                ps_params[2].buffer_type = MYSQL_TYPE_VAR_STRING;
                ps_params[2].buffer = new_nomePadre;
                ps_params[2].buffer_length = strlen(new_nomePadre);

                // bind input parameters
                if(mysql_stmt_bind_param(stmt, ps_params) != 0){
                        finish_with_stmt_error(conn, stmt, "Errore nel binding dei parametri.\
```

```
n", true);
                        printf("\nPremi invio per continuare...\n");
                        while(getchar()!= '\n'){}
                        goto top;
                }

                // Run the stored procedure
                if(mysql_stmt_execute(stmt) != 0){
                        print_stmt_error(stmt, "\nErrore nell'associazione della categoria.\n");
                        return;
                }

                printf(ANSI_COLOR_GREEN "\nAssociazione completata\n"
ANSI_COLOR_RESET);
                break;

        default:
                goto top;
                break;
    }

    mysql_stmt_close(stmt);
}



//CANCELLAZIONE CATEGORIA ****
void op_a4(MYSQL *conn){

    MYSQL_STMT *stmt;
    MYSQL_BIND ps_params[1];  // input parameter buffers

    //define input variable
    char nome[64];
```

```
top:

printf("\033[2J\033[H");
printf("\nInserire il nome della categoria da eliminare: ");

getInput(64, nome, false);

//inizializzazione statement procedurale.
if(!setup_prepared_stmt(&stmt, "call cancellazione_categoria(?)", conn)) {
        finish_with_stmt_error(conn, stmt, "Impossibile inizializzare lo statement.\n", false);
        printf("\nPremi invio per continuare...\n");
        while(getchar()!= '\n'){}
        goto top;
}

// initialize parameters
memset(ps_params, 0, sizeof(ps_params));

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

// bind input parameters
if(mysql_stmt_bind_param(stmt, ps_params) != 0){
        finish_with_stmt_error(conn, stmt, "Errore nel binding dei parametri.\n", true);
        printf("\nPremi invio per continuare...\n");
        while(getchar()!= '\n'){}
        goto top;
}

// Run the stored procedure
if(mysql_stmt_execute(stmt) != 0){
        print_stmt_error(stmt, "\nErrore nella calcellazione della categoria.\n");
```

```
            return;
        }

        printf(ANSI_COLOR_GREEN "Cancellazione effettuata" ANSI_COLOR_RESET);
        mysql_stmt_close(stmt);
}

//INIZIALIZZAZIONE NUOVA ASTA
void op_a5(MYSQL *conn){

        MYSQL_STMT *stmt;
        MYSQL_BIND ps_params[9];  // input parameter buffers

        top:

        //titolo : operazione
        printf("\033[2J\033[H");
        printf("\n\n**** INIZIALIZZAZIONE NUOVA ASTA ****\n\n");

        //define input variable
        char* nome = malloc(sizeof(char)*64);
        printf("inserire il nome dell'oggetto all'asta: ");
        getInput(64, nome, false);

        char* cat = malloc(sizeof(char)*64);
        printf("\ninserire la categoria di afferenza dell'oggetto: ");
        getInput(64, cat, false);

        char* desc = malloc(sizeof(char)*64);
        printf("\ninserire una breve descrizione dell' oggetto: ");
        getInput(64, desc, false);

        char* stato = malloc(sizeof(char)*64);
        printf("\ninserire lo stato attuale dell'oggetto (nuovo/usato/buone condizioni/etc...): ");
```

```
getInput(64, stato, false);

char* colore = malloc(sizeof(char)*64);
printf("\ninserire il colore dell'oggetto: ");
getInput(64, colore, false);

char* dimens = malloc(sizeof(char)*64);
printf("\ninserire le dimensioni dell'oggetto: ");
getInput(64, dimens, false);

char* _prezzobase = malloc(sizeof(char)*64);
printf("\ninserire il prezzo di partenza per l'oggetto all'asta: ");
getInput(64, _prezzobase, false);
float *prezzobase = malloc(sizeof(float));
*prezzobase = (float) atof(_prezzobase);
free(_prezzobase);

char* _durata = malloc(sizeof(char)*64);
printf("\ninserire la durata, in giorni, dell'asta (il range da considerare è [1, 7] ) : ");
getInput(64, _durata, false);
int *durata = malloc(sizeof(int));
*durata = (int) atoi(_durata);
free(_durata);

//inizializzazione statement procedurale.
if(!setup_prepared_stmt(&stmt, "call inizializzazione_asta(?, ?, ?, ?, ?, ?, ?, ?, ?)", conn)) {
        finish_with_stmt_error(conn, stmt, "Impossibile inizializzare lo statement.\n", false);
        printf("\nPremi invio per continuare...\n");
        while(getchar()!= '\n'){}
        free(nome);
        free(cat);
        free(desc);
        free(stato);
        free(colore);
```

```
            free(dimens);
            free(prezzobase);
            free(durata);
            goto top;
    }

    // initialize parameters
    memset(ps_params, 0, sizeof(ps_params));

    ps_params[0].buffer_type = MYSQL_TYPE_STRING;
    ps_params[0].buffer = nome;
    ps_params[0].buffer_length = strlen(nome);

    ps_params[1].buffer_type = MYSQL_TYPE_STRING;
    ps_params[1].buffer = cat;
    ps_params[1].buffer_length = strlen(cat);

    ps_params[2].buffer_type = MYSQL_TYPE_STRING;
    ps_params[2].buffer = desc;
    ps_params[2].buffer_length = strlen(desc);

    ps_params[3].buffer_type = MYSQL_TYPE_STRING;
    ps_params[3].buffer = stato;
    ps_params[3].buffer_length = strlen(stato);

    ps_params[4].buffer_type = MYSQL_TYPE_STRING;
    ps_params[4].buffer = colore;
    ps_params[4].buffer_length = strlen(colore);

    ps_params[5].buffer_type = MYSQL_TYPE_STRING;
    ps_params[5].buffer = logged_user.cf;
    ps_params[5].buffer_length = strlen(logged_user.cf);

    ps_params[6].buffer_type = MYSQL_TYPE_STRING;
```

```
ps_params[6].buffer = dimens;
ps_params[6].buffer_length = strlen(dimens);

ps_params[7].buffer_type = MYSQL_TYPE_FLOAT;
ps_params[7].buffer = prezzobase;
ps_params[7].buffer_length = sizeof(float);

ps_params[8].buffer_type = MYSQL_TYPE_LONG;
ps_params[8].buffer = durata;
ps_params[8].buffer_length = sizeof(int);

// bind input parameters
if(mysql_stmt_bind_param(stmt, ps_params) != 0){
        finish_with_stmt_error(conn, stmt, "Errore nel binding dei parametri.\n", true);
        printf("\nPremi invio per continuare...\n");
        while(getchar()!= '\n'){}
        free(nome);
        free(cat);
        free(desc);
        free(stato);
        free(colore);
        free(dimens);
        free(prezzobase);
        free(durata);
        goto top;
}

// Run the stored procedure
if(mysql_stmt_execute(stmt) != 0){
        print_stmt_error(stmt, "\nErrore nell'inserimento dell'oggetto.\n");
        free(nome);
        free(cat);
        free(desc);
        free(stato);
```

```
            free(colore);
            free(dimens);
            free(prezzobase);
            free(durata);
            return;
        }

        printf(ANSI_COLOR_GREEN  "\nOggetto inserito correttamente" ANSI_COLOR_RESET);

        free(nome);
        free(cat);
        free(desc);
        free(stato);
        free(colore);
        free(dimens);
        free(prezzobase);
        free(durata);

        mysql_stmt_close(stmt);
}

//REPORT ASTA
void op_a6(MYSQL *conn){

        MYSQL_STMT *stmt;
        MYSQL_BIND ps_params[1];  // input parameter buffers

        top:

        //titolo : operazione
        printf("\033[2J\033[H");
        printf("\n\n**** REPORT ASTA ****\n\n");

        //define input variable
```

```
int *id = malloc(sizeof(int));
char* _id = malloc(sizeof(char)*64);
printf("Inserire il codice dell'asta: ");
getInput(64, _id, false);
*id =  atoi(_id);
free(_id);
printf("\n");

//inizializzazione statement procedurale.
if(!setup_prepared_stmt(&stmt, "call report_asta(?)", conn)) {
        finish_with_stmt_error(conn, stmt, "Impossibile inizializzare lo statement.\n", false);
        printf("\nPremi invio per continuare...\n");
        while(getchar()!= '\n'){}
        free(id);
        goto top;
}

// initialize parameters
memset(ps_params, 0, sizeof(ps_params));

ps_params[0].buffer_type = MYSQL_TYPE_LONG;
ps_params[0].buffer = id;
ps_params[0].buffer_length = sizeof(int);

// bind input parameters
if(mysql_stmt_bind_param(stmt, ps_params) != 0){
        finish_with_stmt_error(conn, stmt, "Errore nel binding dei parametri.\n", true);
        printf("\nPremi invio per continuare...\n");
        while(getchar()!= '\n'){}
        free(id);
        goto top;
}

// Run the stored procedure
```

```
	if(mysql_stmt_execute(stmt) != 0){
		print_stmt_error(stmt, "\nErrore nell'inserimento dell'oggetto.\n");
		free(id);
		return;
	}

	procedure_output(conn, stmt);
	free(id);
	mysql_stmt_close(stmt);
}


//VISUALIZZAZIONE ASTE INDETTE
void op_a7(MYSQL *conn){

	MYSQL_STMT *stmt;
	MYSQL_BIND ps_params[1];  // input parameter buffers

	top:

	//titolo : operazione
	printf("\033[2J\033[H");
	printf("\n\n**** VISUALIZZAZIONE ASTE INDETTE ****\n\n\n");

	//inizializzazione statement procedurale.
	if(!setup_prepared_stmt(&stmt, "call visualizza_aste_per_espositore(?)", conn)) {
		finish_with_stmt_error(conn, stmt, "Impossibile inizializzare lo statement.\n", false);
		printf("\nPremi invio per continuare...\n");
		while(getchar()!= '\n'){}
		goto top;
	}

	// initialize parameters
	memset(ps_params, 0, sizeof(ps_params));
```

```
        ps_params[0].buffer_type = MYSQL_TYPE_STRING;
        ps_params[0].buffer = logged_user.cf;
        ps_params[0].buffer_length = strlen(logged_user.cf);

        // bind input parameters
        if(mysql_stmt_bind_param(stmt, ps_params) != 0){
                finish_with_stmt_error(conn, stmt, "Errore nel binding dei parametri.\n", true);
                printf("\nPremi invio per continuare...\n");
                while(getchar()!= '\n'){}
                goto top;
        }

        // Run the stored procedure
        if(mysql_stmt_execute(stmt) != 0){
                print_stmt_error(stmt, "\nErrore nella visualizzazione delle aste indette.\n");
                return;
        }

        procedure_output(conn, stmt);
        mysql_stmt_close(stmt);
}

//AGGIUNGI UTENTE AMMINISTRATORE
void op_a9(MYSQL *conn){

        MYSQL_STMT *stmt;
        MYSQL_BIND ps_params[4];  // input parameter buffers

        top:

        //titolo : operazione
        printf("\033[2J\033[H");
        printf("\n\n**** AGGIUNGI UTENTE AMMINISTRATORE ****\n\n");
```

```
//define input variable
char *cf = malloc(sizeof(char)*32);
printf("\nCodice fiscale : ");
getInput(32, cf, false);

char *nome = malloc(sizeof(char)*32);
printf("\nNome utente : ");
getInput(32, nome, false);

char *cognome = malloc(sizeof(char)*32);
printf("\nCognome : ");
getInput(32, cognome, false);

char *pswd = malloc(sizeof(char)*32);
printf("\nPassword di accesso : ");
getInput(32, pswd, false);
printf("\n");

//inizializzazione statement procedurale.
if(!setup_prepared_stmt(&stmt, "call registrazione_utente_amm(?, ?, ?, ?)", conn)) {
        finish_with_stmt_error(conn, stmt, "Impossibile inizializzare lo statement.\n", false);
        printf("\nPremi invio per continuare...\n");
        while(getchar()!= '\n'){}
        //free(type);
        free(cf);
        free(nome);
        free(cognome);
        free(pswd);
        goto top;
}

// initialize parameters
memset(ps_params, 0, sizeof(ps_params));
```

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

ps_params[1].buffer_type = MYSQL_TYPE_VAR_STRING;
ps_params[1].buffer = nome;
ps_params[1].buffer_length = strlen(nome);

ps_params[2].buffer_type = MYSQL_TYPE_VAR_STRING;
ps_params[2].buffer = cognome;
ps_params[2].buffer_length = strlen(cognome);

ps_params[3].buffer_type = MYSQL_TYPE_VAR_STRING;
ps_params[3].buffer = pswd;
ps_params[3].buffer_length = strlen(pswd);

// bind input parameters
if(mysql_stmt_bind_param(stmt, ps_params) != 0){
        finish_with_stmt_error(conn, stmt, "Errore nel binding dei parametri.\n", true);
        printf("\nPremi invio per continuare...\n");
        while(getchar()!= '\n'){}
        //free(type);
        free(cf);
        free(nome);
        free(cognome);
        free(pswd);
        goto top;
}

// Run the stored procedure
if(mysql_stmt_execute(stmt) != 0){
        print_stmt_error(stmt, "\nErrore nella registrazione dell'utente Amministratore.\n");
        free(cf);
```

```
                free(nome);
                free(cognome);
                free(pswd);
                return;
        }

        printf(ANSI_COLOR_GREEN"\nUtente Amministratore registrato\n"
ANSI_COLOR_RESET);
        free(cf);
        free(nome);
        free(cognome);
        free(pswd);

        mysql_stmt_close(stmt);
}



void admin(MYSQL *conn, char *cf){

        char operation[20];

        if(mysql_change_user(conn, getJson(ADMIN, USERNAME), getJson(ADMIN,
        PASSWORD), getJson(ADMIN, DATABASE))){
                fprintf(stderr, "mysql_change_user() failed\n");
                exit(EXIT_FAILURE);
        }

        memcpy(logged_user.cf, cf, 64);

        start:

        printf("\033[2J\033[H");
        printf("Connected as");
```

```
        printf(ANSI_COLOR_BOLD" %s"ANSI_COLOR_OFF , cf);
        printf("-admin\n\n");

        printf(".....\n");
        printf(".....\n");
        printf(".....|    MENU AMMINISTRATORE    |.....\n");
        printf(".....\n");
        printf(".....\n\n\n");

        printf(" ____ ________ ________ ____ Gestione delle Categorie ____ ________ ________
_______\n");
        printf("|                                                    |\n");
        printf("|                                              [LOG OUT: 0]   |\n");
        printf("|                                                    |\n");
        printf("|   OPERAZIONE 1 : Visualizzazione titolario completo                |\n");
        printf("|   OPERAZIONE 2 : Inserimento categoria                             |\n");
        printf("|   OPERAZIONE 3 : Aggiornamento categoria                           |\n");
        printf("|   OPERAZIONE 4 : Cancellazione categoria                           |\n");
        printf("|____ ________ ________ ________ ________ ________ ________ ________
________ ______|\n\n");

        printf(" _____ ________ ________ ______ Gestione delle Aste ______ ________ ________
_______\n");
        printf("|                                                    |\n");
        printf("|   OPERAZIONE 5 : Inizializzazione di una nuova asta                |\n");
        printf("|   OPERAZIONE 6 : Generazione Report di un'asta                     |\n");
        printf("|   OPERAZIONE 7 : Visualizzazione aste indette                      |\n");
        printf("|   OPERAZIONE 8 : Visualizzazione dello stato di un'asta            |\n");
        printf("|   OPERAZIONE 9 : Aggiungu utente Amministratore                    |\n");
        printf("|____ ________ ________ ________ ________ ________ ________ ________
________ ______|\n\n");
        printf("   Inserisci il codice dell'operazione : ");

        getInput(64, operation, false);
```

```
if(!strcmp(operation, "0")){
        printf(ANSI_COLOR_RED"\n   Logout...\n\n" ANSI_COLOR_RESET);
        printf("\n\n Premi invio per tornare al Menù Login...\n");
while(getchar() != '\n'){}
        return;
}
else if(!strcmp(operation, "1")){
        op_a1(conn);
}
else if(!strcmp(operation, "2")){
        op_a2(conn);
}
else if(!strcmp(operation, "3")){
        op_a3(conn);
}
else if(!strcmp(operation, "4")){
        op_a4(conn);
}
else if(!strcmp(operation, "5")){
        op_a5(conn);
}
else if(!strcmp(operation, "6")){
        op_a6(conn);
}
else if(!strcmp(operation, "7")){
        op_a7(conn);
}
else if(!strcmp(operation, "8")){
        op_3(conn);
}
else if(!strcmp(operation, "9")){
        op_a9(conn);
}
```

```
        else{
                printf(ANSI_COLOR_RED"\n   Comando non riconosciuto"
                ANSI_COLOR_RESET);
        }

        printf("\n\n Premi invio per tornare al Menù Utente...\n");
      while(getchar() != '\n'){}
      goto start;

}
```

utils.c

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

#include "defines.h"

int *buff_table;

void print_error(MYSQL *con, char *message){

        fprintf(stderr, "%s\n", message);
        if(con != NULL){
        #if MYSQL_VERSION_ID >= 40101
                fprintf(stderr, "Error %u, (%s): %s\n", mysql_errno(con), mysql_sqlstate(con),
mysql_error(con));
        #else
                fprintf(stderr, "Error %u: %s\n", mysql_errno(con), mysql_error(con));
        #endif
        }
}

void print_stmt_error(MYSQL_STMT *stmt, char *message){
```

```
	fprintf(stderr, "%s\n", message);
	if(stmt != NULL){
		fprintf(stderr, "Error %u (%s): %s\n", mysql_stmt_errno(stmt),
mysql_stmt_sqlstate(stmt), mysql_stmt_error(stmt));
	}
}


bool setup_prepared_stmt(MYSQL_STMT **stmt, char *statement, MYSQL *con){

	bool update_length = true;
	*stmt = mysql_stmt_init(con);
	if(*stmt == NULL){
		print_error(con, "Could not initializestatement 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_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);
```

```
}


int checkDate(MYSQL_TIME *date){

	struct tm* ptr;
	time_t t;
	t = time(NULL);
	ptr = localtime(&t);

	//check year
	if(date->year > ptr->tm_year + 1900){
	return 1;
	}
	else if (date->year < ptr->tm_year + 1900)
	{
	return 0;
	}
	// year1 ==  year2
	else{
		//check month
	if (date->month ==  ptr->tm_mon + 1){
		// check day
			if (date->day ==  ptr->tm_mday){
			//check hour
			if(date->hour == ptr->tm_hour + 1){
				//chech minute
				if(date->minute == ptr->tm_min + 1){
					//check second
					if(date->second > ptr->tm_sec + 1){
						return 1;
					}
						else{
					return 0;
```

```
                                        }
                                }
                                else if(date->minute > ptr->tm_min + 1){
                                        return 1;
                                        }
                                        else{
                                return 0;
                                        }
                        }
                        else if(date->hour> ptr->tm_hour + 1){
                                return 1;
                                }
                                else{
                                return 0;
                                }
                }
                        else if(date->day > ptr->tm_mday){
                        return 1;
                        }
                        else{
                        return 0;
                        }
        }
        else if (date->month > ptr->tm_mon + 1){
                        return 1;
        }
        else{
                        return 0;
        }
   }
}


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 < buff_table[i] + 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;

	mysql_field_seek(res_set, 0);
	putchar('|');

	for(i = 0; i < mysql_num_fields(res_set); ++i){

		field = mysql_fetch_field(res_set);

		if(field->type == MYSQL_TYPE_TIMESTAMP){
			printf(ANSI_COLOR_BOLD" %-*s "ANSI_COLOR_OFF, (int)(strlen(field-
>name)*3), field->name);
			printf("|");
			buff_table[i] = (int)(strlen(field->name)*3);
		}
```

```
                else{
                        printf(ANSI_COLOR_BOLD" %-*s " ANSI_COLOR_OFF, (int)(strlen(field-
>name)*2), field->name);
                        printf("|");
                        buff_table[i] = (int)(strlen(field->name)*2);
                }
        }
        putchar('\n');
        print_dashes(res_set);
        putchar('\n');
}


void procedure_output(MYSQL *conn, MYSQL_STMT *stmt){

        int i;
        int status;
        int num_fields;                 /*number of columns in result*/
        MYSQL_FIELD *fields;        /*for result set metadata*/
        MYSQL_BIND *rs_bind;        /*for output buffers*/
        MYSQL_RES *rs_metadata, *mysql_res;
        MYSQL_TIME *date;
        MYSQL_TIME *time;
        size_t attr_size;

        bool *is_null; // output value nullability

        if(mysql_stmt_store_result(stmt)){
                fprintf(stderr, "mysql_stmt_execute() 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);
is_null = malloc(sizeof(num_fields));

buff_table = malloc(64*sizeof(num_fields));

do {

        if(num_fields > 0){

                /*there is a result set to fetch*/
                // Get information about the outcome of the stored procedure
                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);

                // Retrieve the fields associated with OUT/INOUT parameters
                fields = mysql_fetch_fields(rs_metadata);
                rs_bind = (MYSQL_BIND *)malloc(sizeof(MYSQL_BIND) * num_fields);

                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_TIMESTAMP:
case MYSQL_TYPE_DATETIME:
case MYSQL_TYPE_DATE:
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
parameters\n\n", true);
                }
        }

        if(mysql_stmt_bind_result(stmt, rs_bind)){
                finish_with_stmt_error(conn, stmt, "Unable to bind output parameters\
n", true);
        }

        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){
        printf(" %-*s |", buff_table[i], "NULL");
        continue;
    }

    switch (rs_bind[i].buffer_type) {

        case MYSQL_TYPE_VAR_STRING:
        case MYSQL_TYPE_DATETIME:
            printf(" %-*s |", buff_table[i],
(char*)rs_bind[i].buffer);
            break;

        case MYSQL_TYPE_DATE:
        case MYSQL_TYPE_TIMESTAMP:
            date = (MYSQL_TIME *)rs_bind[i].buffer;
            if(checkDate(date)){
                printf(ANSI_COLOR_GREEN" %d-
%02d-%02d %.02d:%.02d:%.04d %*s"ANSI_COLOR_RESET,date->year, date->month, date-
>day, date->hour, date->minute, date->second, buff_table[i] -(int)sizeof("0000-00-0000:00:0000"),
"");
                printf("|");
            }
            else{
                printf(ANSI_COLOR_RED" %d-%02d-
%02d %.02d:%.02d:%.04d %*s"ANSI_COLOR_RESET,date->year, date->month, date->day, date-
>hour, date->minute, date->second, buff_table[i] -(int)sizeof("0000-00-0000:00:0000"), "");
                printf("|");
            }
            break;

        case MYSQL_TYPE_STRING:
```

```
                                        printf(" %-*s |", buff_table[i], (char*)
rs_bind[i].buffer);
                                        break;

                                case MYSQL_TYPE_FLOAT:
                                case MYSQL_TYPE_DOUBLE:
                                        printf(" %-*.02f |", buff_table[i], *(float*)
rs_bind[i].buffer);
                                        break;



                                case MYSQL_TYPE_LONG:
                                case MYSQL_TYPE_SHORT:
                                case MYSQL_TYPE_TINY:
                                        printf(" %-*d |", buff_table[i], *(int*)
rs_bind[i].buffer);
                                        break;


                                case MYSQL_TYPE_NEWDECIMAL:
                                        printf(" %-*.021f |", buff_table[i], *(float*)
rs_bind[i].buffer);
                                        break;

                                case MYSQL_TYPE_TIME:
                                        time = (MYSQL_TIME *)rs_bind[i].buffer;
                                        printf("%.02d:%.02d %*s |", time->hour, time-
>minute, buff_table[i]  - (int)sizeof(time->hour) - (int)sizeof(time->minute), "");
                                        break;

                                default:
                                        printf("ERROR: unexpected type (%d)\n",
rs_bind[i].buffer_type);
```

```
                                        abort();
                                }
                        }
                        putchar('\n');
                        print_dashes(rs_metadata);
                }

                mysql_free_result(rs_metadata);        // free metadata

                /*free output buffers*/
                for (i = 0; i < num_fields; ++i){
                        free(rs_bind[i].buffer);
                }

                free(rs_bind);  // free output buffers
        }
        else{
                printf("\n\nNo more output\n");
        }

        if(mysql_stmt_next_result(stmt) > 0){
                finish_with_stmt_error(conn, stmt, "Unable to bind output parameters\n", true);
        }

        } while (mysql_stmt_next_result(stmt) == 0);

        free(is_null);
        free(buff_table);
}
```

inputOutput.c

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

```
#include <string.h>
#include <termios.h>
#include <signal.h>
#include <errno.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);

int getInput(unsigned int lung, char *stringa, bool hide) {

	// Dichiara le variabili necessarie ad un possibile mascheramento dell'input
	struct sigaction sa, savealrm, saveint, savehup, savequit, saveterm;
	struct sigaction savetstp, savettin, savettou;
	struct termios term, oterm;

	if(hide) {
		// Svuota il buffer
		(void) fflush(stdout);

		// Cattura i segnali che altrimenti potrebbero far terminare il programma, lasciando l'utente senza output sulla shell
		sigemptyset(&sa.sa_mask);
		sa.sa_flags = SA_INTERRUPT; // Per non resettare le system call
		sa.sa_handler = handler;
		(void) sigaction(SIGALRM, &sa, &savealrm);
		(void) sigaction(SIGINT, &sa, &saveint);
		(void) sigaction(SIGHUP, &sa, &savehup);
		(void) sigaction(SIGQUIT, &sa, &savequit);
		(void) sigaction(SIGTERM, &sa, &saveterm);
```

```
        (void) sigaction(SIGTSTP, &sa, &savetstp);
        (void) sigaction(SIGTTIN, &sa, &savettin);
        (void) sigaction(SIGTTOU, &sa, &savettou);

        // 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
char c;
unsigned int i;

for(i = 0; i < lung; i++) {
        int size = fread(&c, sizeof(char), 1, stdin);
        if(size == 0 && errno == EINTR){
                if(hide){
                        (void) tcsetattr(fileno(stdin), TCSAFLUSH, &oterm);

                        // Ripristina la gestione dei segnali
                        (void) sigaction(SIGALRM, &savealrm, NULL);
                        (void) sigaction(SIGINT, &saveint, NULL);
                        (void) sigaction(SIGHUP, &savehup, NULL);
                        (void) sigaction(SIGQUIT, &savequit, NULL);
                        (void) sigaction(SIGTERM, &saveterm, NULL);
                        (void) sigaction(SIGTSTP, &savetstp, NULL);
                        (void) sigaction(SIGTTIN, &savettin, NULL);
                        (void) sigaction(SIGTTOU, &savettou, NULL);
```

```
                }
                return -1;
        }
        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));
        }
}

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

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

if(hide) {
        //L'a capo dopo l'input
        (void) write(fileno(stdout), "\n", 1);
```

```
		// Ripristina le impostazioni precedenti dello schermo
		(void) tcsetattr(fileno(stdin), TCSAFLUSH, &oterm);

		// Ripristina la gestione dei segnali
		(void) sigaction(SIGALRM, &savealrm, NULL);
		(void) sigaction(SIGINT, &saveint, NULL);
		(void) sigaction(SIGHUP, &savehup, NULL);
		(void) sigaction(SIGQUIT, &savequit, NULL);
		(void) sigaction(SIGTERM, &saveterm, NULL);
		(void) sigaction(SIGTSTP, &savetstp, NULL);
		(void) sigaction(SIGTTIN, &savettin, NULL);
		(void) sigaction(SIGTTOU, &savettou, NULL);

		// Se era stato ricevuto un segnale viene rilanciato al processo stesso
		if(signo)
			(void) raise(signo);
	}

	return strlen(stringa);
}

// Per la gestione dei segnali
static void handler(int s) {
	signo = s;
}
```

parseJson.c

```
#include<stdio.h>
#include<json-c/json.h>

const char* getJson(char *user, char *record){

	FILE *fp;
	char buffer[1024];
```

```
        struct json_object *parsed_json;
        struct json_object *jObj;
        struct json_object *result;

        fp = fopen("test.json","r");
        fread(buffer, 1024, 1, fp);
        fclose(fp);

        parsed_json = json_tokener_parse(buffer);

        jObj = json_object_object_get(parsed_json, user);
        int count = json_object_array_length(jObj);

        json_object *element = json_object_array_get_idx(jObj, 0);
        result = json_object_object_get(element, record);

        return json_object_get_string(result);
}
```

test.json

```
{
        "login":[
                {
                "host": "localhost",
                "username": "login",
                "password": "Login!00",
                "port": "3306",
                "database": "sistemaAsteOnline"
                }],
        "admin":[
                {
                "host": "localhost",
                "username": "utenteamm",
                "password": "Utenteamm!0",
```

```
                "port": "3306",
                "database": "sistemaAsteOnline"
                }],
        "standard":[
                {
                "host": "localhost",
                "username": "utentebase",
                "password": "Utentebase!0",
                "port": "3306",
                "database": "sistemaAsteOnline"
                }]
}
```

defines.h

```
#include <mysql.h>
#include <stdbool.h>

#define HOST "host"
#define USERNAME "username"
#define PASSWORD "password"
#define DATABASE "database"
#define PORT "port"

#define ANSI_COLOR_RED          "\x1b[31m"
#define ANSI_COLOR_GREEN        "\e[0;32m"
#define ANSI_COLOR_RESET        "\x1b[0m"
#define ANSI_COLOR_BOLD         "\e[1m"
#define ANSI_COLOR_OFF          "\e[m"


typedef struct{
        char cf[16];
} UTENTE;
```

```
int getInput(unsigned int lung, char *string, bool hide);
void admin(MYSQL *conn, char *username);
void standard(MYSQL *conn, char *username);
extern void print_stmt_error(MYSQL_STMT *stmt, char *message);
extern void finish_with_stmt_error(MYSQL *conn, MYSQL_STMT *stmt, char *message, bool
lcose_stmt);
extern bool setup_prepared_stmt(MYSQL_STMT **stmt, char *statement, MYSQL *conn);
extern void procedure_output(MYSQL *conn, MYSQL_STMT *stmt);
extern void op_3(MYSQL *conn);
extern void op_a1(MYSQL *conn);
const char* getJson(char *user, char *record);
```