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

Basi di Dati

Progetto A.A. 2021/2022

SISTEMA DI ASTE ONLINE

0253129

Daniele d’Errico

**Indice**

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

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

[3. Progettazione concettuale 5](#_Toc2081466291)

[4. Progettazione logica 6](#_Toc2147004904)

[5. Progettazione fisica 8](#_Toc518560220)

[Appendice: Implementazione 9](#_Toc403811585)

# Descrizione del Minimondo

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49 | 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. |

# 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. |

# 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.

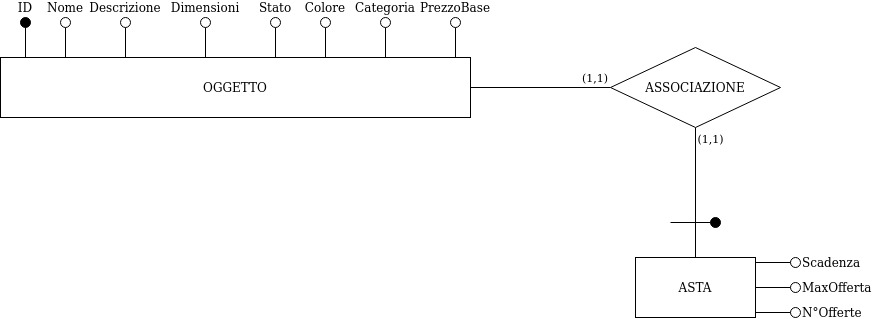
Si può definire l’attributo “scadenza” per l’entità Asta, in questo modo, al momento della pubblicazione, da parte di un Utente Amministratore, di un Oggetto da mettere all’Asta, verrà stabilito l’istante temporale della scadenza dell’Asta.

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.

1. L’Utente Amministratore espone uno o più Oggetti, ognuno di essi implica l’inizio del ciclo di vita di un’asta.
2. 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.

1. L’Utente Base fa le sue offerte, in qualsiasi momento, su qualsiasi Asta disponibile nel Sistema.
2. 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.

1. 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

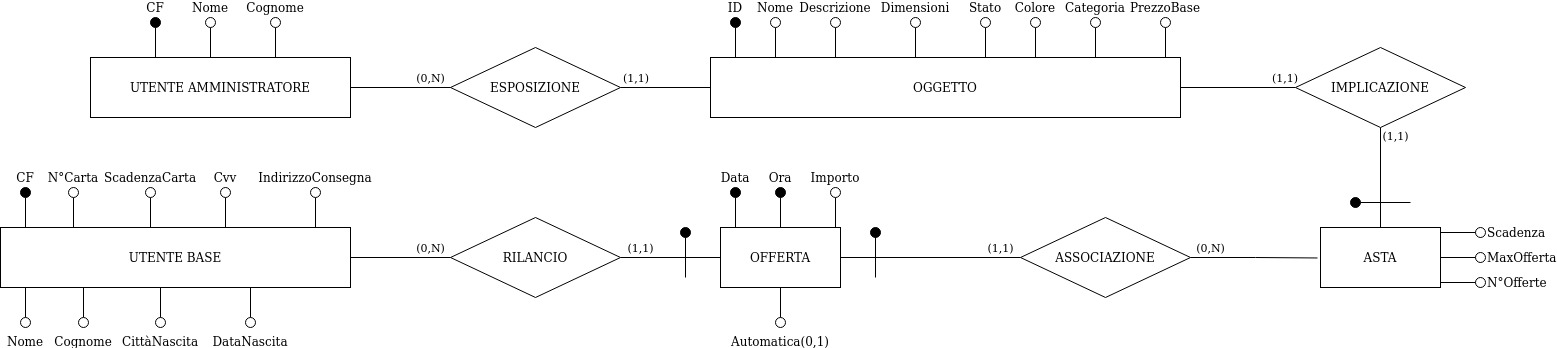
aste disponibili.

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

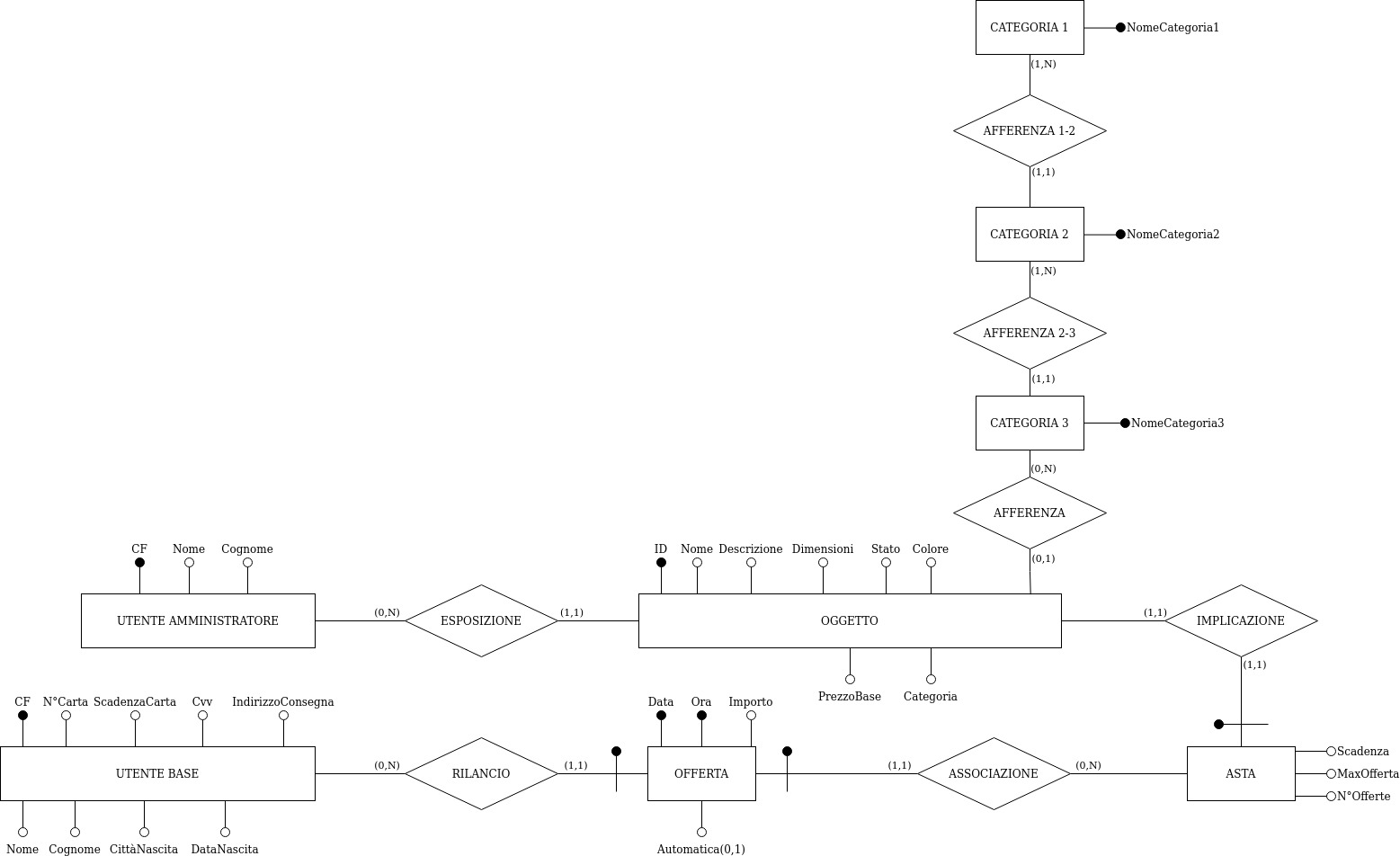
schematizzato come un attributo opzionale (“Automatica”) dell’entità Offerta.



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 |

# 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]](#footnote-2)** | **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 |

## 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.

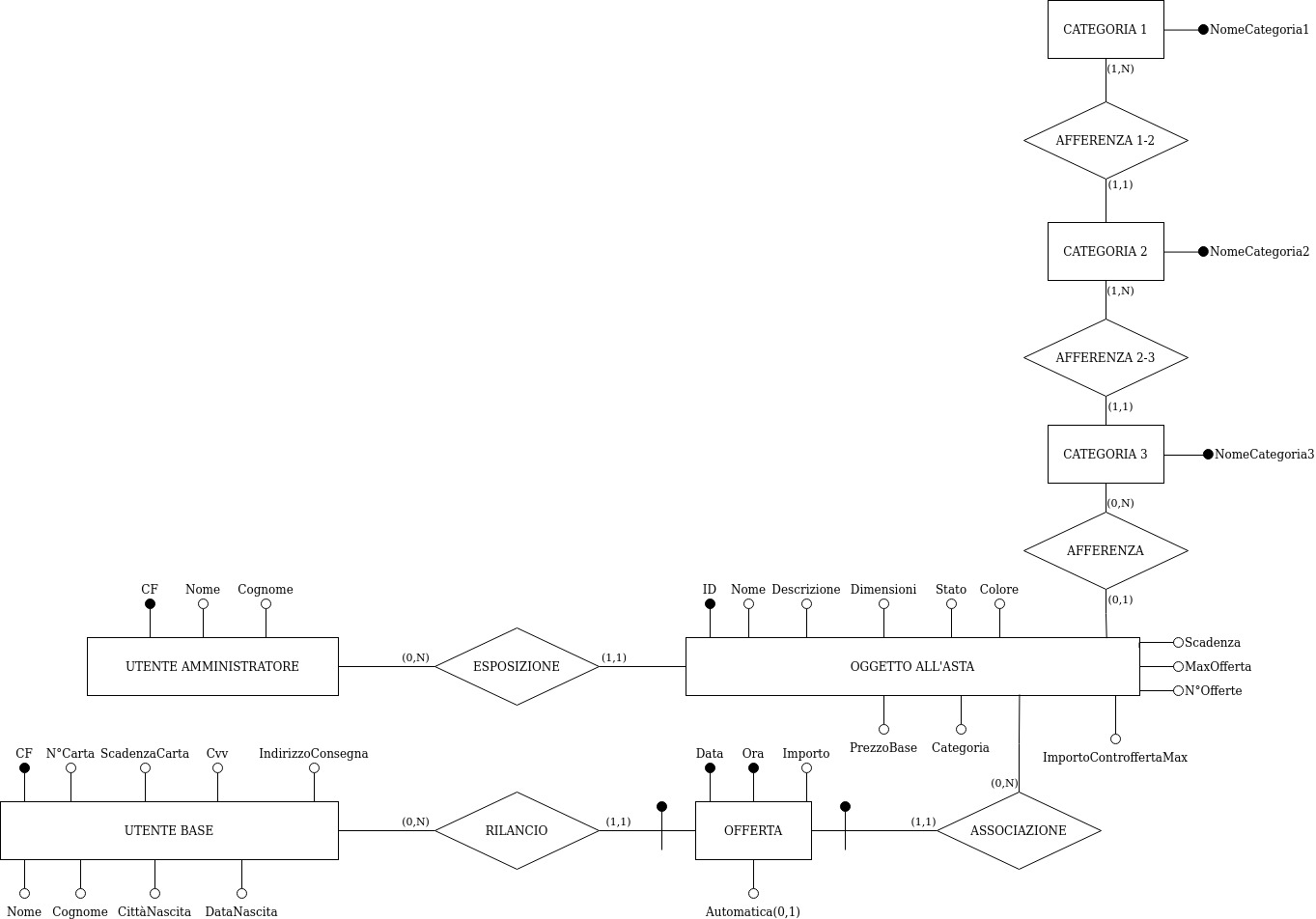
Inoltre l’Applicazione lavora molto in lettura e quindi risulterebbe poco funzionale mantenere una relazione 1 a 1 che causerebbe una duplicazione degli accessi per recuperare informazioni che invece potrebbro essere unificate senza problemi.

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).

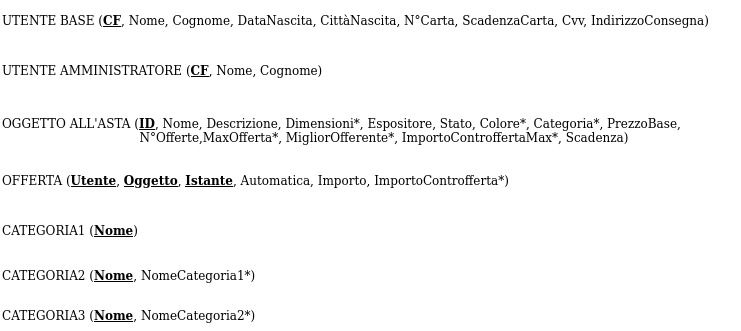
Inoltre, è da notare che, considerando un modello relazionale dei dati, il doppio identificatore esterno

implica che la relazione Offerta rappresenterà obbligatoriamente anche le associazioni “Rilancio” e

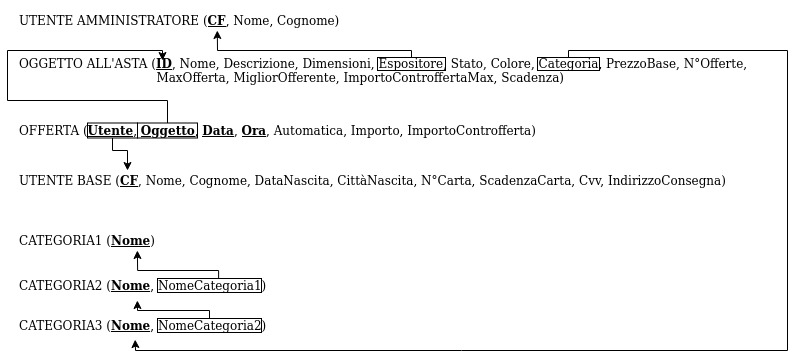
“Associazione”, che non avranno dunque una propria rappresentazione tabellare.

## Traduzione di entità e associazioni

Schema relazionale:



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.

# 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).

1. 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]](#footnote-3)** |
| **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]](#footnote-4)** |
| **CF** | CHAR(16) | PK, NN |
| **Nome** | VARCHAR(20) | NN |
| **Cognome** | VARCHAR(20) | NN |

|  |  |  |
| --- | --- | --- |
| **Tabella OFFERTA** | | |
| **Colonna** | **Tipo di dato** | **Attributi[[4]](#footnote-5)** |
| **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]](#footnote-6)** |
| **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]](#footnote-7)** |
| **Nome** | VARCHAR(20) | PK, NN |
| **NomeCategoria2** | VARCHAR(20) |  |

|  |  |  |
| --- | --- | --- |
| **Tabella CATEGORIA2** | | |
| **Colonna** | **Tipo di dato** | **Attributi[[7]](#footnote-8)** |
| **Nome** | VARCHAR(20) | PK, NN |
| **NomeCategoria1** | VARCHAR(20) |  |

|  |  |  |
| --- | --- | --- |
| **Tabella CATEGORIA1** | | |
| **Colonna** | **Tipo di dato** | **Attributi[[8]](#footnote-9)** |
| **Nome** | VARCHAR(20) | PK, NN |

## 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]](#footnote-10): 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**

## 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,NO\_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);

1. Indicare con E le entità, con R le relazioni [↑](#footnote-ref-2)
2. PK = primary key, NN = not null, UQ = unique, UN = unsigned, AI = auto increment. È ovviamente possibile specificare più di un attributo per ciascuna colonna. [↑](#footnote-ref-3)
3. PK = primary key, NN = not null, UQ = unique, UN = unsigned, AI = auto increment. È ovviamente possibile specificare più di un attributo per ciascuna colonna. [↑](#footnote-ref-4)
4. PK = primary key, NN = not null, UQ = unique, UN = unsigned, AI = auto increment. È ovviamente possibile specificare più di un attributo per ciascuna colonna. [↑](#footnote-ref-5)
5. PK = primary key, NN = not null, UQ = unique, UN = unsigned, AI = auto increment. È ovviamente possibile specificare più di un attributo per ciascuna colonna. [↑](#footnote-ref-6)
6. PK = primary key, NN = not null, UQ = unique, UN = unsigned, AI = auto increment. È ovviamente possibile specificare più di un attributo per ciascuna colonna. [↑](#footnote-ref-7)
7. PK = primary key, NN = not null, UQ = unique, UN = unsigned, AI = auto increment. È ovviamente possibile specificare più di un attributo per ciascuna colonna. [↑](#footnote-ref-8)
8. PK = primary key, NN = not null, UQ = unique, UN = unsigned, AI = auto increment. È ovviamente possibile specificare più di un attributo per ciascuna colonna. [↑](#footnote-ref-9)
9. IDX = index, UQ = unique, FT = full text, PR = primary. [↑](#footnote-ref-10)