



Basi di Dati e Conoscenza

Progetto A.A. 2018/2019

## SISTEMA DI ASTE ONLINE

0218205

Gianmarco Bencivenni

### Indice

<b>1. Descrizione del Minimondo.....</b>	<b>2</b>
<b>2. Analisi dei Requisiti .....</b>	<b>4</b>
<b>3. Progettazione concettuale.....</b>	<b>10</b>
<b>4. Progettazione logica .....</b>	<b>19</b>
<b>5. Progettazione fisica .....</b>	<b>33</b>
<b>Appendice: Implementazione .....</b>	<b>46</b>

## 1. Descrizione del Minimondo

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

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

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

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

## 2. Analisi dei Requisiti

### Identificazione dei termini ambigui e correzioni possibili

Linea	Termine	Nuovo termine	Motivo correzione
2	Amministratore	Utente Amministratore	Il concetto di Amministratore è un'estensione di quello di Utente, pertanto si ritiene opportuno ridefinire entrambi i termini per meglio visualizzare la linea gerarchica diretta tra essi.
3	Utente del sistema	Utente Base	Si vuole sottolineare il fatto che un Utente, rispetto ad un Amministratore, può accedere a una più stretta gamma di servizi.
8	Indirizzo cui consegnare eventuali oggetti acquistati	Indirizzo di consegna	Si esprime lo stesso concetto in forma più compatta e semplice.
13	Decidere la durata dell'asta	Impostare la durata dell'asta	Il nuovo termine è meno ambiguo e più tecnico.
18	Aste aperte	Aste attive	Il nuovo termine è meno ambiguo e più tecnico.
10	Codice alfanumerico univoco	Identificatore	Si esprime lo stesso concetto in forma più compatta e semplice.
4	L'offerta maggiore	L'offerta massima	Terminologia più precisa
40	Asta in corso	Asta attiva	Il nuovo termine è meno ambiguo e più tecnico.
23	Valore attuale di offerta	Offerta massima attuale	Terminologia più precisa

### Specifiche disambiguata

Una casa d'aste intende realizzare un sistema di aste online. Il sistema deve consentire agli Utenti Amministratori la gestione degli oggetti che si vogliono pubblicare e di tutto il ciclo di vita delle aste. Gli Utenti Base del sistema, previa registrazione, hanno la possibilità di fare offerte su un qualsiasi oggetto disponibile. Al termine dell'asta, l'offerta massima 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), l'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, corrispondente a quella dell'Oggetto d'interesse. 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 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. Gli Utenti Base non possono però visualizzare chi è che ha effettuato l'offerta massima.

Dato un oggetto in asta, gli Utenti Base 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 Base che ha attualmente piazzato l'offerta massima, può sfruttare la funzionalità di "controfferta automatica". Tale funzionalità permette all'Utente Base 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  $I + 0,50\text{€}$ .

Il sistema tiene traccia, per ogni oggetto, di tutte le offerte che sono state fatte e dell'istante 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 Base, 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.

## Glossario dei Termini

Termine	Descrizione	Sinonimi	Collegamenti
Utente Base	È l'attore che si interfaccia al sistema per concorrere alle aste.	Utente, Utente del Sistema	Oggetto, Asta, Offerta, Controfferta automatica
Utente Amministratore	Si occupa della gestione degli oggetti che vuole pubblicare, e del ciclo di vita delle Aste.	Amministratore, Amministratore del Sistema	Oggetto, Asta, Categoria,
Oggetto	È ciò che l'Utente Base intende acquistare partecipando ad un'asta.		Asta, Utente Amministratore, Utente Base
Asta	È una competizione tra Utenti Base, i quali cercano di aggiudicarsi un Oggetto di interesse, rilanciando sull'offerta massima attuale.		Utente Base, Utente Amministratore, Oggetto, Offerta, Controfferta automatica
Offerta	È la somma di 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.		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 documento che mostri lo storico delle offerte per un Oggetto, distinguendo le offerte dalle “controfferte automatiche”.

**Frase 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'istante temporale in cui queste sono state inserite nel sistema.

**Frase 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

**Frase relative all' Offerta**

Un'Offerta è la somma di 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.

La granularità di 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  $I + 0,50\text{€}$ .

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

#### **Frasi relative alla Controfferta Automatica**

Automatizzazione del concetto di Offerta.

La funzionalità di Controfferta Automatica permette all'Utente Base di indicare un importo massimo con cui si intende rilanciare l'offerta, qualora un altro utente faccia un'offerta maggiore.

Tutte le transazioni automatiche generate dal sistema di controfferta automatica devono essere registrate nel sistema.

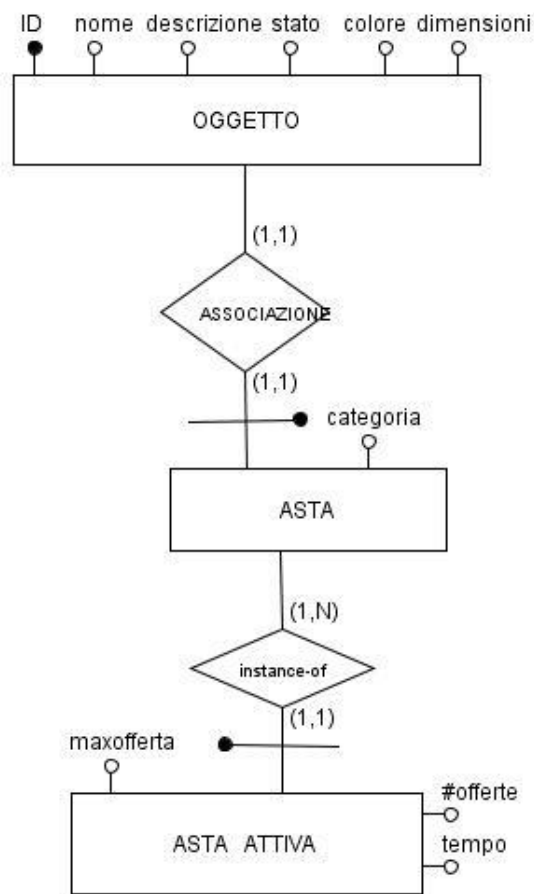
#### **Frasi relative alla Categoria**

A ciascuna asta viene associata una categoria, quella dell'Oggetto all'asta.

Le categorie appartengono ad un titolario gerarchico, organizzato su un massimo di tre livelli.

### 3. Progettazione concettuale

#### Costruzione dello schema E-R



#### STEP 1

Per la schematizzazione E-R si segue un approccio inside-out.

L'idea è quella di partire dal nucleo del Sistema e costruire un puzzle che rappresenti lo scheletro dell'applicazione.

Il concetto principale attorno cui ruota il Sistema è di fatto quello di Asta.

Ciò che questa prima bozza vuole rappresentare è la corrispondenza biunivoca tra l'asta e l'oggetto per cui essa viene indetta: non può esistere, sostanzialmente, un'asta non associata ad un oggetto, e ogni oggetto viene pubblicato online per essere messo all'asta.

Da questa assunzione nasce la necessità di porre un'asta come entità debole, identificata dall'oggetto associato.

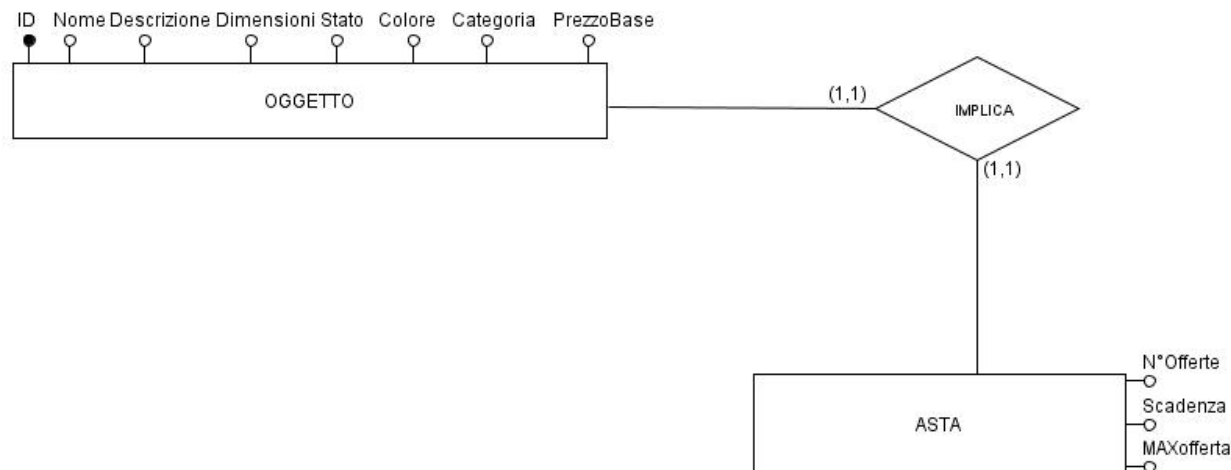
Il Design Pattern "instance-of" definisce il concetto di "Asta Attiva".

Al momento della pubblicazione dell'Oggetto, questo dovrà esser messo all'asta dall'Utente Amministratore: da quel momento l'asta risulterà attiva, in attesa di offerte da parte degli Utenti, per un periodo di tempo limitato.

L'Asta Attiva rappresenta l'istanza dell'Asta, ma ampliando lo schema notiamo immediatamente come questa rappresentazione sia in realtà superflua.

## STEP 2

L'Attributo "tempo" dell'entità Asta Attiva può essere modificato, o meglio specificato, in "scadenza".



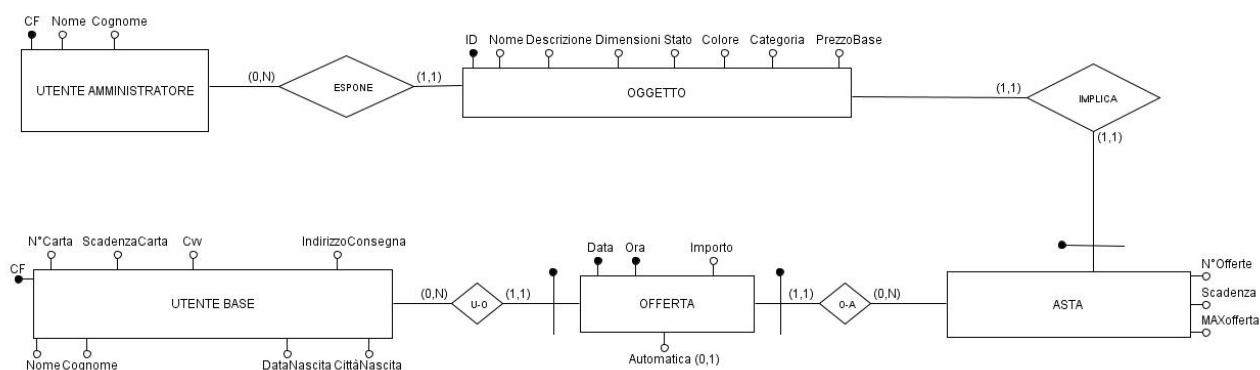
In questo modo, al momento della pubblicazione, da parte di un Utente Amministratore, di un Oggetto da mettere all'Asta, verrà stabilito l'istante temporale della scadenza dell'Asta, che automaticamente nasce come attiva.

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 3

La semplificazione viene apprezzata quando lo schema si espande nella direzione degli attori principali del Sistema, ovvero l'Utente Base e l'Utente Amministratore.



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

Senza scrivere questo vincolo, sarebbe opportuno storicizzare il concetto di Asta, aggiungendo una generalizzazione totale che divida le aste in Aste Attive e Aste concluse: il problema è che il concetto di "Asta Conclusa" non aggiunge ulteriori informazioni rispetto al generico concetto di Asta.

- e) L'Offerta è un'entità "molto debole", in quanto risulta definita in riferimento all'Utente Base e all'Oggetto di interesse, oltre che dall'istante temporale in cui viene lanciata.

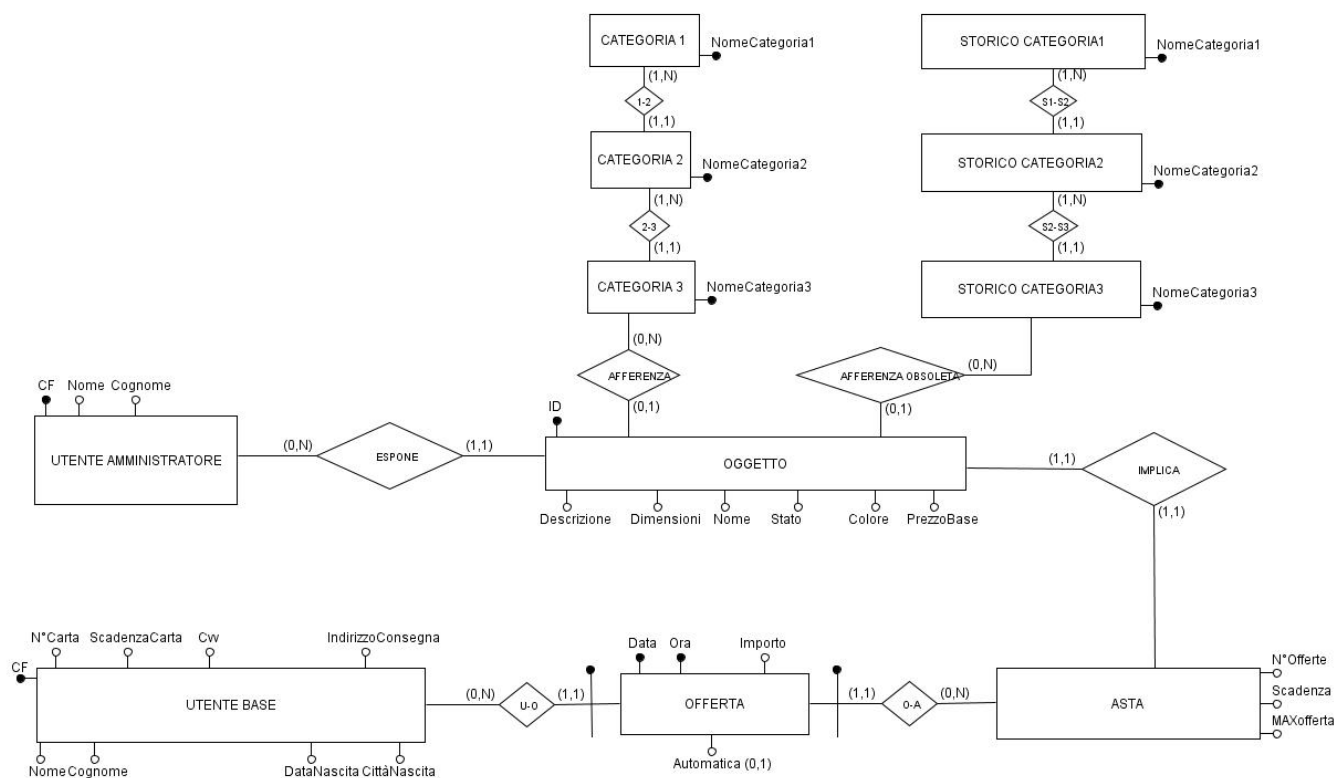
L'Offerta non può essere intesa come una relazione binaria tra Utente Base ed Asta, poiché ciò implicherebbe che ogni Utente avrebbe a disposizione una sola offerta per ognuna delle aste disponibili. È stato dunque applicato in questo caso un design pattern di "reificazione di relazione", essendo per giunta l'Offerta un concetto ricco di attributi caratterizzanti.

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

#### STEP 4

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

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



A tal proposito, è necessario mantenere uno storico delle categorie, in modo da evitare problemi di inconsistenza:

se un Amministratore elimina una categoria dalle categorie correnti bisogna fare in modo che gli oggetti afferenti a tale categoria possano mantenere l'attributo inalterato, utilizzando quindi lo Storico delle Categorie, attraverso la relazione di Afferenza Obsoleta.

Si noti che il titolare contenente le categorie correnti è sempre strettamente contenuto nello storico del titolare.

## Integrazione finale

Prima di procedere con la fase di progettazione logica e la conseguente ristrutturazione del diagramma E-R prodotto in questa fase, è necessario testare i requisiti di buona progettazione dello schema, quali correttezza, completezza, leggibilità e minimalità.

a) **Correttezza:**




Lo schema è sintatticamente e semanticamente corretto.

b) **Completezza:**

- Attraverso un'analisi sulla navigabilità del diagramma si è potuta riscontrare una mancanza importante riguardo la rappresentazione della funzionalità di controfferta automatica, componente essenziale del sistema in esame.

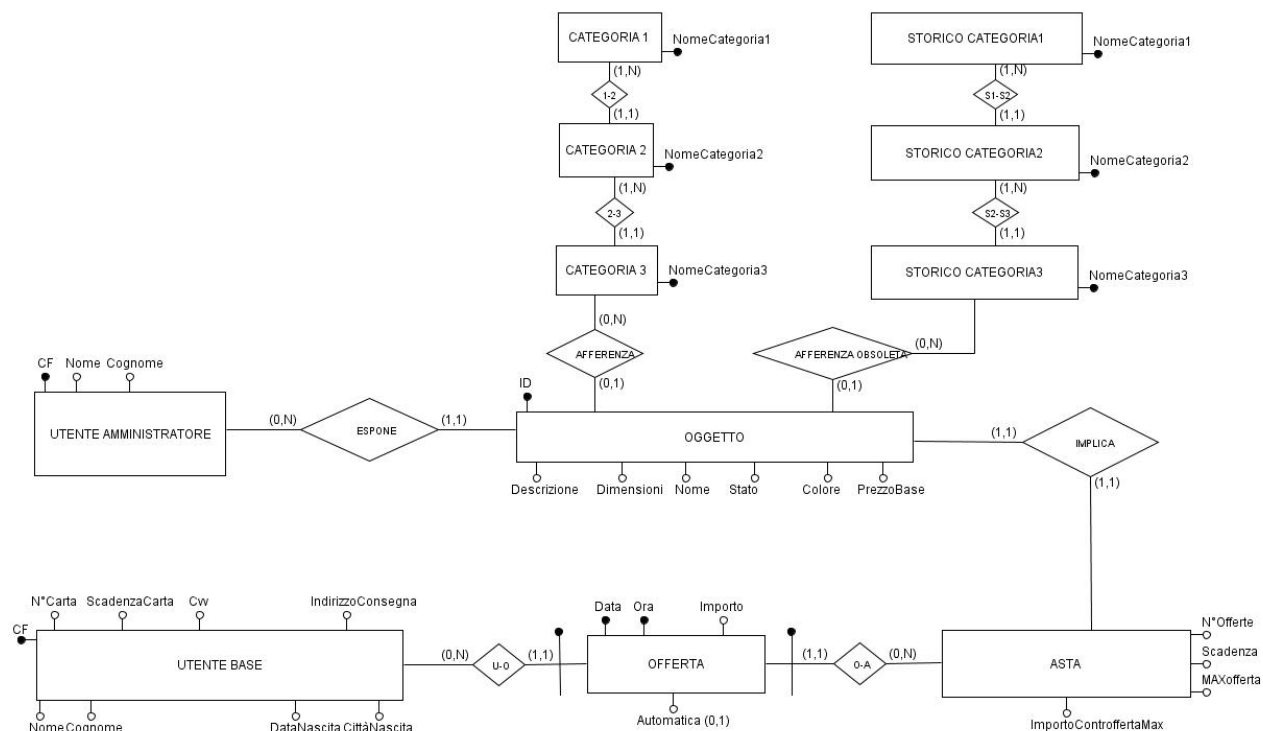
Per via di una simulazione di uno scenario possibile, si spiega come è necessario completare lo schema:

*Si consideri un'occorrenza di Asta, in cui due Utenti Base concorrono ad un buono per un pranzo in un pub. Nella tabella seguente si riporta l'evoluzione temporale dell'asta, in particolare si descrive il funzionamento dell'offerta e della controfferta automatica (CA).*

TEMPO	 UTENTE BASE A	 ASTA (Prezzo base = 2,00 euro)	 UTENTE BASE B
T0		2,00 euro	
T1	Offre 1,00 euro	3,00 euro (A)	
T2	CA = 2,00 euro	3,00 euro (A)	
T3		3,50 euro (B)	Offre 0,50 euro
T4		( <i>AUTO</i> : 2,00 > 0,50) 4,00 euro (A) ( <i>CA</i> = 1,50 euro)	
T5		5,00 euro (B)	Offre 1,00 euro
T6		( <i>AUTO</i> : 1,50 > 1,00) 5,50 euro (A) ( <i>CA</i> = 0,50 euro => <i>NULL</i> )	
T7		6,00 euro (B)	Offre 0,50 euro
T8		6,00 euro (B)	CA = 3,00 euro

La tabella fa capire come, per ogni asta, in ogni momento, possa esistere al più una Controfferta Automatica, qualora sia impostata dal miglior offerente attuale.

A tal proposito, è necessario aggiungere l'attributo `ImportoControffertaMax` sull'entità Asta.



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:

- Registrazione, tra le offerte dell'Utente in vantaggio, l'offerta generata dalla funzionalità di controfferta automatica, segnalata dalla partecipazione dell'attributo "`Automatica`" sull'entità Offerta
- Mantenere lo stato del Miglior Offerente, aggiornando il valore dell'importo massimo di controfferta.

In questo modo la condizione di completezza è soddisfatta.

**c) Leggibilità:**

Sicuramente da migliorare è la leggibilità: le associazioni “espone” e “implica” sono scritte come predicati, il che rende lo schema leggibile solo da un verso.

È opportuno modificare come segue:

- ESPONE → ESPOSIZIONE
- IMPLICA → IMPLICAZIONE

Le relazioni “U-O” e “O-A” sono due facce della stessa medaglia: utilizzando il pattern di reificazione di relazione per il concetto di offerta, queste associazioni nascono automaticamente. Si potrebbero riscrivere come:

- U-O → RILANCIO (Un Utente esegue un rilancio attraverso un’offerta)
- O-A → ASSOCIAZIONE

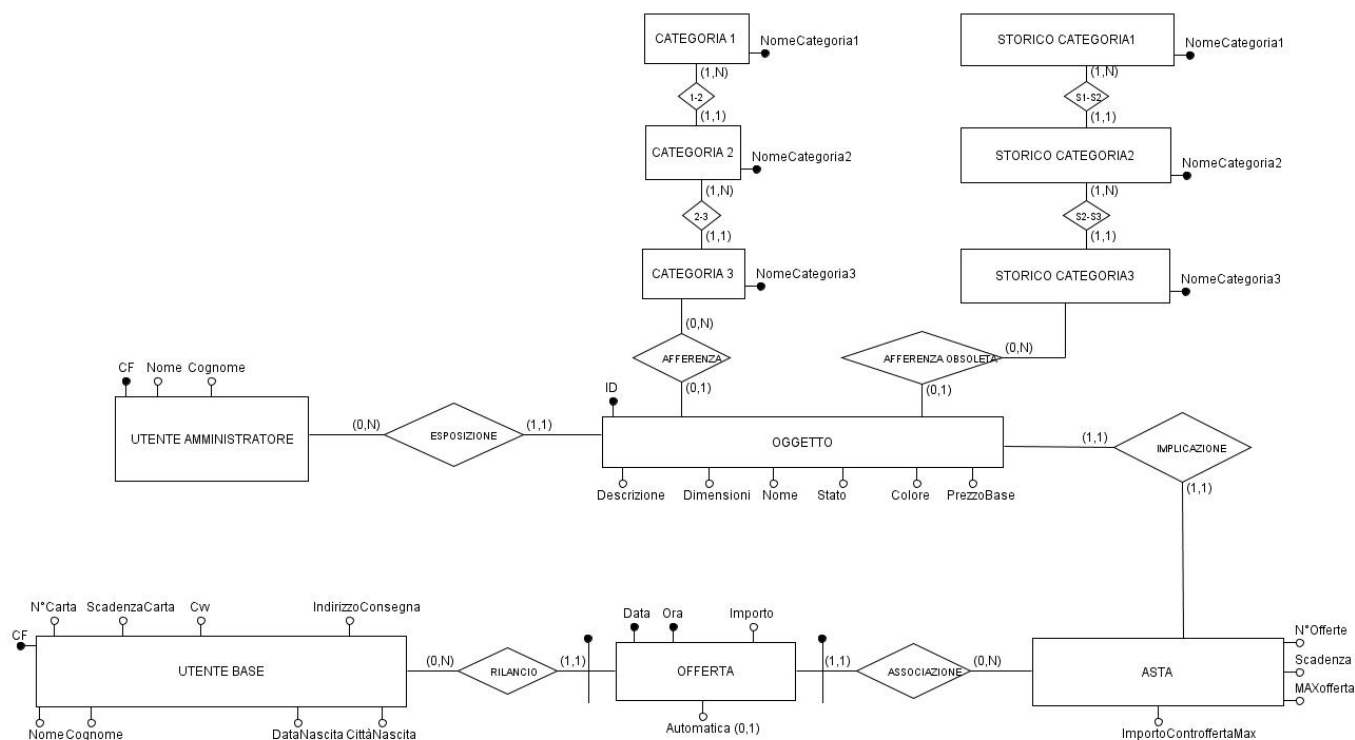
**d) Minimalità:**

Il fatto di mantenere uno storico delle categorie è ovviamente una forte ridondanza di dati, in quanto si tratta di una copia esatta delle categorie correnti, che però mantiene le informazioni passate, su cui possono essere state applicate eliminazioni e aggiornamenti.

Tuttavia, come sarà discusso nella fase di progettazione logica, si può scegliere di tenere questa ridondanza come “cuscinetto” per eventuali anomalie, senza sovraccaricare eccessivamente il sistema.



Per il resto, lo schema risulta apparentemente minimale e privo di ridondanze.



## Regole aziendali

È necessario aggiungere qualche asserzione affinché il diagramma risulti corretto:

- 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,	CF

		CittàNascita, IndirizzoConsegna, N°Carta, ScadenzaCarta, Cvv	
UTENTE AMMINISTRATORE		CF, Nome, Cognome, DataNascita, CittàNascita, IndirizzoConsegna, N°Carta, ScadenzaCarta, Cvv	CF
OGGETTO		ID, Nome, Descrizione, Dimensioni, Stato, Colore, Categoria, PrezzoBase	ID
ASTA		N°Offerte, TempoRestante, MAXofferta	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	NomeCategoria3
STORICO CATEGORIA1		NomeCategoria1	NomeCategoria1
STORICO CATEGORIA2		NomeCategoria2	NomeCategoria2
STORICO CATEGORIA3		NomeCategoria3	NomeCategoria3

## 4. Progettazione logica

### Volume dei dati

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

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

Concetto nello schema	Tipo <sup>1</sup>	Volume atteso “a regime”	Volume atteso V(t)
UTENTE AMMINISTRATORE	E	20	$O(\log t)$
UTENTE BASE	E	400	$O(\log t)$
ASTA	E	400	$O(t)$
OFFERTA	E	160.000	$O(t^2)$
OGGETTO	E	400	$O(t)$
MIGLIOR OFFERENTE	E	400	$O(\log t)$
ESPOSIZIONE	R	400	$O(t)$
IMPLICAZIONE	R	400	$O(t)$
RILANCIO	R	160.000	$O(t^2)$
ASSOCIAZIONE	R	160.000	$O(t^2)$
CATEGORIA1	E	6	~ COST
CATEGORIA2	E	20	~ COST

<sup>1</sup> Indicare con E le entità, con R le relazioni

CATEGORIA3	E	80	~ COST
STORICO CATEGORIA1	E	12	~ COST
STORICO CATEGORIA2	E	40	~ COST
STORICO CATEGORIA3	E	150	~ COST

### Tavola delle operazioni

Cod.	Descrizione	Frequenza attesa
1	Registra un nuovo Utente nel Sistema	1 a settimana
2	Elimina un Utente dal Sistema	1 ogni due mesi
3	Login	100 al giorno
4	Log out	100 al giorno
5	Inserisci un nuovo Oggetto nel Sistema	10 al giorno
6	Inizializza l'Asta per un nuovo Oggetto	10 al giorno
7	Inserisci nuova Categoria	1 ogni due mesi
8	Elimina Categoria	1 ogni due mesi
9	Aggiorna Categoria	1 ogni due mesi
10	Visualizza tutte le Aste attive	150 al giorno
11	Visualizza le Aste attive associate al nome di un Oggetto	500 al giorno
12	Visualizza le Aste attive associate ad una Categoria	200 al giorno
13	Registra l'Offerta di un Utente per un'Asta	300 al giorno
14	Visualizza le Aste indette da un Utente Amministratore	20 al giorno
15	Trova tutte le Offerte fatte in un'Asta	50 al giorno
16	Trova tutte le Offerte generate in un'Asta dal sistema di controfferta automatica	20 al giorno
17	Visualizza lo stato di un'Asta	500 al giorno
18	Visualizza tutti gli Oggetti aggiudicati da un Utente Base	100 al giorno
19	Trova tutte le Aste attive in cui un Utente ha lanciato almeno un'Offerta	300 al giorno

*È opportuno fare una considerazione sulle tavole dei volumi e delle operazioni:*

*Per quanto si possano stimare i volumi di un'applicazione, questa tipologia di Sistema prevede una continua espansione, e i volumi e le grandezze derivate dovrebbero esser scritti in funzione del tempo:*

- *nuovi Utenti, seppur lentamente, continueranno a registrarsi;*
- *nuove Aste saranno generalmente indette dagli Utenti Amministratori;*
- *sempre più Offerte andranno ad aggiungersi al volume delle Offerte (proprio questo concetto sarà quello di maggiore gravosità per il Sistema!)*

### **Costo delle operazioni**

Per stabilire i costi delle operazioni è utile riportare, per ognuna di esse, una tabella degli accessi, specificando la tipologia dell'accesso (in lettura e in scrittura).

Operazione 1: Registra un nuovo Utente nel Sistema

Concetto nello schema	Costrutto	Accessi	Tipo
UTENTE BASE / UTENTE AMMINISTRATORE	E	1	S

Operazione 2: Elimina un Utente dal Sistema

Concetto nello schema	Costrutto	Accessi	Tipo
UTENTE BASE/ UTENTE AMMINISTRATORE	E	1	L

Operazione 3: Login

Concetto nello schema	Costrutto	Accessi	Tipo
UTENTE BASE/ UTENTE AMMINISTRATORE	E	1	L

Operazione 4: Log Out

Concetto nello schema	Costrutto	Accessi	Tipo
/	/	0	/

Operazione 5: Inserisci un nuovo Oggetto nel Sistema

Concetto nello schema	Costrutto	Accessi	Tipo
UTENTE AMMINISTRATORE	E	1	L
OGGETTO	E	1	S

Operazione 6: Inizializza l'Asta per un nuovo Oggetto

Concetto nello schema	Costrutto	Accessi	Tipo
ASTA	E	1	S
OGGETTO	E	1	L
ESPOSIZIONE	R	1	L

Operazione 7: Inserisci Nuova Categoria

Concetto nello schema	Costrutto	Accessi	Tipo
CATEGORIA3	E	1	S
CATEGORIA2	E	1	L
STORICOCATEGORIA3	E	1	S
STORICOCATEGORIA2	E	1	L

Operazione 8: Elimina Categoria

Concetto nello schema	Costrutto	Accessi	Tipo
CATEGORIA3 (/ 2 / 1)	E	1	L

Operazione 9: Aggiorna Categoria

Concetto nello schema	Costrutto	Accessi	Tipo
CATEGORIA3 (/ 2 / 1)	E	1	S

Operazione 10: Visualizza tutte le Aste attive

*Considerando il numero di Aste attive come 2/5 delle Aste*

Concetto nello schema	Costrutto	Accessi	Tipo
ASTA	E	160	L

Operazione 11: Trova le Aste associate al nome di un Oggetto

*Considerando, in media (ma per eccesso) la presenza di 3 omonimi per ogni oggetto*

Concetto nello schema	Costrutto	Accessi	Tipo
ASTA	E	3	L
IMPLICAZIONE	R	3	L
OGGETTO	E	3	L

Operazione 12: Trova tutte le Aste associate ad una categoria

*Considerando l'ipotesi in cui gli oggetti sono suddivise in una media di 5 categorie*

Concetto nello schema	Costrutto	Accessi	Tipo
ASTA	E	80	L
OGGETTO	E	80	L
IMPLICAZIONE	R	80	L

Operazione 13: Registra l'Offerta di un Utente per un'asta

Concetto nello schema	Costrutto	Accessi	Tipo
OFFERTA	E	1	S
RILANCIO	R	1	S
ASSOCIAZIONE	R	1	S
VANTAGGIO	R	1	S

MIGLIOR OFFERENTE	E	1	S
-------------------	---	---	---

Operazione 14: Trova tutte le Aste indette da un Utente Amministratore

Concetto nello schema	Costrutto	Accessi	Tipo
UTENTE AMMINISTRATORE	E	1	L
ASTA	E	20	L
ESPOSIZIONE	R	20	L
IMPLICAZIONE	R	20	L

Operazione 15: Trova tutte le Offerte fatte in un'Asta

*Considerando che in media ogni Asta riceve 1 offerta per ogni Utente (ed è anche tanto!)*

Concetto nello schema	Costrutto	Accessi	Tipo
ASTA	E	1	L
OFFERTA	E	400	L
ASSOCIAZIONE	R	400	L

Operazione 16: Trova tutte le Offerte generate in un'Asta dal sistema di controfferta automatica

*Considerando che in media il venti per cento delle offerte sono generate dalla funzionalità di controfferta automatica*

Concetto nello schema	Costrutto	Accessi	Tipo
ASTA	E	1	L
OFFERTA	E	80	L
ASSOCIAZIONE	R	80	L

Operazione 17: Visualizza lo stato di un'Asta



Concetto nello schema	Costrutto	Accessi	Tipo
ASTA	E	1	L

Operazione 18: Visualizza tutti gli Oggetti aggiudicati da un Utente Base

*Considerando che:*

- A) la quantità di Oggetti aggiudicati è la stessa delle Aste concluse vinte da uno stesso Utente;
- B) il numero di Aste concluse è pari ai 3/5 delle Aste totali (240)
- C) in media ogni Utente si sia aggiudicato  $240/400 = 0,6$  oggetti ~ 1 oggetto

Concetto nello schema	Costrutto	Accessi	Tipo
OGGETTO	E	1	L
VANTAGGIO	R	1	L

Operazione 19: Trova tutte le Aste attive in cui un Utente ha lanciato almeno un'Offerta

*Considerando che ogni Utente partecipa in media a due Aste*




















Concetto nello schema	Costrutto	Accessi	Tipo
UTENTE BASE	E	1	L
ASTA	E	2	L
RILANCIO	R	2	L

## Ristrutturazione dello schema E-R

Lo schema concettuale presentato è funzionante e navigabile, ma dall'analisi dei costi diventa evidente la necessità di ottimizzare diversi aspetti del Sistema.

Si schematizza nel seguito la distribuzione complessiva del carico applicativo

(con \* si fa riferimento alle quantità nel complesso trascurabili).

N° OPERAZIONE	PESO	FREQUENZA	COSTO	VOLUME	CONCETTI COINVOLTI
1		*	*	*	*
2		*	*	*	*
3		*	*	*	*
4		*	*	*	*
5		*	*	*	*
6		*	*	*	*
7		*	*	*	*
8		*	*	*	*
9		*	*	*	*
10		150 al giorno	160 (L)	400	Asta
11		500 al giorno	9 (L)	1200	Asta, Oggetto, Implicazione
12		200 al giorno	240(L)	1200	Asta, Oggetto, Implicazione
13		300 al giorno	6(S)	/	
14		20 al giorno	61(L)	1220	Asta, Implicazione, Esposizione, Oggetto
15		50 al giorno	801(L)	320.400	Asta, Offerta, Associazione
16		20 al giorno	161	320.400	Asta, Offerta, Associazione
17		*	*	*	*
18		*	*	*	*
19		*	*	*	*

Dalla tabella salta all'occhio una peculiarità tipica di un'applicazione per la gestione di dati, ovvero che l'80 per cento del carico applicativo è generato dal 20 per cento delle operazioni (*regola, seppur approssimativa, dell'ottanta-venti*).

In particolare, si possono individuare due fattori centrali responsabili dell'inefficienza del sistema, così com'è stato costruito:

- 1) Una forte ridondanza per quanto riguarda le entità di Oggetto e Asta, che tra l'altro sono legati da una relazione 1 a 1, la coesistenza dei quali incrementa notevolmente il numero di accessi in lettura;
- 2) La gravosità, in termini di volume di dati (e quindi anche nei tempi di accesso in lettura!) nel mantenere lo storico di tutte le offerte che sono state fatte per ogni asta esistente, attiva o conclusa.

Sulla base di queste considerazioni, si può ristrutturare il diagramma E-R in modo più efficiente possibile.

### **Analisi delle Ridondanze**

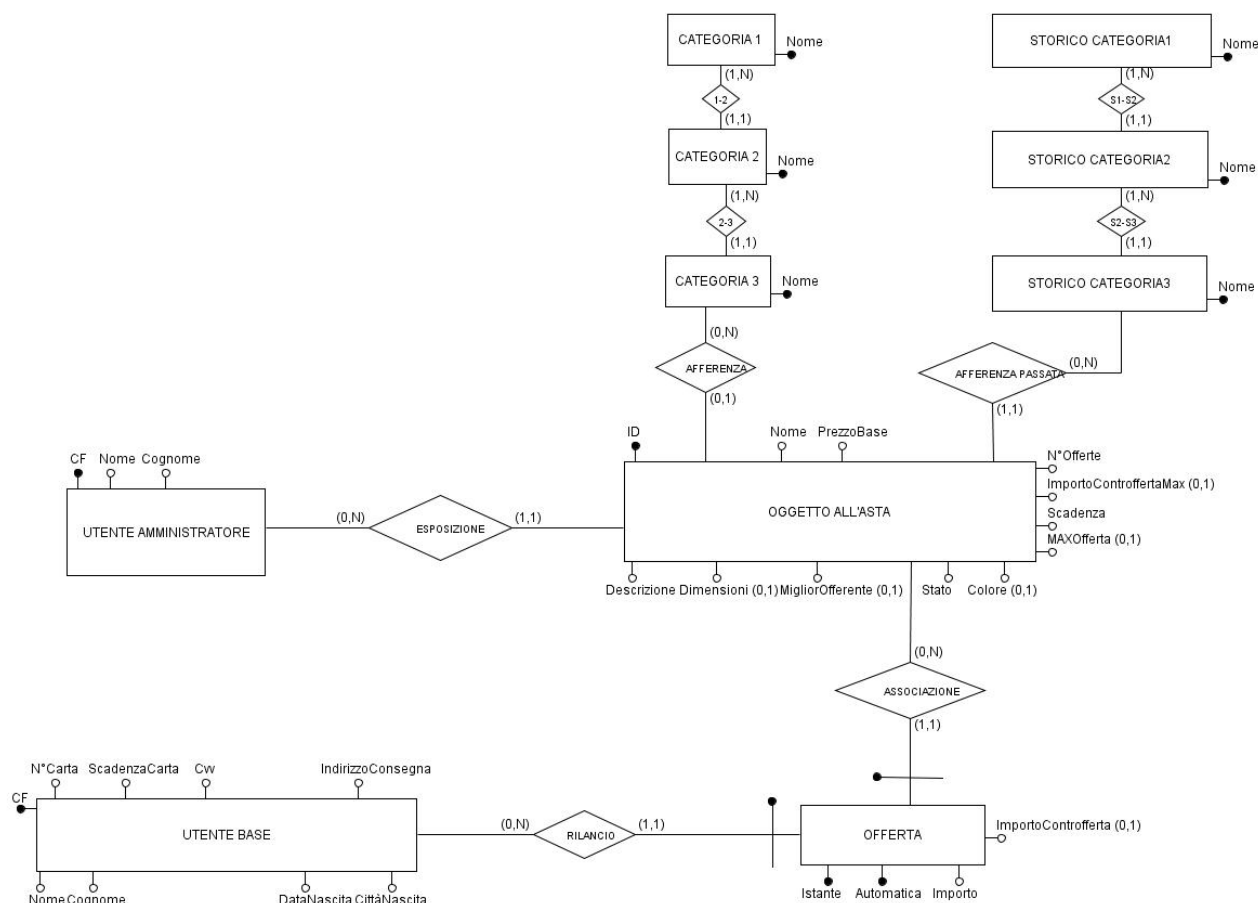
- 1) Come accennato, le Entità Oggetto e Asta sono due facce della stessa medaglia, e generano una forte ridondanza “concettuale”.

Perché mantenere i due concetti separati, quando di fatto l'inserimento di un Oggetto nel sistema implica l'esposizione di questo all'Asta, e vincere l'asta significa aggiudicarsi l'Oggetto?

L'applicazione che si sta progettando lavora molto in lettura, e mantenere relazioni 1 a 1 può in generale essere una scelta poco funzionale, in quanto diventerebbe necessario duplicare gli accessi necessari ad ottenere informazioni che potrebbero essere unificate senza troppi problemi.

Per questo e per altri motivi andiamo a considerare una tipologia di soluzione efficace, nota come “*Accorpamento di Concetti*”.

Proprio per la loro natura biunivoca, non ha senso mantenere separati i concetti di Oggetto ed Asta, pertanto è possibile evitare una ingente quantità di accessi, sia in scrittura che in lettura, accorpando il tutto nell'entità “Oggetto all'Asta”, che presenta gli attributi di Oggetto e di Asta insieme. 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.



- 2) La seconda ridondanza su cui è opportuno soffermarsi è quella sul mantenimento dei dati relativi alle categorie passate, che corrisponde ad un insieme contenente anche le categorie correnti.

Per ragioni di sicurezza, però, conviene mantenere la storicizzazione:

nello scenario in cui un utente amministratore eliminasse una categoria a cui afferiscono uno o più oggetti all'asta, questi manterranno la connessione a tale categoria attraverso lo storico delle categorie.

Inoltre, le operazioni sulla gestione delle categorie saranno sporadiche, e una volta stabilizzato un titolare accettabile le sue dimensioni rimarranno pressoché costanti.

Ciò non dovrebbe gravare sulla base di dati.

### Eliminazione delle Generalizzazioni

Non sono presenti generalizzazioni all'interno dello schema.

## **Trasformazione di attributi e identificatori**

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

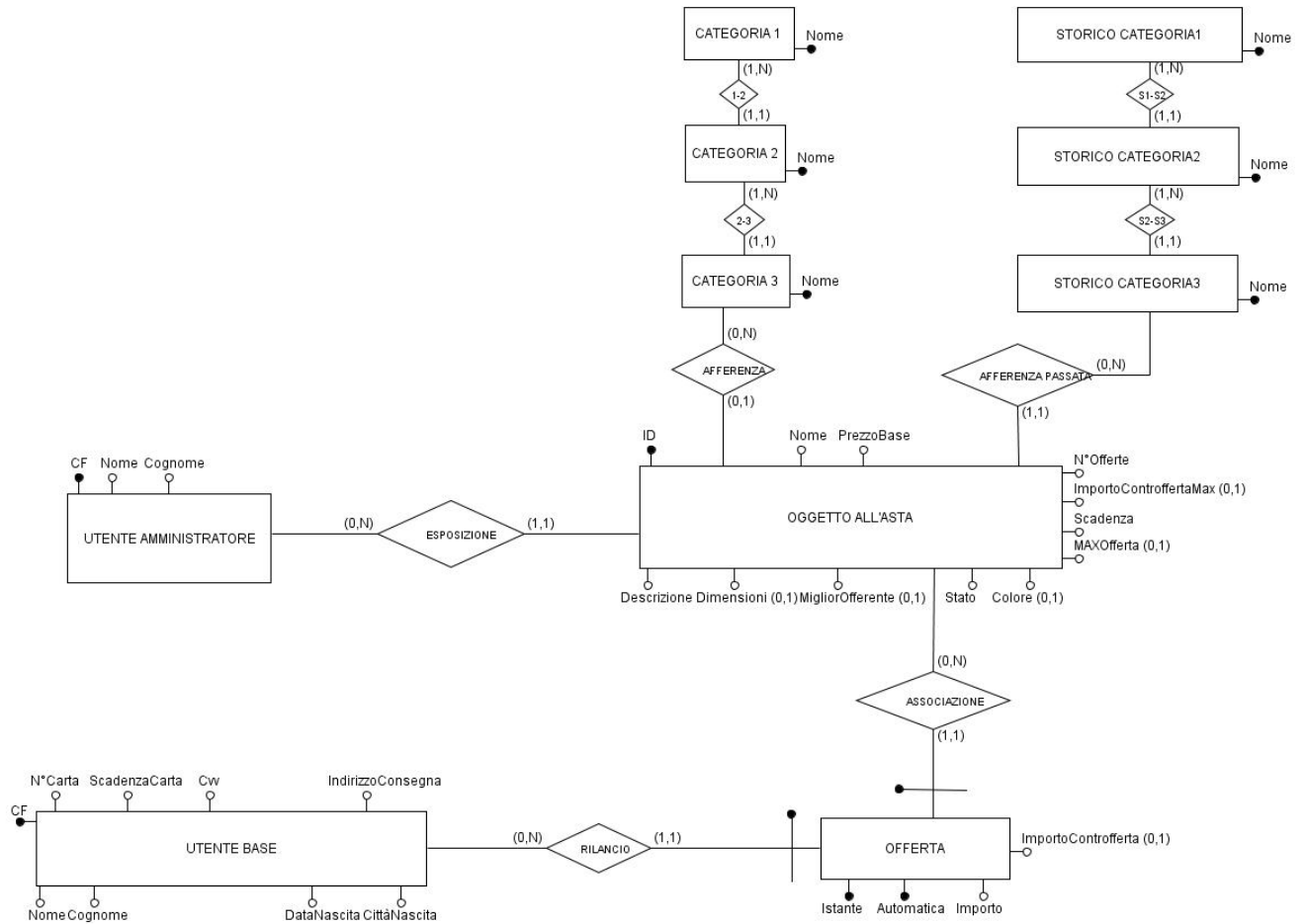
Gli identificatori esterni presenti per l'entità Offerta sono necessari, in quanto ogni offerta può essere identificata solo se si conoscono l'offerente, l'oggetto di interesse, la data e l'ora in cui essa viene effettuata.

Si potrebbe pensare di aggiungere un codice identificativo univoco per ogni offerta, ma sarebbe un inutile spreco di risorse hardware (in termini di volume dei dati).

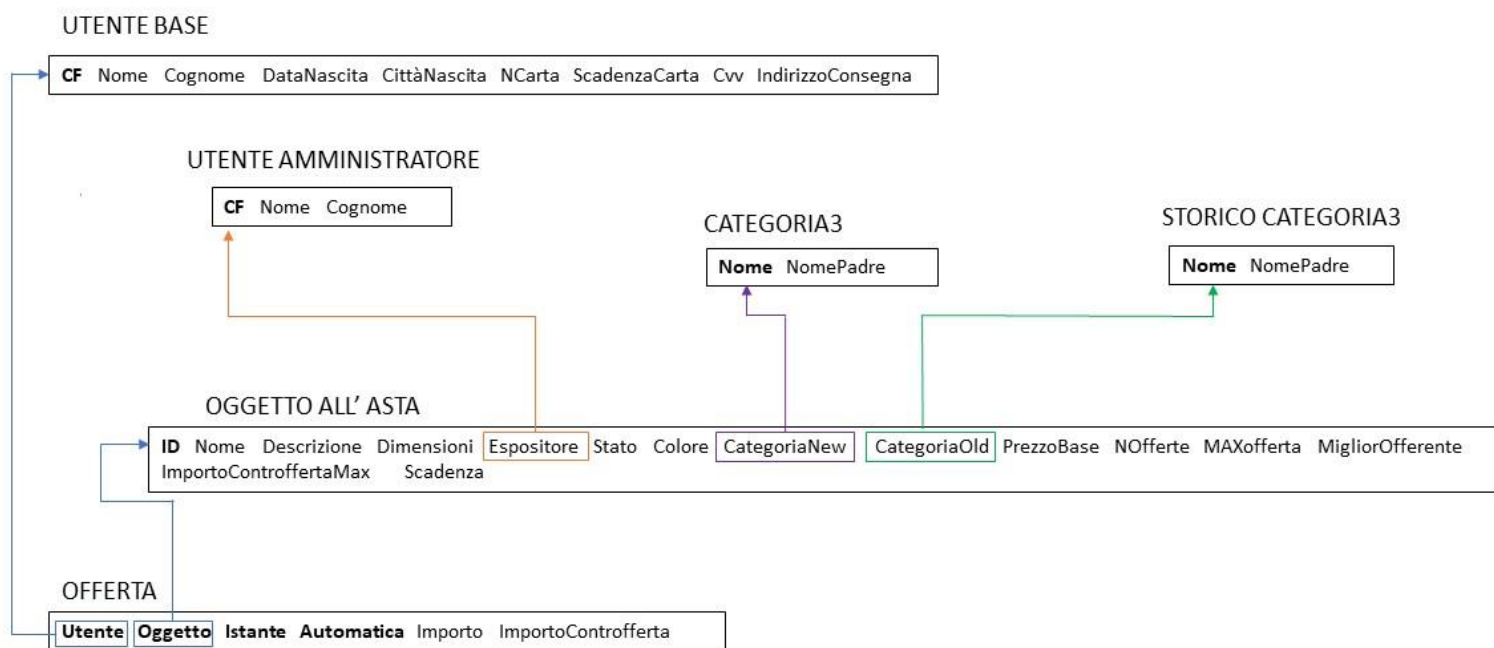
Inoltre, è da notare che, considerando un modello relazionale dei dati, il doppio identificatore esterno implica che la relazione Offerta rappresenterà obbligatoriamente anche le associazioni RILANCIO e ASSOCIAZIONE, che non avranno dunque una propria rappresentazione tabellare.

## Traduzione di entità e associazioni

- 1) Si riporta dapprima lo schema concettuale Entity-Relationship, per poi passare alla rappresentazione grafica della traduzione di tale schema.



## 2) Traduzione dell'E-R con i vincoli di integrità referenziale tra le varie relazioni.



### 3) Schema Relazionale.

UTENTEBASE (CF, Nome, Cognome, DataNascita, CittàNascita, NCarta, ScadenzaCarta, Cvv, IndirizzoConsegna)

UTENTEAMMINISTRATORE (CF, Nome, Cognome)

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

OGGETTOALLASTA (ID, Espositore, CategoriaNew\*, CategoriaOld, Nome, Descrizione, Dimensioni\*, Stato, Colore\*, PrezzoBase, NOfferte, MAXofferta\*, MigliorOfferente\*, ImportoControffertaMax\*, Scadenza)

CATEGORIA3(Nome, NomePadre\*)

CATEGORIA2(Nome, NomePadre\*)

CATEGORIA1(Nome)

STORICOCATEGORIA3(Nome, NomePadre\*)

STORICOCATEGORIA2(Nome, NomePadre\*)

STORICOCATEGORIA1(Nome)

## Normalizzazione del modello relazionale

Consideriamo separatamente le varie relazioni.

Per ognuna delle relazioni, bisogna determinare le dipendenze funzionali tra gli attributi costituenti, e verificare se sono soddisfatte le forme normali.

La soddisfazione delle forme normali determina la qualità dello schema relazionale, in termini di ridondanze e anomalie.

Di fatto, lo schema relazionale non contiene, nelle sue relazioni, alcun tipo di dipendenza funzionale.

Ciò implica inequivocabilmente che la soluzione è già ottimale e in particolare non ridondante: ci troviamo nel caso in cui la forma normale di “Boyce e Codd” è soddisfatta, e di conseguenza lo sono anche 1NF, 2NF, 3NF.



## 5. Progettazione fisica

### Utenti e privilegi

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

#### 1) *Privilegi concessi agli Utenti Base.*

Un Utente Base può accedere alle seguenti operazioni, stando alle specifiche dei requisiti:

- Operazione 6: visualizza tutte le aste attive
- Operazione 7: visualizza le aste attive associate al nome di un oggetto
- Operazione 8: visualizza le aste attive associate ad una categoria
- Operazione 9: registra l'offerta di un utente per un'asta (rilancia un'offerta)
- Operazione 10: visualizza le aste indette da un utente amministratore
- Operazione 16: visualizza lo stato di un'asta
- Operazione 17: visualizza tutti gli oggetti aggiudicati
- Operazione 18: trova tutte le aste attive in cui l'utente ha lanciato almeno un'offerta

I privilegi concessi agli Utenti Base sono:

1. Un Utente può accedere allo stato di un'asta qualsiasi, senza poter vedere chi sia il miglior offerente di essa.

A tal proposito si può definire le viste “VISUALIZZAZIONE\_ASTE” e “ASTE\_ATTIVE”, definite con l'istruzione SQL:

```
CREATE VIEW VISUALIZZAZIONE_ASTE AS
SELECT ID, Espositore, Nome, Categoria, Descrizione,
Dimensioni, Stato, Colore, PrezzoBase, N°Offerte, MAXofferta,
Scadenza
FROM OGGETTOALLASTA;
```

```
CREATE VIEW ASTE_ATTIVE AS
SELECT ID, Espositore, Nome, Categoria, Descrizione,
Dimensioni, Stato, Colore, PrezzoBase, N°Offerte, MAXofferta,
Scadenza
FROM OGGETTOALLASTA
WHERE Scadenza > now ()
```

e garantire il **privilegio di SELECT** sulle viste **“VISUALIZZAZIONE\_ASTE”** e **“ASTE\_ATTIVE”** all’Utente Base.

In questo modo l’Utente potrà leggere (non scrivere né cancellare!) tutte le informazioni che egli necessita dalle aste, senza accedere al contenuto degli attributi **“ImportoControffertaMax”** (che deve restare segreto, per far sì che la funzione di controfferta funzioni regolarmente) e **“MigliorOfferente”**.

2. Un Utente può rilanciare un’offerta su un’asta, indicando un importo e un importo di controfferta automatica.

L’Utente non può inserire un’offerta contrassegnata come **“automatica”**, in quanto queste ultime saranno necessariamente gestite dal sistema.

È garantito il **privilegio di INSERT** sugli attributi **“Importo”**, **“ImportoControfferta”** sulla tabella **“OFFERTA”** all’Utente Base.

## 2) *Privilegi concessi all’Utente Amministratore.*

Stando alle specifiche dei requisiti, un Utente Amministratore può accedere alle seguenti operazioni:

- Operazione 1: **“inserisci un nuovo oggetto nel sistema”**
  - Operazione 2: **“inizializza l’asta per un nuovo oggetto”**
  - Operazione 3: **“inserisci nuova categoria”**
  - Operazione 4: **“elimina categoria”**
  - Operazione 5: **“aggiorna categoria”**
  - Operazione 11: **“trova tutte le offerte fatte in un’asta”**
  - Operazione 12: **“trova tutte le offerte generate in un’asta dal sistema di controfferta automatica”**
1. Un Utente Amministratore può inserire un oggetto nel sistema, quindi inizializzarne un’asta (le operazioni 1 e 2 sono in effetti un’unica operazione, avendo accorpato i concetti di oggetto e asta):

Viene garantito il **privilegio di INSERT** sull'entità **“OGGETTOALLASTA”** all'Utente **Amministratore**.

2. Un Utente Amministratore ha in carico la *Gestione delle Categorie* a cui gli oggetti afferiscono:

vengono garantiti i seguenti privilegi all'Utente Amministratore:

a) **SELECT, DELETE, UPDATE** sulle tabelle **“CATEGORIA3”, “CATEGORIA2”, “CATEGORIA1”**

b) **INSERT** sulla tabella **“CATEGORIA3”**

c) **SELECT** sulle tabelle **“STORICOCATEGORIA3”, “STORICOCATEGORIA2”, “STORICOCATEGORIA1”**

Per giustificare queste scelte, è necessario comprendere l'idea di schematizzazione del titolare gerarchico, quindi della Gestione delle Categorie.

## Strutture di memorizzazione

Tabella UTENTEbase		
Attributo	Tipo di dato	Attributi <sup>2</sup>
CF	CHAR (16)	PK
Nome	VARCHAR (20)	NN
Cognome	VARCHAR (20)	NN
DataNascita	DATE	NN
CittàNascita	VARCHAR (16)	NN
NCarta	CHAR (16)	UQ, NN
ScadenzaCarta	CHAR (4)	NN
Cvv	CHAR (3)	NN
IndirizzoConsegna	VARCHAR (30)	NN

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

<b>Tabella UTENTEAMMINISTRATORE</b>		
<b>Attributo</b>	<b>Tipo di dato</b>	<b>Attributi<sup>3</sup></b>
<b>CF</b>	CHAR (16)	PK
<b>Nome</b>	VARCHAR (20)	NN
<b>Cognome</b>	VARCHAR (20)	NN

<b>Tabella OFFERTA</b>		
<b>Attributo</b>	<b>Tipo di dato</b>	<b>Attributi<sup>4</sup></b>
<b>Utente</b>	CHAR (16)	PK
<b>Oggetto</b>	INTEGER	PK
<b>Istante</b>	TIMESTAMP	PK
<b>Importo</b>	NUMERIC (5,2)	NN
<b>Automatica</b>	BOOLEAN	PK
<b>ImportoControfferta</b>	NUMERIC (5,2)	

---

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

Tabella OGGETTOALLASTA		
Attributo	Tipo di dato	Attributi <sup>5</sup>
ID	INTEGER	PK, AI
Nome	VARCHAR (20)	NN
Descrizione	LONGTEXT	
CategoriaNew	VARCHAR (20)	
CategoriaOld	VARCHAR (20)	NN
Stato	VARCHAR (20)	NN
Colore	VARCHAR (16)	
Espositore	CHAR (16)	NN
Dimensioni	VARCHAR (10)	
PrezzoBase	NUMERIC (7,2)	NN
NOfferte	INTEGER	NN
MAXofferta	NUMERIC (5,2)	
MigliorOfferente	CHAR (16)	
ImportoControffertaMax	NUMERIC (5,2)	
Scadenza	TIMESTAMP	NN

Tabella CATEGORIA3		
Attributo	Tipo di dato	Attributi <sup>6</sup>
Nome	VARCHAR (20)	PK
NomeCategoriaPadre	VARCHAR (20)	

Tabella CATEGORIA2		
Attributo	Tipo di dato	Attributi <sup>7</sup>
Nome	VARCHAR (20)	PK
NomeCategoriaPadre	VARCHAR (20)	

Tabella CATEGORIA1		
Attributo	Tipo di dato	Attributi <sup>8</sup>
Nome	VARCHAR (20)	PK

Tabella STORICOCATEGORIA3		
Attributo	Tipo di dato	Attributi <sup>9</sup>
Nome	VARCHAR (20)	PK
NomeCategoriaPadre	VARCHAR (20)	

Tabella STORICOCATEGORIA2		
Attributo	Tipo di dato	Attributi <sup>10</sup>
Nome	VARCHAR (20)	PK
NomeCategoriaPadre	VARCHAR (20)	

---

Tabella STORICOCATEGORIA1		
Attributo	Tipo di dato	Attributi <sup>11</sup>
Nome	VARCHAR (20)	PK

## Indici

### 1) INDICI SULLA TABELLA “OFFERTA”

Particolare attenzione va posta sulla tabella “OFFERTA”, in quanto essa mantiene le informazioni su tutte le offerte, da parte di tutti gli utenti, su tutte le aste che sono attive e su quelle aggiudicate.

Le operazioni di visualizzazione (che implicano selezioni e join) sono quelle che necessitano effettivamente di un’indicizzazione. In particolare, sulla tabella offerta, insistono le operazioni:

- Operazione 11: “trova tutte le offerte fatte in un’asta”
- Operazione 12: “trova tutte le offerte generate in un’asta dal sistema di controfferta automatica”
- Operazione 18: “trova tutte le aste attive in cui l’utente ha lanciato almeno un’offerta”

Non è previsto un ordinamento del file su chiave primaria o attributi secondari, in quanto per ogni tupla inserita nella tabella sarebbe opportuno riorganizzare il file secondo l’ordinamento stabilito, e la tabella “OFFERTA” è perlopiù soggetta ad inserimenti.

Per quanto detto, si può procedere con la definizione di indici secondari (indicizzazione densa), che massimizzino le prestazioni delle operazioni di visualizzazione.

Tabella OFFERTA	
Indice OFFERTE_ASTA_IDX	Tipo <sup>12</sup> : IDX
Colonna 2	Oggetto
Colonna 3	Istante

---

<sup>12</sup> IDX = index, UQ = unique, FT = full text, PR = primary.

Le chiavi di questo indice sono ordinate secondo Oggetto, Istante, e ciò permette di ottenere rapidamente le informazioni relative a tutte le offerte fatte su un determinato oggetto all'asta (che ad esempio gli Utenti Amministratori possono visualizzare in un report). L'ordinamento per Oggetto si presta tra l'altro all'equi-join con la tabella "OGGETTOALLASTA", su cui verranno definiti indici adeguatamente.

## 2) INDICI SULLA TABELLA "OGGETTOALLASTA"

Le operazioni di visualizzazione che coinvolgono la tabella "OGGETTOALLASTA"

- Operazione 6: visualizza tutte le aste attive
- Operazione 7: visualizza le aste attive associate al nome di un oggetto
- Operazione 8: visualizza le aste attive associate ad una categoria
- Operazione 10: visualizza le aste indette da un utente amministratore
- Operazione 16: visualizza lo stato di un'asta
- Operazione 17: visualizza tutti gli oggetti aggiudicati
- Operazione 18: trova tutte le aste attive in cui l'utente ha lanciato almeno un'offerta

La più visibile suddivisione che può essere fatta sulle tuple di "OGGETTOALLASTA" è quella tra oggetti aggiudicati e oggetti in palio (quindi aste concluse e aste attive):

le operazioni 6,7,8,17,18 si basano tutte, in un modo o nell'altro, su questa suddivisione.

Introdurre un indice secondario sul campo "Scadenza" velocizza quindi le operazioni più frequenti, accedendo le risorse in tempo logaritmico.

Tabella OGGETTOALLASTA	
Indice OGGETTI_SCAD_IDX	Tipo <sup>13</sup> : IDX
Colonna 2	Scadenza

<sup>13</sup> IDX = index, UQ = unique, FT = full text, PR = primary.



Tabella OGGETTOALLASTA	
Indice OGGETTI_CAT_IDX	Tipo <sup>14</sup> : IDX
Colonna 5	CategoriaOld
Colonna 14	Scadenza

Tabella OGGETTOALLASTA	
Indice OGGETTI_NOME_IDX	Tipo <sup>15</sup> : IDX
Colonna 2	Nome
Colonna 14	Scadenza

## Trigger

1. Trigger per l'inserimento di nuove categorie nello storico, nei rispettivi tre livelli del titolario gerarchico, *dopo l'inserimento* della nuova categoria nel titolario operativo.

```
CREATE TRIGGER inserimento_storico1_trigger
AFTER INSERT ON categorial
FOR EACH ROW
BEGIN
    declare var_compara_nome varchar(20);
    select `Nome` from `storicocategorial`
    where `Nome` = New.`Nome` into var_compara_nome;
    if var_compara_nome is null then
        insert into `storicocategorial` (`Nome`)
        values (New.`Nome`);
    end if;
END;
```

---

<sup>14</sup> IDX = index, UQ = unique, FT = full text, PR = primary.

<sup>15</sup> IDX = index, UQ = unique, FT = full text, PR = primary.

```

CREATE TRIGGER inserimento_storico2_trigger
AFTER INSERT ON categoria2
FOR EACH ROW
BEGIN
    declare var_compara_nome varchar (20);
    select `Nome`
    from `storicocategoria2`
    where `Nome` = New.`Nome`
    into var_compara_nome;
    if var_compara_nome is null then
        insert into `storicocategoria2` (`Nome`,
        `NomeCategoriaPadre`)
        values (New.`Nome`, New.`NomeCategoriaPadre`);
    end if;
END;

```

```

CREATE TRIGGER inserimento_storico3_trigger
AFTER INSERT ON categoria3
FOR EACH ROW
BEGIN
    declare var_compara_nome varchar (20);
    select `Nome`
    from `storicocategoria3`
    where `Nome` = New.`Nome`
    into var_compara_nome;
    if var_compara_nome is null then
        insert into `storicocategoria3` (`Nome`,
        `NomeCategoriaPadre`)
        values (New.`Nome`, New.`NomeCategoriaPadre`);
    end if;
END;

```

2. Trigger per l'inserimento di nuove categorie nello storico, nei rispettivi tre livelli del titolario gerarchico, *dopo l'aggiornamento* di una categoria nel titolario operativo.

```

CREATE TRIGGER aggiornamento_storico1_trigger
AFTER UPDATE ON categoria1
FOR EACH ROW
BEGIN
    update `storicocategoria1` set `Nome` = New.`Nome`
    where `Nome` = Old.`Nome`;
END;

```

```

CREATE TRIGGER aggiornamento_storico2_trigger
AFTER UPDATE ON categoria2

```

```
FOR EACH ROW
BEGIN
    update `storicocategoria2`
    set `Nome` = New.`Nome`,
        `NomeCategoriaPadre` = New.`NomeCategoriaPadre`
    where `Nome` = Old.`Nome`
    and `NomeCategoriaPadre` = Old.`NomeCategoriaPadre`;
END;
```

```
CREATE TRIGGER aggiornamento_storico3_trigger
AFTER UPDATE ON categoria3
FOR EACH ROW
BEGIN
    update `storicocategoria3`
    set `Nome` = New.`Nome`,
        `NomeCategoriaPadre` = New.`NomeCategoriaPadre`
    where `Nome` = Old.`Nome`
    and `NomeCategoriaPadre` = Old.`NomeCategoriaPadre`;
END;
```

Sostanzialmente, i trigger delle famiglie 1 e 2 implementano la funzionalità della gestione dello storico delle categorie:

Gli Utenti Amministratori possono accedere direttamente al titolare gerarchico corrente, rappresentato dalle relazioni Categoria3, Categoria2, Categoria1, attraverso procedure di inserimento, modifica e cancellazione di categorie.

Il titolare corrente è quello che può essere quindi utilizzato, durante l'inizializzazione di un'asta, per stabilire la categoria di appartenenza dell'oggetto in questione.

Per ogni inserimento di una nuova categoria e per ogni modifica di una già esistente, i trigger sopra riportati automatizzano l'aggiornamento dello storico, che quindi varia in generale allo stesso modo del titolare corrente, fatta eccezione per le eliminazioni di categorie, che non sono ovviamente previste in quest'ultimo.

## Viste

Le viste implementate sono perlopiù utilizzate per operazioni degli utenti base.

Non avendo accesso diretto autorizzato sulla relazione Oggettoallasta, vengono utilizzate le tabelle virtuali di “visualizzazione\_aste” (per l’accesso a tutte le aste, attive e concluse) e “aste\_attive” (per le operazioni sulle aste attive).

```
CREATE OR REPLACE VIEW `aste_attive` AS
SELECT `ID`, `Nome`, `Espositore`, `CategoriaOld`, `Descrizione`,
`Dimensioni`, `Stato`, `Colore`, `PrezzoBase`, `N°Offerte`,
`MAXofferta`, `Scadenza`
FROM `oggettoallasta`
WHERE `Scadenza` > current_timestamp ()
```

```
CREATE OR REPLACE VIEW `visualizzazione_aste` AS
SELECT `ID`, `Nome`, `Espositore`, `CategoriaOld`, `Descrizione`,
`Dimensioni`, `Stato`, `Colore`, `PrezzoBase`, `N°Offerte`,
`MAXofferta`, `Scadenza`
FROM `oggettoallasta`
```

## 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 l’”intorno” del ciclo di vita delle aste.

### 1. 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`

### 2. Logica delle Aste

- `inizializzazione\_asta`
- `registra\_offerta`
- `inserimento\_categoria1`
- `inserimento\_categoria2`
- `inserimento\_categoria3`
- `aggiornamento\_categoria`
- `cancellazione\_categoria`

### 3. Accesso

- `registrazione\_utente`
- `validazione\_accesso`

## Appendice: Implementazione

### Codice SQL per istanziare il database

```
-- MySQL Script generated by MySQL Workbench
-- gio 06 giu 2019 16:39:13 CEST
-- Model: New Model      Version: 1.0
-- MySQL Workbench Forward Engineering

SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0;
SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0;
SET @OLD_SQL_MODE=@@SQL_MODE, SQL_MODE='TRADITIONAL,ALLOW_INVALID_DATES';

-- -----
-- Schema sistemaaste
-- -----

-- -----
-- Schema sistemaaste
-- -----

CREATE SCHEMA IF NOT EXISTS `sistemaaste` DEFAULT CHARACTER SET latin1 ;
USE `sistemaaste` ;

-- -----
-- Table `sistemaaste`.`categorial`
-- -----
CREATE TABLE IF NOT EXISTS `sistemaaste`.`categorial` (
  `Nome` VARCHAR(20) NOT NULL,
  PRIMARY KEY (`Nome`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = latin1;

-- -----
-- Table `sistemaaste`.`categoria2`
-- -----
CREATE TABLE IF NOT EXISTS `sistemaaste`.`categoria2` (
  `Nome` VARCHAR(20) NOT NULL,
  `NomeCategoriaPadre` VARCHAR(20) NULL DEFAULT NULL,
  PRIMARY KEY (`Nome`),
  CONSTRAINT `fk_categoria2_categorial`
    FOREIGN KEY (`NomeCategoriaPadre`)
    REFERENCES `sistemaaste`.`categorial` (`Nome`)
    ON DELETE SET NULL
    ON UPDATE CASCADE)
ENGINE = InnoDB
DEFAULT CHARACTER SET = latin1;

CREATE INDEX `fk_categoria2_categorial_idx` ON `sistemaaste`.`categoria2`
(`NomeCategoriaPadre` ASC);

-- -----
-- Table `sistemaaste`.`categoria3`
-- -----
CREATE TABLE IF NOT EXISTS `sistemaaste`.`categoria3` (
  `Nome` VARCHAR(20) NOT NULL,
  `NomeCategoriaPadre` VARCHAR(20) NULL,
  PRIMARY KEY (`Nome`),
  CONSTRAINT `fk_categoria3_categoria2`
    FOREIGN KEY (`NomeCategoriaPadre`)
    REFERENCES `sistemaaste`.`categoria2` (`Nome`)
    ON DELETE SET NULL
    ON UPDATE CASCADE)
ENGINE = InnoDB
```

```

DEFAULT CHARACTER SET = latin1;

CREATE INDEX `fk_categoria3_categoria2l_idx` ON `sistemaaste`.`categoria3`
(`NomeCategoriaPadre` ASC);

-- -----
-- Table `sistemaaste`.`utentebase`
-- -----
CREATE TABLE IF NOT EXISTS `sistemaaste`.`utentebase` (
  `CF` CHAR(16) NOT NULL,
  `Nome` VARCHAR(20) NOT NULL,
  `Cognome` VARCHAR(20) NOT NULL,
  `DataNascita` DATE NOT NULL,
  `CittaNascita` VARCHAR(20) NOT NULL,
  `NCarta` CHAR(16) NOT NULL,
  `ScadenzaCarta` CHAR(4) NOT NULL,
  `CVV` CHAR(3) NOT NULL,
  `IndirizzoConsegna` VARCHAR(30) NOT NULL,
  PRIMARY KEY (`CF`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = latin1;

CREATE UNIQUE INDEX `NCarta` ON `sistemaaste`.`utentebase` (`NCarta` ASC);

-- -----
-- Table `sistemaaste`.`storicocategoria1`
-- -----
CREATE TABLE IF NOT EXISTS `sistemaaste`.`storicocategoria1` (
  `Nome` VARCHAR(20) NOT NULL,
  PRIMARY KEY (`Nome`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = latin1;

-- -----
-- Table `sistemaaste`.`storicocategoria2`
-- -----
CREATE TABLE IF NOT EXISTS `sistemaaste`.`storicocategoria2` (
  `Nome` VARCHAR(20) NOT NULL,
  `NomeCategoriaPadre` VARCHAR(20) NULL DEFAULT NULL,
  PRIMARY KEY (`Nome`),
  CONSTRAINT `fk_storicocategoria2_storicocategoria1l`
    FOREIGN KEY (`NomeCategoriaPadre`)
      REFERENCES `sistemaaste`.`storicocategoria1` (`Nome`)
      ON DELETE NO ACTION
      ON UPDATE NO ACTION)
ENGINE = InnoDB
DEFAULT CHARACTER SET = latin1;

CREATE INDEX `fk_storicocategoria2_storicocategoria1l_idx` ON
`sistemaaste`.`storicocategoria2` (`NomeCategoriaPadre` ASC);

-- -----
-- Table `sistemaaste`.`storicocategoria3`
-- -----
CREATE TABLE IF NOT EXISTS `sistemaaste`.`storicocategoria3` (
  `Nome` VARCHAR(20) NOT NULL,
  `NomeCategoriaPadre` VARCHAR(20) NULL DEFAULT NULL,
  PRIMARY KEY (`Nome`),
  CONSTRAINT `fk_storicocategoria3_storicocategoria2l`
    FOREIGN KEY (`NomeCategoriaPadre`)
      REFERENCES `sistemaaste`.`storicocategoria2` (`Nome`)
      ON DELETE NO ACTION
      ON UPDATE NO ACTION)
ENGINE = InnoDB

```

```

DEFAULT CHARACTER SET = latin1;

CREATE INDEX `fk_storicocategoria3_storicocategoria21_idx` ON
`sistemaaste`.`storicocategoria3` (`NomeCategoriaPadre` ASC);

-- -----
-- Table `sistemaaste`.`utenteamministratore`
-- -----
CREATE TABLE IF NOT EXISTS `sistemaaste`.`utenteamministratore` (
  `CF` CHAR(16) NOT NULL,
  `Nome` VARCHAR(20) NOT NULL,
  `Cognome` VARCHAR(20) NOT NULL,
  PRIMARY KEY (`CF`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = latin1;

-- -----
-- Table `sistemaaste`.`oggettoallasta`
-- -----
CREATE TABLE IF NOT EXISTS `sistemaaste`.`oggettoallasta` (
  `ID` INT(11) NOT NULL AUTO_INCREMENT,
  `Nome` VARCHAR(20) NOT NULL,
  `CategoriaNew` VARCHAR(20) NULL DEFAULT NULL,
  `CategoriaOld` VARCHAR(20) NOT NULL,
  `Descrizione` LONGTEXT NULL DEFAULT NULL,
  `Stato` VARCHAR(20) NOT NULL,
  `Colore` VARCHAR(20) NULL DEFAULT NULL,
  `Espositore` CHAR(16) NOT NULL,
  `Dimensioni` VARCHAR(10) NULL DEFAULT NULL,
  `PrezzoBase` DECIMAL(7,2) NOT NULL,
  `NOfferte` INT(11) NOT NULL DEFAULT 0,
  `MAXOfferta` DECIMAL(7,2) NULL DEFAULT NULL,
  `MigliorOfferente` CHAR(16) NULL DEFAULT NULL,
  `ImportoControffertaMax` DECIMAL(7,2) NULL DEFAULT NULL,
  `Scadenza` TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP() ON UPDATE CURRENT_TIMESTAMP(),
  PRIMARY KEY (`ID`),
  CONSTRAINT `fk_oggettoallasta_categoria31`
    FOREIGN KEY (`CategoriaNew`)
    REFERENCES `sistemaaste`.`categoria3` (`Nome`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_oggettoallasta_storicocategoria31`
    FOREIGN KEY (`CategoriaOld`)
    REFERENCES `sistemaaste`.`storicocategoria3` (`Nome`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_oggettoallasta_utenteamministratore1`
    FOREIGN KEY (`Espositore`)
    REFERENCES `sistemaaste`.`utenteamministratore` (`CF`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB
DEFAULT CHARACTER SET = latin1;

CREATE INDEX `fk_oggettoallasta_categoria31_idx` ON `sistemaaste`.`oggettoallasta`
(`CategoriaNew` ASC);

CREATE INDEX `fk_oggettoallasta_storicocategoria31_idx` ON `sistemaaste`.`oggettoallasta`
(`CategoriaOld` ASC);

CREATE INDEX `fk_oggettoallasta_utenteamministratore1_idx` ON
`sistemaaste`.`oggettoallasta` (`Espositore` ASC);

-- -----
-- Table `sistemaaste`.`offerta`

```



```

-----
CREATE TABLE IF NOT EXISTS `sistemaaste`.`offerta` (
  `Utente` CHAR(16) NOT NULL,
  `Oggetto` INT(11) NOT NULL,
  `Istante` TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP() ON UPDATE CURRENT_TIMESTAMP(),
  `Importo` DECIMAL(7,2) NOT NULL,
  `Automatica` TINYINT(1) NOT NULL DEFAULT 0,
  `ImportoControfferta` DECIMAL(7,2) NULL DEFAULT NULL,
  PRIMARY KEY (`Utente`, `Oggetto`, `Istante`),
  CONSTRAINT `fk_offerta_utentebasel`
    FOREIGN KEY (`Utente`)
      REFERENCES `sistemaaste`.`utentebase` (`CF`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_offerta_oggettoallasta1`
    FOREIGN KEY (`Oggetto`)
      REFERENCES `sistemaaste`.`oggettoallasta` (`ID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB
DEFAULT CHARACTER SET = latin1;

CREATE INDEX `fk_offerta_utentebasel_idx` ON `sistemaaste`.`offerta` (`Utente` ASC);

CREATE INDEX `fk_offerta_oggettoallasta1_idx` ON `sistemaaste`.`offerta` (`Oggetto` ASC);

USE `sistemaaste` ;

-----
-- Placeholder table for view `sistemaaste`.`aste_active`
-----
CREATE TABLE IF NOT EXISTS `sistemaaste`.`aste_active` (`ID` INT, `Nome` INT, `Espositore`
INT, `CategoriaOld` INT, `Descrizione` INT, `Dimensioni` INT, `Stato` INT, `Colore` INT,
`PrezzoBase` INT, `NÂ°Offerte` INT, `MAXofferta` INT, `Scadenza` INT);

-----
-- Placeholder table for view `sistemaaste`.`visualizzazione_aste`
-----
CREATE TABLE IF NOT EXISTS `sistemaaste`.`visualizzazione_aste` (`ID` INT, `Nome` INT,
`Espositore` INT, `CategoriaOld` INT, `Descrizione` INT, `Dimensioni` INT, `Stato` INT,
`Colore` INT, `PrezzoBase` INT, `NÂ°Offerte` INT, `MAXofferta` INT, `Scadenza` INT);

-----
-- Placeholder table for view `sistemaaste`.`titolario_gerarchico_lower`
-----
CREATE TABLE IF NOT EXISTS `sistemaaste`.`titolario_gerarchico_lower` (`Nome` INT, `Padre`
INT);

-----
-- Placeholder table for view `sistemaaste`.`titolario_gerarchico_upper`
-----
CREATE TABLE IF NOT EXISTS `sistemaaste`.`titolario_gerarchico_upper` (`Nome` INT, `Padre`
INT);

```

## Codice delle stored procedures

```

-----
-- procedure inserimento_categoria3
-----

DELIMITER $$
USE `sistemaaste`$$

```

```

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;

    #controlla se la categoria Ã¨ giÃ  esistente
    select `Nome` from `categoria3` as C where C.`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 inserita al secondo livello
        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 titolare!";
        end if;
        insert into `categoria3`(`Nome`,`NomeCategoriaPadre`) values
(`nome_categoria`,`padre_categoria`);
    end if;

    commit;

END$$

DELIMITER ;

-- -----
-- procedure inserimento_categoria2
-- -----

DELIMITER $$
USE `sistemaaste`$$
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;

    # controlla se esiste gia la categoria
    select `Nome` from `categoria2` as C where C.`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 `categorial` where `Nome` = `padre_categoria` into
var_compara_nome_padre;
        # se non esiste questa categoria-padre viene inserita al secondo livello
        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 titolare!";
        end if;
        insert into `categoria2`(`Nome`,`NomeCategoriaPadre`) values
(`nome_categoria`,`padre_categoria`);
    end if;

    commit;

END$$

DELIMITER ;

-- -----
-- procedure inserimento_categorial
-- -----

DELIMITER $$
USE `sistemaaste`$$
CREATE PROCEDURE `inserimento_categorial` (in nome_categoria varchar(20))
BEGIN

    declare var_compara_nome varchar(20);

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

    start transaction;

    # controlla se esiste gia la categoria
    select `Nome` from `categorial` as C where C.`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 `categorial`(`Nome`) values (`nome_categoria`);

    commit;

END$$

DELIMITER ;

-- -----
-- procedure cancellazione_categoria
-- -----

DELIMITER $$
USE `sistemaaste`$$
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 `Nome` from `categorial1` as C where C.`Nome` = `nome_categoria` into
var_compara_nome;
        if var_compara_nome is null then

            select `Nome` from `categorial2` as C where C.`Nome` = `nome_categoria` into
var_compara_nome;
            if var_compara_nome is null then

                select `Nome` from `categorial3` as C where C.`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 `categorial3` where `Nome` =
`nome_categoria`;
                end if;

            end if;

        if var_compara_nome is not null then
            delete from `categorial2` where `Nome` = `nome_categoria`;
        end if;

    end if;

    if var_compara_nome is not null then
        delete from `categorial1` where `Nome` = `nome_categoria`;
    end if;

    commit;

```

```

END$$

DELIMITER ;

-- -----
-- procedure popolamento_categorie
-- -----

DELIMITER $$
USE `sistemaaste`$$
CREATE PROCEDURE `popolamento_categorie`()
BEGIN

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

    start transaction;

    call inserimento_categoria1('alimentari');
    call inserimento_categoria1('elettronica');
    call inserimento_categoria1('casa');

    call inserimento_categoria2('elettrodomestici', 'elettronica');
    call inserimento_categoria2('console', 'elettronica');
    call inserimento_categoria2('cibo fresco', 'alimentari');
    call inserimento_categoria2('cibo in scatola', 'alimentari');
    call inserimento_categoria2('giardinaggio', 'casa');
    call inserimento_categoria2('mobili', 'casa');

    call inserimento_categoria3('cucina', 'elettrodomestici');
    call inserimento_categoria3('pulizia', 'elettrodomestici');
    call inserimento_categoria3('portatili', 'console');
    call inserimento_categoria3('fisse', 'console');
    call inserimento_categoria3('palmari', 'console');
    call inserimento_categoria3('potatura', 'giardinaggio');
    call inserimento_categoria3('muratura', 'giardinaggio');
    call inserimento_categoria3('innaffiamento', 'giardinaggio');
    call inserimento_categoria3('pensili', 'mobili');
    call inserimento_categoria3('ripianti', 'mobili');
    call inserimento_categoria3('poltrone e divani', 'mobili');

    commit;

END$$

DELIMITER ;

-- -----
-- procedure aggiornamento_categoria
-- -----

DELIMITER $$
USE `sistemaaste`$$
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` 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` as C where C.`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` as C where C.`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` as C where C.`Nome` =
`old_nome_categoria` into var_compara_nome;
        if var_compara_nome is null then #chiave non trovata al
primo livello, quindi 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
primo livello.

            if `new_nome_padre` is null then

                if `new_nome_categoria` is null then
                    signal sqlstate '45004'
                    set message_text = "Non si puÃ²
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,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;
            end if;
        end if;
    end if;
end if;

```

```

                                if `new_nome_categoria` is null then #si
vuole aggiornare solo il padre
                                update `categoria3` set
                                `NomeCategoriaPadre` = `new_nome_padre` where `Nome` = `old_nome_categoria`;
                                end if;

                                if `new_nome_categoria` is not null then
#si vuole aggiornare sia il padre che il nome della categoria
                                update `categoria3` set `Nome` =
                                `new_nome_categoria`, `NomeCategoriaPadre` = `new_nome_padre`
                                where `Nome` =
                                `old_nome_categoria`;

                                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 then
                                if `new_nome_categoria` is null then
                                signal sqlstate '45004'
                                set message_text = "Non si puÃ² 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, ERRORE
                                select `Nome` from `categorial` 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 titolare!";
                                end if;

                                if `new_nome_categoria` is null then #si vuole
aggiornare solo il padre

                                update `categoria2` set `NomeCategoriaPadre` =
                                `new_nome_padre` where `Nome` = `old_nome_categoria`;
                                end if;

                                if `new_nome_categoria` is not null then #si vuole
aggiornare sia il padre che il nome della categoria
                                update `categoria2` set `Nome` =
                                `new_nome_categoria`, `NomeCategoriaPadre` = `new_nome_padre`
                                where `Nome` = `old_nome_categoria`;
                                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 then
                signal sqlstate '45004'
                set message_text = "Non si puÃ² aggiornare a valore nullo il
nome di una categoria.";
            end if;

            update `categorial` set `Nome` = `new_nome_categoria` where `Nome` =
`old_nome_categoria`;

        end if;

commit;

END$$

DELIMITER ;

-- -----
-- procedure registrazione_utente
-- -----

DELIMITER $$
USE `sistemaaste`$$
CREATE PROCEDURE `registrazione_utente` (in user_type int,
                                         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(4),
                                         in cvv char(3),
                                         in indirizzo_consegna varchar(30),
                                         in psw varchar(30))
BEGIN

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

    start transaction;

    if `user_type` = 0 then

        insert into `utenteamministratore` (`CF`, `Nome`, `Cognome`) values
(`cf`, `nome_utente`, `cognome_utente`);
        insert into `Login` (`CodiceFiscale`, `Password`, `Admin`) values
(`cf`, `psw`, `user_type`);

    end if;

    if `user_type` = 1 then

        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`(`CodiceFiscale`,`Password`,`Admin`) values
        (`cf`,`psw`,`user_type`);

    end if;

    if `user_type` <> 1 and `user_type` <> 0 then
        signal sqlstate '45010'
        set message_text = "tipologia utente non specificata.";
    end if;

    commit;

END$$

DELIMITER ;

-- -----
-- procedure inizializzazione_asta
-- -----

DELIMITER $$
USE `sistemaaste`$$
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(16), 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 (`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 `cat` is not null then

        insert into
`oggettoallasta`(`Nome`,`CategoriaNew`,`CategoriaOld`,`Descrizione`,`Stato`,`Colore`,`Espo
sitore`,
`Dimensioni`,`PrezzoBase`,`Scadenza`,`MAXOfferta`) values

        (`nome_ogg`,`cat`,`cat`,`descrizione_ogg`,`stato_ogg`,`color_ogg`,`espositore_ogg`,
`dim`,`prezzo_base`, date_add(now(), interval `scad_asta` day), `prezzo_base`);

    end if;

    commit;

```

```

END$$

DELIMITER ;

-- -----
-- procedure registra_offerta
-- -----

DELIMITER $$
USE `sistemaaste`$$
CREATE PROCEDURE `registra_offerta` (in cf_offerente char(16),
                                     in oggetto_asta int(11),
                                     in importo_rilancio
decimal(7,2),
                                     in importo_controfferta_max
decimal (7,2) )

BEGIN

    declare var_compara_offerta decimal(7,2);
    declare var_importo_controff decimal(7,2);

    declare ultimo_offerente char(16);
    declare scadenza_asta timestamp;

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

    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 Ã 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 Ã 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 Ã" stato indicato un importo di
rilancio sull'asta!";
    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 giÃ un'offerta massima
sull'oggetto all'asta

        if `importo_rilancio` <= var_compara_offerta then
            signal sqlstate '45012'
            set message_text = "Attenzione, l'importo dell'offerta non Ã"
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;

    if var_compara_offerta is null then #se questa Ã" 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 Ã"
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$$

```

```

DELIMITER ;

-- -----
-- procedure visualizza_aste_per_nome_oggetto
-- -----

DELIMITER $$
USE `sistemaaste`$$
CREATE PROCEDURE `visualizza_aste_per_nome_oggetto` (in nomeoggetto varchar(20))

BEGIN

    declare var_nomeoggetto varchar(20);

    set var_nomeoggetto = concat( "%", nomeoggetto, "%");

    select `ID`, `Nome`, `CategoriaOld`, `MAXOfferta`
    from `aste_attive`
    where `Nome` like var_nomeoggetto;

END$$

DELIMITER ;

-- -----
-- procedure visualizza_aste_attive
-- -----

DELIMITER $$
USE `sistemaaste`$$
CREATE PROCEDURE `visualizza_aste_attive` ()

BEGIN

    select `ID`, `Nome`, `CategoriaOld`, `MAXOfferta`
    from `aste_attive`;

END$$

DELIMITER ;

-- -----
-- procedure visualizza_aste_per_categoria
-- -----

DELIMITER $$
USE `sistemaaste`$$
CREATE PROCEDURE `visualizza_aste_per_categoria` (in categoria varchar(20))
BEGIN

    declare var_categoria varchar(20);

    set var_categoria = concat( "%", categoria, "%");

    select `ID`, `aste`.`Nome`, `CategoriaOld`, `MAXOfferta`
    from `aste_attive` as `aste`
    join `storicocategoria3` as `scat3` on `CategoriaOld` = `scat3`.`Nome`
    join `storicocategoria2` as `scat2` on `scat3`.`NomeCategoriaPadre` = `scat2`.`Nome`
    join `storicocategoria1` as `scat1` on `scat2`.`NomeCategoriaPadre` = `scat1`.`Nome`
    where `scat3`.`Nome` like var_categoria
    or `scat2`.`Nome` like var_categoria
    or `scat1`.`Nome` like var_categoria;

```

```

END$$

DELIMITER ;

-- -----
-- procedure visualizza_aste_per_espositore
-- -----

DELIMITER $$
USE `sistemaaste`$$
CREATE PROCEDURE `visualizza_aste_per_espositore` (in espositore char(16))

BEGIN

    select `Espositore`, `ID`, `Nome`, `CategoriaOld`, `MAXOfferta`, 'attiva'
        from `visualizzazione_aste`
        where `Espositore` = espositore and Scadenza > current_timestamp()
        union
        select `Espositore`, `ID`, `Nome`, `CategoriaOld`, `MAXOfferta`, 'terminata'
        from `visualizzazione_aste`
        where `Espositore` = espositore and Scadenza <= current_timestamp()
        ;

END$$

DELIMITER ;

-- -----
-- procedure visualizza_stato_asta
-- -----

DELIMITER $$
USE `sistemaaste`$$
CREATE PROCEDURE `visualizza_stato_asta` (in asta_id int)

BEGIN

    select `Espositore`, `Nome`, `Descrizione`, `CategoriaOld`, `Stato`, `Colore`,
        `Dimensioni`, `NOfferte`, `MAXOfferta`, `Scadenza`
        from `visualizzazione_aste`
        where `ID` = asta_id;

END$$

DELIMITER ;

-- -----
-- procedure visualizza_oggetti_aggiudicati
-- -----

DELIMITER $$
USE `sistemaaste`$$
CREATE PROCEDURE `visualizza_oggetti_aggiudicati` (in utente char(16))

BEGIN

    select `Nome`, `Descrizione`, `MAXOfferta`, `Scadenza`
        from `oggettoallasta`
        where `MigliorOfferente` = utente
            and `Scadenza` < current_timestamp();

END$$

DELIMITER ;

```

```

-- -----
-- procedure visualizza_partecipazione_aste
-- -----

DELIMITER $$
USE `sistemaaste`$$
CREATE PROCEDURE `visualizza_partecipazione_aste` (in utente char(16))

BEGIN

    select distinct `ID`, `Nome`, `Descrizione`, `Scadenza`
    from `offerta`
    join `oggettoallasta` on `Oggetto` = `ID`
    where `Utente` = utente;

END$$

DELIMITER ;

-- -----
-- procedure report_asta
-- -----

DELIMITER $$
USE `sistemaaste`$$
CREATE PROCEDURE `report_asta` (in id_asta int)

BEGIN

    select `Utente`, `Importo`, `Istante`, `Automatica`
    from `offerta`
    join `oggettoallasta` on `Oggetto` = `ID`
    where `Oggetto` = id_asta
    order by `Importo` desc;

END$$

DELIMITER ;

-- -----
-- procedure validazione_accesso
-- -----

DELIMITER $$
USE `sistemaaste`$$
CREATE PROCEDURE `validazione_accesso` (in var_cf char(16), in var_psw varchar(30), in
var_admin bool)

BEGIN

    declare var_compara_cf char(16);

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

    start transaction;

        select `CodiceFiscale`
        from `Login`
        where `CodiceFiscale` = `var_cf` and `Password` = `var_psw` and `Admin` =
`var_admin`
        into var_compara_cf;

        if var_compara_cf is null then

```

```

        signal sqlstate '45022'
        set message_text = "Accesso negato.";
        end if;

commit;

END$$

DELIMITER ;

-- -----
-- procedure visualizzazione_titolario_gerarchico
-- -----

DELIMITER $$
USE `sistemaaste`$$
CREATE PROCEDURE `visualizzazione_titolario_gerarchico` ()
BEGIN

    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$$

DELIMITER ;

-- -----
-- View `sistemaaste`.`aste_attive`
-- -----
DROP TABLE IF EXISTS `sistemaaste`.`aste_attive`;
USE `sistemaaste`;
CREATE OR REPLACE VIEW `aste_attive` AS
SELECT `ID`,`Nome`,`Espositore`,`CategoriaOld`,`Descrizione`,
`Dimensioni`,`Stato`,`Colore`,`PrezzoBase`,`NA°Offerte`,
`MAXofferta`,`Scadenza`
FROM `oggettoallasta`
WHERE `Scadenza` > current_timestamp();

-- -----
-- View `sistemaaste`.`visualizzazione_aste`
-- -----
DROP TABLE IF EXISTS `sistemaaste`.`visualizzazione_aste`;
USE `sistemaaste`;
CREATE OR REPLACE VIEW `visualizzazione_aste` AS
SELECT `ID`,`Nome`,`Espositore`,`CategoriaOld`,`Descrizione`,
`Dimensioni`,`Stato`,`Colore`,`PrezzoBase`,`NA°Offerte`,
`MAXofferta`,`Scadenza`
FROM `oggettoallasta`;

-- -----
-- View `sistemaaste`.`titolario_gerarchico_lower`

```

```

-----
DROP TABLE IF EXISTS `sistemaaste`.`titolario_gerarchico_lower`;
USE `sistemaaste`;
CREATE OR REPLACE VIEW `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.`NomeCategoriaPadre` = c2.`Nome`
    GROUP BY c2.`Nome`, c3.`Nome`

    UNION

    SELECT c3.`Nome` AS Nome, c2.`Nome` AS Padre
    FROM `categoria3` as c3 RIGHT JOIN `categoria2` as c2
    ON c3.`NomeCategoriaPadre` = c2.`Nome`
    GROUP BY c2.`Nome`, c3.`Nome`;

-- -----
-- View `sistemaaste`.`titolario_gerarchico_upper`
-- -----
DROP TABLE IF EXISTS `sistemaaste`.`titolario_gerarchico_upper`;
USE `sistemaaste`;
CREATE OR REPLACE VIEW `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.`NomeCategoriaPadre` = c1.`Nome`
    GROUP BY c1.`Nome`, c2.`Nome`

    UNION

    SELECT c2.`Nome` AS Nome, c1.`Nome` AS Padre
    FROM `categoria2` as c2 RIGHT JOIN `categoria1` as c1
    ON c2.`NomeCategoriaPadre` = c1.`Nome`
    GROUP BY c1.`Nome`, c2.`Nome`;
CREATE USER 'utentebase' IDENTIFIED BY 'totigimmi';

CREATE USER 'utenteamm' IDENTIFIED BY 'utenteamministratore';

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

```

## Codice del Front-End

### I. main.c

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <mysql.h>
#include "program.h"

extern bool *user_type;

```



```

extern MYSQL *con;
struct configuration conf;

UTENTE logged_user;

void menu_utente_base();

void menu_utente_admin();

int log_or_sign();

static void test_error(MYSQL * con, int status){
    if (status) {
        fprintf(stderr, "Error: %s (errno: %d)\n", mysql_error(con),
            mysql_errno(con));
        exit(1);
    }
}

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

    user_type = malloc(sizeof(bool));
    con = mysql_init(NULL);

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

    load_file(&config, "config.json");
    parse_config();
    if (con == NULL) {
        fprintf(stderr, "Initilization error: %s\n", mysql_error(con));
        exit(1);
    }
    if (mysql_real_connect(con, conf.host, conf.username, conf.password,
conf.database, conf.port, NULL, 0) == NULL) {
        fprintf(stderr, "Connection error: %s\n", mysql_error(con));
        exit(1);
    }

    //Operazioni sul Portale d'Accesso: Login / Sign in

    int logOrSign;
    portal: logOrSign = log_or_sign();

    switch((int)logOrSign){
        case 0: //login
            if (*user_type == 0){ //login -> admin
                //cambio utente: accesso al database per l'utente
amministratore

                if(mysql_change_user(con,"utenteamm","utenteamm","sistemaaste") != 0){
                    //inserire gestione errore connessione
                }
                menu_utente_admin();
            } else if (*user_type == 1){ // login -> standard user
                //cambio utente: accesso al database per l'utente base

                if(mysql_change_user(con,"utentebase","utentebase","sistemaaste") != 0){
                    //inserire gestione errore connessione
                }
            }
    }
}

```

```

        }
        menu_utente_base();
    }

    goto portal;

case 1: //sign in
    printf("\nUtente Registrato!");
    goto portal;

default: //quit
    break;
}

mysql_close(con);
free(user_type);
return 0;
}

```

## II. portale.c

```

#include <stdio.h>
#include <ctype.h>
#include <stdlib.h>
#include <string.h>
#include <mysql.h>
#include "program.h"

MYSQL *con;
bool *user_type;
UTENTE logged_user;

MYSQL_ROW row;
MYSQL_FIELD *field;
MYSQL_RES *rs_metadata;
MYSQL_STMT *stmt;
int i;
int num_fields; // number of columns in result
MYSQL_FIELD *fields; // for result set metadata
MYSQL_BIND *rs_bind; // for output buffers
int status;

unsigned int convert(char *st) {
    char *x;
    for (x = st ; *x ; x++) {
        if (!isdigit(*x))
            return 0L;
    }
    return (strtoul(st, 0L, 10));
}

bool bool_test_stmt_error(MYSQL_STMT * stmt, int status){
    if (status) {
        fprintf(stderr, "\n                                     [Error: %s (errno:
%d)]\n\n\n",
                mysql_stmt_error(stmt), mysql_stmt_errno(stmt));
        return 1;
    }
}

```





```

fflush(stdout);

char *_type = malloc(sizeof(char)*32);
printf("          TIPOLOGIA UTENTE [ admin:0   | base:1 ] : ");
fflush(stdout);
getInput(32, _type, false);
int *type = malloc(sizeof(int));
*type = atoi(_type);
length[0] = sizeof(int);
free(_type);

char *cf = malloc(sizeof(char)*16);
printf("\n          CODICE FISCALE : ");
fflush(stdout);
getInput(16, cf, false);
length[1] = 16;

char *nome = malloc(sizeof(char)*32);
printf("\n          NOME UTENTE : ");
fflush(stdout);
getInput(32, nome, false);
length[2] = strlen(nome);

char *cognome = malloc(sizeof(char)*32);
printf("\n          COGNOME : ");
fflush(stdout);
getInput(32, cognome, false);
length[3] = strlen(cognome);

//data di nascita suddivisa in 3 sezioni!!!!--

char *_anno = malloc(sizeof(char)*32);
printf("\n          ANNO DI NASCITA : ");
fflush(stdout);
getInput(32, _anno, false);
date->year = atoi(_anno);
free(_anno);

char *_mese = malloc(sizeof(char)*32);
printf("\n          MESE DI NASCITA : ");
fflush(stdout);
getInput(32, _mese, false);
date->month = atoi(_mese);
free(_mese);

char *_giorno = malloc(sizeof(char)*32);
printf("\n          GIORNO DI NASCITA : ");
fflush(stdout);
getInput(32, _giorno, false);
date->day = atoi(_giorno);
free(_giorno);

length[4] = sizeof(MYSQL_TIME);

//-----

char *birthplace = malloc(sizeof(char)*32);
printf("\n          LUOGO DI NASCITA : ");
fflush(stdout);
getInput(32, birthplace, false);
length[5] = strlen(birthplace);

char *numcarta = malloc(sizeof(char)*16);
printf("\n          NUMERO DI CARTA : ");
fflush(stdout);
getInput(16, numcarta, false);
length[6] = 16;

```

```

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

char *_scadanno = malloc(sizeof(char)*32);
printf("\n                ANNO SCADENZA : ");
fflush(stdout);
getInput(32, _scadanno, false);
scad_carta->year = atoi(_scadanno);
free(_scadanno);

char *_scadmese = malloc(sizeof(char)*32);
printf("\n                MESE SCADENZA : ");
fflush(stdout);
getInput(32, _scadmese, false);
scad_carta->month = atoi(_scadmese);
free(_scadmese);

length[7] = sizeof(MYSQL_TIME);

//-----

char *cvv = malloc(sizeof(char)*3);
printf("\n                CVV : ");
fflush(stdout);
getInput(3, cvv, false);
length[8] = 3;

char *indirizzo = malloc(sizeof(char)*32);
printf("\n                INDIRIZZO DI CONSEGNA : ");
fflush(stdout);
getInput(32, indirizzo, false);
length[9] = strlen(indirizzo);

char *pswd = malloc(sizeof(char)*32);
printf("\n                PASSWORD DI ACCESSO : ");
fflush(stdout);
getInput(32, pswd, false);
length[10] = strlen(pswd);

stmt = mysql_stmt_init(con);
if (!stmt) {
    printf("Could not initialize statement\n");
    exit(1);
}
status = mysql_stmt_prepare(stmt, "CALL registrazione_utente(?, ?, ?, ?, ?, ?, ?,
?, ?, ?, ?)", strlen("CALL registrazione_utente(?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)"));
bool_test_stmt_error(stmt, status);

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

ps_params[0].buffer_type = MYSQL_TYPE_LONG;
ps_params[0].buffer = type;
ps_params[0].buffer_length = sizeof(int);
ps_params[0].length = &length[0];
ps_params[0].is_null = 0;

ps_params[1].buffer_type = MYSQL_TYPE_VAR_STRING;
ps_params[1].buffer = cf;
ps_params[1].buffer_length = 16;
ps_params[1].length = &length[1];
ps_params[1].is_null = 0;

ps_params[2].buffer_type = MYSQL_TYPE_VAR_STRING;
ps_params[2].buffer = nome;
ps_params[2].buffer_length = 32;
ps_params[2].length = &length[2];

```

```
ps_params[2].is_null = 0;

ps_params[3].buffer_type = MYSQL_TYPE_VAR_STRING;
ps_params[3].buffer = cognome;
ps_params[3].buffer_length = 32;
ps_params[3].length = &length[3];
ps_params[3].is_null = 0;

ps_params[4].buffer_type = MYSQL_TYPE_DATE;
ps_params[4].buffer = date;
ps_params[4].buffer_length = 256;
ps_params[4].length = &length[4];
ps_params[4].is_null = 0;

ps_params[5].buffer_type = MYSQL_TYPE_VAR_STRING;
ps_params[5].buffer = birthplace;
ps_params[5].buffer_length = 32;
ps_params[5].length = &length[5];
ps_params[5].is_null = 0;

ps_params[6].buffer_type = MYSQL_TYPE_VAR_STRING;
ps_params[6].buffer = numcarta;
ps_params[6].buffer_length = 16;
ps_params[6].length = &length[6];
ps_params[6].is_null = 0;

ps_params[7].buffer_type = MYSQL_TYPE_DATE;
ps_params[7].buffer = scad_carta;
ps_params[7].buffer_length = 256; //provo a sparare alto. non funziona.
ps_params[7].length = &length[7];
ps_params[7].is_null = 0;

ps_params[8].buffer_type = MYSQL_TYPE_VAR_STRING;
ps_params[8].buffer = cvv;
ps_params[8].buffer_length = 3;
ps_params[8].length = &length[8];
ps_params[8].is_null = 0;

ps_params[9].buffer_type = MYSQL_TYPE_VAR_STRING;
ps_params[9].buffer = indirizzo;
ps_params[9].buffer_length = 32;
ps_params[9].length = &length[9];
ps_params[9].is_null = 0;

ps_params[10].buffer_type = MYSQL_TYPE_VAR_STRING;
ps_params[10].buffer = pswd;
ps_params[10].buffer_length = 32;
ps_params[10].length = &length[10];
ps_params[10].is_null = 0;

// bind input parameters
status = mysql_stmt_bind_param(stmt, ps_params);
bool_test_stmt_error(stmt, status);

// Run the stored procedure
status = mysql_stmt_execute(stmt); //segmentation fault
bool err = bool_test_stmt_error(stmt, status);
mysql_stmt_close(stmt);

free(type);
free(cf);
free(nome);
free(cognome);
free(date);
free(birthplace);
free(numcarta);
free(scad_carta);
```





```

correttamente        isvalid = valid(); //se ritorna 0 allora l'accesso è avvenuto

                        if (isvalid == 0){
                            return 0;
                        }

                        if(isvalid != 0)
                            goto portale;

        case 1:
            //sign in

            sign();

            return 1;

        default:

            return 2;
    }
}

```

### **III. menuutente.c - si consiglia di guardare direttamente al file nella cartella del codice**

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <mysql.h>
#include "program.h"

bool *user_type;
MYSQL *con;
UTENTE logged_user;

// Inizializzazione delle variabili per la gestione delle stored procedures
MYSQL_RES *result;
MYSQL_ROW row;
MYSQL_FIELD *field;
MYSQL_RES *rs_metadata;
my_bool is_null[3]; // output value nullability
int i;
int num_fields; // number of columns in result
MYSQL_FIELD *fields; // for result set metadata
MYSQL_BIND *rs_bind; // for output buffers

//gestione segnali di errori dal dalle procedure (sqlstate...)
static int test_stmt_error(MYSQL_STMT * stmt, int status){
    if (status) {
        fprintf(stderr, "Errore: %s (errno: %d)\n",
                mysql_stmt_error(stmt), mysql_stmt_errno(stmt));
        return(1);
    }
    else{
        return 0;
    }
}

```

```

//Ciclo di stampa degli output di una stored procedure, comune a tutte le operazioni.
void procedure_output(MYSQL_STMT * stmt, int status){

    MYSQL_TIME *date;

    // This is a general piece of code, to show how to deal with
    // generic stored procedures. A lighter version tailored to
    // a single specific stored procedure is shown below, for
    // the second stored procedure used in the application

    // process results until there are no more
    do {
        /* the column count is > 0 if there is a result set */
        /* 0 if the result is only the final status packet */
        num_fields = mysql_stmt_field_count(stmt);

        if (num_fields > 0) {

            // what kind of result set is this?
            if (con->server_status & SERVER_PS_OUT_PARAMS)
                //gestione della stampa parametri I/O stored procedure:
                printf("The stored procedure has returned output in
OUT/INOUT parameter(s)\n");

            // Get information about the outcome of the stored procedure
            rs_metadata = mysql_stmt_result_metadata(stmt);
            if(test_stmt_error(stmt, rs_metadata == NULL) == 1){
                printf("\nPremi invio per continuare...\n");
                while(getchar() != '\n'){}
                return;
            }

            // Retrieve the fields associated with OUT/INOUT parameters
            fields = mysql_fetch_fields(rs_metadata);
            rs_bind = (MYSQL_BIND *)malloc(sizeof(MYSQL_BIND) * num_fields);
            if (!rs_bind) {
                printf("Cannot allocate output buffers\n");
                exit(1);
            }
            //initialize result set_bind structure allocated as null (or
zero)
            memset(rs_bind, 0, sizeof(MYSQL_BIND) * num_fields);

            // set up and bind result set output buffers
            for (i = 0; i < num_fields; ++i) {

                //void *memory = malloc(fields[i].length);
                void *memory = malloc(256); //numero grosso di byte per
contenere chiunque dei campi di una tabella!

                rs_bind[i].buffer_type = fields[i].type;
                rs_bind[i].is_null = &is_null[i];

                switch (fields[i].type) {
                    //associazione del type della colonna i al
corrispettivo tipo C (parsing)
                    //inoltre dice dove va a finire l'elemento di
ogni riga corrispondente a quella colonna:
                    // cio implica che se due colonne sono di tipo
stringa punteranno allo stesso indirizzo di memoria (&token[0])
                    //bisogna risolverlo!

                    case MYSQL_TYPE_VAR_STRING:
                        rs_bind[i].buffer = memory;
                        rs_bind[i].buffer_length =
fields[i].length;

                        break;

```

```

fields[i].length;

case MYSQL_TYPE_TINY:
    rs_bind[i].buffer = memory;
    rs_bind[i].buffer_length =

break;

case MYSQL_TYPE_STRING:
    rs_bind[i].buffer = memory;
    rs_bind[i].buffer_length =

fields[i].length;

break;

case MYSQL_TYPE_LONG:
    rs_bind[i].buffer = memory;
    rs_bind[i].buffer_length =

fields[i].length;

break;

case MYSQL_TYPE_TIMESTAMP:
    rs_bind[i].buffer = memory;
    rs_bind[i].buffer_length =

fields[i].length;

break;

case MYSQL_TYPE_DATETIME:
    rs_bind[i].buffer = memory;
    rs_bind[i].buffer_length =

fields[i].length;

break;

case MYSQL_TYPE_BLOB:
    rs_bind[i].buffer = memory;
    rs_bind[i].buffer_length =

fields[i].length;

break;

case MYSQL_TYPE_SHORT:
    rs_bind[i].buffer = memory;
    rs_bind[i].buffer_length =

fields[i].length;

break;

case MYSQL_TYPE_NEWDECIMAL:
    rs_bind[i].buffer = memory;
    rs_bind[i].buffer_length =

fields[i].length;

break;

case MYSQL_TYPE_FLOAT:
    rs_bind[i].buffer = memory;
    rs_bind[i].buffer_length =

fields[i].length;

break;

case MYSQL_TYPE_DOUBLE:
    rs_bind[i].buffer = memory;
    rs_bind[i].buffer_length =

fields[i].length;

break;

default:
    fprintf(stderr, "ERROR: unexpected
type column %d: %d.\n", i, fields[i].type);
    exit(1);

```

```

    }

}

status = mysql_stmt_bind_result(stmt, rs_bind);
if(test_stmt_error(stmt, status) == 1){
    printf("\nPremi invio per continuare...\n");
    while(getchar() != '\n'){}
    return;
}

for (i = 0; i < num_fields; ++i) {
    printf("          Colonna.%d", i);    //
(strlen = 30)
}

printf("\n\n");
// fetch and display result set rows
while (1) {

    status = mysql_stmt_fetch(stmt);

    if (status == 1 || status == MYSQL_NO_DATA)
        break;

    for (i = 0; i < num_fields; ++i) {
        switch (rs_bind[i].buffer_type) {
            case MYSQL_TYPE_VAR_STRING:
                if (*rs_bind[i].is_null){
                    printf("%30s",
"NULL");
                    fflush(stdout);
                }
                else{
                    printf("%30s", (char*)
rs_bind[i].buffer);
                    fflush(stdout);
                }
                break;

            case MYSQL_TYPE_DATE:
                if (*rs_bind[i].is_null){
                    printf("%30s",
"NULL");
                    fflush(stdout);
                }
                else{
                    date =
                    printf("%15d-%d-%d
%d:%d:%d", date->year, date->month, date->day, date->hour, date->minute, date->second);
                    fflush(stdout);
                }
                break;

            case MYSQL_TYPE_STRING:
                if (*rs_bind[i].is_null){
                    printf("%30s",
"NULL");
                    fflush(stdout);
                }
                else{

```

```

rs_bind[i].buffer);

printf("%30s", (char*)
fflush(stdout);
}
break;

case MYSQL_TYPE_FLOAT:
    if (*rs_bind[i].is_null){
        printf("%30s",

fflush(stdout);
    }
    else{
        printf("%30.2f",

fflush(stdout);
    }
    break;

case MYSQL_TYPE_DOUBLE:
    if (*rs_bind[i].is_null){
        printf("%30s",

fflush(stdout);
    }
    else{
        printf("%30.2f",

    }
    break;

case MYSQL_TYPE_LONG:
    if (*rs_bind[i].is_null){
        printf("%30s",

fflush(stdout);
    }
    else{
        printf("%30d", *(int*)

fflush(stdout);
    }
    break;

case MYSQL_TYPE_TIMESTAMP:
    if (*rs_bind[i].is_null){
        printf("%30s",

fflush(stdout);
    }
    else{
        date =

        printf("%21d-%d-%d
%d:%d:%d",date->year, date->month, date->day, date->hour, date->minute, date->second);
    }
    break;

case MYSQL_TYPE_BLOB:
    if (*rs_bind[i].is_null){
        printf("%30s",

fflush(stdout);
    }
    else{
        printf("%30s", (char*)

fflush(stdout);

```

```

        break;
    }
    break;

case MYSQL_TYPE_DATETIME:
    if (*rs_bind[i].is_null){
        printf("%30s",

"NULL");

        fflush(stdout);
    }
    else{
        printf("%30s", (char*)

rs_bind[i].buffer);

        fflush(stdout);
    }
    break;

case MYSQL_TYPE_SHORT:
    if (*rs_bind[i].is_null){
        printf("%30s",

"NULL");

        fflush(stdout);
    }
    else{
        printf("%30d", *(int*)

rs_bind[i].buffer);

        fflush(stdout);
    }
    break;

case MYSQL_TYPE_TINY:
    if (*rs_bind[i].is_null){
        printf("%30s",

"NULL");

        fflush(stdout);
    }
    else{
        printf("%30d", *(int*)

rs_bind[i].buffer);

        fflush(stdout);
    }
    break;

case MYSQL_TYPE_NEWDECIMAL:
    if (*rs_bind[i].is_null){
        printf("%30s",

"NULL");

        fflush(stdout);
    }
    else{
        printf("%30.6lf",

*(float*) rs_bind[i].buffer);

        fflush(stdout);
    }
    break;

default:
    printf("ERROR: unexpected type

(%d)\n", rs_bind[i].buffer_type);

    }
    printf("\n");
}
mysql_free_result(rs_metadata); // free metadata

for (int j = 0; j < num_fields; ++j){

```

```

        free(rs_bind[j].buffer);
    }

    free(rs_bind);    // free output buffers
} else {
    // no columns = final status packet
    printf("\n\n%20s", "-----fine dell'output-----
-----\n");
}

// more results? -1 = no, >0 = error, 0 = yes (keep looking)
status = mysql_stmt_next_result(stmt);
if (status > 0)
    if(test_stmt_error(stmt, status) == 1){
        printf("\nPremi invio per continuare...\n");
        while(getchar() != '\n'){}
        return;
    }
} while (status == 0);

mysql_stmt_close(stmt);
}

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

void op_a1();

//RICERCA ASTA PER NOME OGGETTO
void op_1(){

    MYSQL_STMT *stmt;
    int status;
    MYSQL_BIND ps_params[1];    // input parameter buffers
    unsigned long length[6];    // Can do like that because all IN parameters have the
    same length

    top:
    //titolo : operazione
    printf("\e[1;1H\e[2J");
    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);
    length[0] = strlen(nome);

    //inizializzazione statement procedurale.
    stmt = mysql_stmt_init(con);
    if (!stmt) {
        printf("Could not initialize statement\n");
        exit(1);
    }

    status = mysql_stmt_prepare(stmt, "CALL visualizza_aste_per_nome_oggetto(?)",
strlen("CALL visualizza_aste_per_nome_oggetto(?));
    if(test_stmt_error(stmt, status) == 1){
        printf("\nPremi invio per continuare...\n");
        while(getchar() != '\n'){}
        goto top;
    }
}

```

```

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

// `visualizza_aste_per_nome_oggetto` (in nomeoggetto varchar(20))
ps_params[0].buffer_type = MYSQL_TYPE_VAR_STRING;
ps_params[0].buffer = nome;
ps_params[0].buffer_length = 64;
ps_params[0].length = &length[0];
ps_params[0].is_null = 0;

// bind input parameters
status = mysql_stmt_bind_param(stmt, ps_params);
if(test_stmt_error(stmt, status) == 1){
    printf("\nPremi invio per continuare...\n");
    while(getchar() != '\n'){}
    goto top;
}
// Run the stored procedure
status = mysql_stmt_execute(stmt);
if(test_stmt_error(stmt, status) == 1){
    printf("\nPremi invio per continuare...\n");
    while(getchar() != '\n'){}
    goto top;
}

procedure_output( stmt, status);
}

//VISUALIZZAZIONE ASTE ATTIVE
void op_2(){

    MYSQL_STMT *stmt;
    int status;

    top:
    //titolo : operazione
    printf("\e[1;1H\e[2J");
    printf("\n\n**** VISUALIZZAZIONE ASTE ATTIVE ****\n\n");

    //inizializzazione statement procedurale.
    stmt = mysql_stmt_init(con);
    if (!stmt) {
        printf("Could not initialize statement\n");
        exit(1);
    }
    status = mysql_stmt_prepare(stmt, "CALL visualizza_aste_attive()", strlen("CALL
visualizza_aste_attive()"));
    if(test_stmt_error(stmt, status) == 1){
        printf("\nPremi invio per continuare...\n");
        while(getchar() != '\n'){}
        goto top;
    }

    // Run the stored procedure
    status = mysql_stmt_execute(stmt);
    if(test_stmt_error(stmt, status) == 1){
        printf("\nPremi invio per continuare...\n");
        while(getchar() != '\n'){}
        return;
    }
    procedure_output(stmt, status);
}

//VISUALIZZAZIONE STATO DI UN ASTA

```



```

void op_3(){
    MYSQL_STMT *stmt;
    int status;
    MYSQL_BIND ps_params[1];          // input parameter buffers
    unsigned long length[1];          // Can do like that because all IN parameters have the
    same length

    top:
    //titolo : operazione
    printf("\e[1;1H\e[2J");
    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);
    *id = atoi(_id);
    free(_id);
    length[0] = sizeof(int);

    //inizializzazione statement procedurale.
    stmt = mysql_stmt_init(con);
    if (!stmt) {
        printf("Could not initialize statement\n");
        exit(1);
    }

    status = mysql_stmt_prepare(stmt, "CALL visualizza_stato_asta(?)", strlen("CALL
visualizza_stato_asta(?)));
    if(test_stmt_error(stmt, status) == 1){
        printf("\nPremi invio per continuare...\n");
        while(getchar() != '\n'){}
        free(id);
        goto top;
    }

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

    // `visualizza_stato_asta` (in id int)
    ps_params[0].buffer_type = MYSQL_TYPE_LONG;
    ps_params[0].buffer = id;
    ps_params[0].buffer_length = sizeof(int);
    ps_params[0].length = &length[0];
    ps_params[0].is_null = 0;

    // bind input parameters
    status = mysql_stmt_bind_param(stmt, ps_params);
    if(test_stmt_error(stmt, status) == 1){
        printf("\nPremi invio per continuare...\n");
        while(getchar() != '\n'){}
        free(id);
        goto top;
    }

    // Run the stored procedure
    status = mysql_stmt_execute(stmt);
    if(test_stmt_error(stmt, status) == 1){
        printf("\nPremi invio per continuare...\n");
        while(getchar() != '\n'){}
        free(id);
        return;
    }
}

```

```

    procedure_output( stmt, status);
    free(id);
}

//VISUALIZZAZIONE OGGETTI AGGIUDICATI
void op_4() {

    MYSQL_STMT *stmt;
    int status;
    MYSQL_BIND ps_params[1];      // input parameter buffers
    unsigned long length[1];      // Can do like that because all IN parameters have the
    same length

    top:
    //titolo : operazione
    printf("\e[1;1H\e[2J");
    printf("\n\n**** VISUALIZZAZIONE OGGETTI AGGIUDICATI ****\n\n");

    //define input variable
    length[0] = 16;

    //inizializzazione statement procedurale.
    stmt = mysql_stmt_init(con);
    if (!stmt) {
        printf("Could not initialize statement\n");
        exit(1);
    }

    status = mysql_stmt_prepare(stmt, "CALL visualizza_oggetti_aggiudicati(?)",
    strlen("CALL visualizza_oggetti_aggiudicati(?)));
    if(test_stmt_error(stmt, status) == 1){
        printf("\nPremi invio per continuare...\n");
        while(getchar() != '\n') {}
        goto top;
    }

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

    // `visualizza_stato_asta` (in id int)
    ps_params[0].buffer_type = MYSQL_TYPE_STRING;
    ps_params[0].buffer = logged_user.cf;
    ps_params[0].buffer_length = 16;
    ps_params[0].length = &length[0];
    ps_params[0].is_null = 0;

    // bind input parameters
    status = mysql_stmt_bind_param(stmt, ps_params);
    if(test_stmt_error(stmt, status) == 1){
        printf("\nPremi invio per continuare...\n");
        while(getchar() != '\n') {}
        goto top;
    }

    // Run the stored procedure
    status = mysql_stmt_execute(stmt);
    if(test_stmt_error(stmt, status) == 1){
        printf("\nPremi invio per continuare...\n");
        while(getchar() != '\n') {}
        return;
    }
    procedure_output( stmt, status);
}

```

```

}

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

    MYSQL_STMT *stmt;
    int status;
    MYSQL_BIND ps_params[4];          // input parameter buffers
    unsigned long length[4];          // Can do like that because all IN parameters have the
    same length

    top:
    //titolo : operazione
    printf("\e[1;1H\e[2J");
    printf("\n\n**** REGISTRAZIONE OFFERTA SU UN'ASTA ****\n\n");

    //define input variable
    length[0] = 16;

    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);
    length[1] = sizeof(int);
    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);
    length[2] = sizeof(float);
    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);
    length[2] = sizeof(float);
    free(_importo_controff);

    //inizializzazione statement procedurale.
    stmt = mysql_stmt_init(con);
    if (!stmt) {
        printf("Could not initialize statement\n");
        exit(1);
    }

    status = mysql_stmt_prepare(stmt, "CALL registra_offerta(?, ?, ?, ?)",
    strlen("CALL registra_offerta(?, ?, ?, ?)"));
    if(test_stmt_error(stmt, status) == 1){
        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));

```

```

// `visualizza_stato_asta` (in id int)
ps_params[0].buffer_type = MYSQL_TYPE_STRING;
ps_params[0].buffer = logged_user.cf;
ps_params[0].buffer_length = 16;
ps_params[0].length = &length[0];
ps_params[0].is_null = 0;

ps_params[1].buffer_type = MYSQL_TYPE_LONG;
ps_params[1].buffer = id;
ps_params[1].buffer_length = sizeof(int);
ps_params[1].length = &length[1];
ps_params[1].is_null = 0;

ps_params[2].buffer_type = MYSQL_TYPE_FLOAT;
ps_params[2].buffer = offerta;
ps_params[2].buffer_length = sizeof(float);
ps_params[2].length = &length[2];
ps_params[2].is_null = 0;

ps_params[3].buffer_type = MYSQL_TYPE_FLOAT;
ps_params[3].buffer = importo_controff;
ps_params[3].buffer_length = sizeof(float);
ps_params[3].length = &length[3];
ps_params[3].is_null = 0;

// bind input parameters
status = mysql_stmt_bind_param(stmt, ps_params);
if(test_stmt_error(stmt, status) == 1){
    printf("\nPremi invio per continuare...\n");
    while(getchar() != '\n'){}
    free(id);
    free(offerta);
    free(importo_controff);
    goto top;
}

// Run the stored procedure
status = mysql_stmt_execute(stmt);
if(test_stmt_error(stmt, status) == 1){
    printf("\nPremi invio per continuare...\n");
    while(getchar() != '\n'){}
    free(id);
    free(offerta);
    free(importo_controff);
    return;
}

printf("L'offerta è stata regolarmente registrata.\n");

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

//RICERCA ASTE PER ESPOSITORE
void op_6(){
    MYSQL_STMT *stmt;
    int status;
    MYSQL_BIND ps_params[1]; // input parameter buffers
    unsigned long length[1]; // Can do like that because all IN parameters have the
    same length

    top:
    //titolo : operazione

```

```

    printf("\e[1;1H\e[2J");
    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);
    length[0] = 16;

    //inizializzazione statement procedurale.
    stmt = mysql_stmt_init(con);
    if (!stmt) {
        printf("Could not initialize statement\n");
        exit(1);
    }

    status = mysql_stmt_prepare(stmt, "CALL visualizza_aste_per_espositore(?)",
strlen("CALL visualizza_aste_per_espositore(?"));
    if(test_stmt_error(stmt, status) == 1){
        printf("\nPremi invio per continuare...\n");
        while(getchar() != '\n'){}
        free(cf_esp);
        goto top;
    }

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

    // `visualizza_stato_asta` (in id int)
    ps_params[0].buffer_type = MYSQL_TYPE_STRING;
    ps_params[0].buffer = cf_esp;
    ps_params[0].buffer_length = 16;
    ps_params[0].length = &length[0];
    ps_params[0].is_null = 0;

    // bind input parameters
    status = mysql_stmt_bind_param(stmt, ps_params);
    if(test_stmt_error(stmt, status) == 1){
        printf("\nPremi invio per continuare...\n");
        while(getchar() != '\n'){}
        free(cf_esp);
        goto top;
    }

    // Run the stored procedure
    status = mysql_stmt_execute(stmt);
    if(test_stmt_error(stmt, status) == 1){
        printf("\nPremi invio per continuare...\n");
        while(getchar() != '\n'){}
        free(cf_esp);
        return;
    }

    procedure_output(stmt, status);

    free(cf_esp);
}

//RICERCA ASTE PER CATEGORIA
void op_7(){

```

```

        MYSQL_STMT *stmt;
int status;
MYSQL_BIND ps_params[1];        // input parameter buffers
unsigned long length[6];        // Can do like that because all IN parameters have the
same length

        top:
//titolo : operazione
        printf("\e[1;1H\e[2J");
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);
length[0] = strlen(categoria);

//inizializzazione statement procedurale.
stmt = mysql_stmt_init(con);
if (!stmt) {
        printf("Could not initialize statement\n");
        exit(1);
}

        status = mysql_stmt_prepare(stmt, "CALL visualizza_aste_per_categoria(?)",
strlen("CALL visualizza_aste_per_categoria(?"));
if(test_stmt_error(stmt, status) == 1){
        printf("\nPremi invio per continuare...\n");
        while(getchar() != '\n'){}
        free(categoria);
        goto top;
}

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

// `visualizza_aste_per_nomeoggetto` (in nomeoggetto varchar(20))
ps_params[0].buffer_type = MYSQL_TYPE_VAR_STRING;
ps_params[0].buffer = categoria;
ps_params[0].buffer_length = 64;
ps_params[0].length = &length[0];
ps_params[0].is_null = 0;

// bind input parameters
status = mysql_stmt_bind_param(stmt, ps_params);
if(test_stmt_error(stmt, status) == 1){
        printf("\nPremi invio per continuare...\n");
        while(getchar() != '\n'){}
        free(categoria);
        goto top;
}

// Run the stored procedure
status = mysql_stmt_execute(stmt);
if(test_stmt_error(stmt, status) == 1){
        printf("\nPremi invio per continuare...\n");
        while(getchar() != '\n'){}
        free(categoria);
        return;
}

        procedure_output(stmt, status);

        free(categoria);
}

//RICERCA ASTE PER PARTECIPAZIONE
void op_9(){

```

```

        MYSQL_STMT *stmt;
        int status;
        MYSQL_BIND ps_params[1];          // input parameter buffers
        unsigned long length[1];          // Can do like that because all IN parameters have the
        same length

        top:
        //titolo : operazione
        printf("\e[1;1H\e[2J");
        printf("\n\n**** RICERCA ASTE PER PARTECIPAZIONE ****\n\n");

        //define input variable
        length[0] = 16;

        //inizializzazione statement procedurale.
        stmt = mysql_stmt_init(con);
        if (!stmt) {
            printf("Could not initialize statement\n");
            exit(1);
        }

        status = mysql_stmt_prepare(stmt, "CALL visualizza_partecipazione_aste (?)",
        strlen("CALL visualizza_partecipazione_aste (?)"));
        if(test_stmt_error(stmt, status) == 1){
            printf("\nPremi invio per continuare...\n");
            while(getchar() != '\n'){}
            goto top;
        }

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

        // `visualizza_stato_asta` (in id int)
        ps_params[0].buffer_type = MYSQL_TYPE_STRING;
        ps_params[0].buffer = logged_user.cf;
        ps_params[0].buffer_length = 16;
        ps_params[0].length = &length[0];
        ps_params[0].is_null = 0;

        // bind input parameters
        status = mysql_stmt_bind_param(stmt, ps_params);
        if(test_stmt_error(stmt, status) == 1){
            printf("\nPremi invio per continuare...\n");
            while(getchar() != '\n'){}
            goto top;
        }

        // Run the stored procedure
        status = mysql_stmt_execute(stmt);
        if(test_stmt_error(stmt, status) == 1){
            printf("\nPremi invio per continuare...\n");
            while(getchar() != '\n'){}
            return;
        }
        procedure_output( stmt, status);
    }

//INTERFACCIA UTENTE : UTENTE BASE
void menu_utente_base(){

    char *operation;
    start: operation = malloc(64*sizeof(char));

```

```

    int op_code;
    printf("\e[1;1H\e[2J");

printf(".....
..\n");
    printf(".....
.....\n");
    printf(".....|          MENU UTENTE
|.....\n");
    printf(".....\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 per categoria di afferenza
|\n");
    printf("|   OPERAZIONE 8 : Visualizzazione categorie
|\n");
    printf("|   OPERAZIONE 9 : Visualizzazione delle aste a cui ho partecipato
|\n");
    printf("|_____
_____ |\n");
    printf("Inserisci il codice dell'operazione : ");
    getInput(64, operation, false);
    op_code = atoi(operation);
    switch((int)op_code){
        case 0:
            break;

        case 1:
            op_1();
            printf("\n\n Premi invio per tornare al Menù Utente...\n");
            while(getchar() != '\n'){}
            free(operation);
            printf("\e[1;1H\e[2J");
            goto start;
            break;

        case 2:
            op_2();
            printf("\n\n Premi invio per tornare al Menù Utente...\n");
            while(getchar() != '\n'){}
            free(operation);
            printf("\e[1;1H\e[2J");
            goto start;
            break;

        case 3:

```



```
op_3();
printf("\n\n Premi invio per tornare al Menù Utente...\n");
while(getchar() != '\n'){}
free(operation);
    printf("\e[1;1H\e[2J");
goto start;
break;

    case 4:
op_4();
printf("\n\n Premi invio per tornare al Menù Utente...\n");
while(getchar() != '\n'){}
free(operation);
    printf("\e[1;1H\e[2J");
goto start;
break;

    case 5:
op_5();
printf("\n\n Premi invio per tornare al Menù Utente...\n");
while(getchar() != '\n'){}
free(operation);
    printf("\e[1;1H\e[2J");
goto start;
break;

    case 6:
op_6();
printf("\n\n Premi invio per tornare al Menù Utente...\n");
while(getchar() != '\n'){}
free(operation);
    printf("\e[1;1H\e[2J");
goto start;
break;

    case 7:
op_7();
printf("\n\n Premi invio per tornare al Menù Utente...\n");
while(getchar() != '\n'){}
free(operation);
    printf("\e[1;1H\e[2J");
goto start;
break;

    case 8:
op_8();
printf("\n\n Premi invio per tornare al Menù Utente...\n");
while(getchar() != '\n'){}
free(operation);
    printf("\e[1;1H\e[2J");
goto start;
break;

    case 9:
op_9();
printf("\n\n Premi invio per tornare al Menù Utente...\n");
while(getchar() != '\n'){}
free(operation);
    printf("\e[1;1H\e[2J");
goto start;
break;

default:
    free(operation);
    printf("\e[1;1H\e[2J");
goto start;
break;
```

```

    }
    return;
}

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

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

//VISUALIZZA TITOLARIO GERARCHICO
void op_al() {

    MYSQL_STMT *stmt;
    int status;
    MYSQL_BIND ps_params[1];      // input parameter buffers
    unsigned long length[6];      // Can do like that because all IN parameters have the
    same length

    top:
    //titolo : operazione
    printf("\e[1;1H\e[2J");
    printf("\n\n      **** VISUALIZZAZIONE CATEGORIE : TITOLARIO GERARCHICO ****\n\n");

    //inizializzazione statement procedurale.
    stmt = mysql_stmt_init(con);
    if (!stmt) {
        printf("Could not initialize statement\n");
        exit(1);
    }

    status = mysql_stmt_prepare(stmt, "CALL visualizzazione_titolario_gerarchico()",
    strlen("CALL visualizzazione_titolario_gerarchico()"));
    if(test_stmt_error(stmt, status) == 1){
        printf("\nPremi invio per continuare...\n");
        while(getchar() != '\n') {}
        goto top;
    }

    // bind input parameters
    status = mysql_stmt_bind_param(stmt, ps_params);
    if(test_stmt_error(stmt, status) == 1){
        printf("\nPremi invio per continuare...\n");
        while(getchar() != '\n') {}
        goto top;
    }

    // Run the stored procedure
    status = mysql_stmt_execute(stmt);
    if(test_stmt_error(stmt, status) == 1){
        printf("\nPremi invio per continuare...\n");
        while(getchar() != '\n') {}
        goto top;
    }

    procedure_output( stmt, status);
}

```



```

char nome[64];
printf("\nInserire il nome della nuova categoria: ");
getInput(64, nome, false);
length[0] = strlen(nome);

char nomePadre[64];
printf("\nInserire il nome della categoria genitore: ");
getInput(64, nomePadre, false);
length[1] = strlen(nomePadre);

//inizializzazione statement procedurale.
stmt = mysql_stmt_init(con);
if (!stmt) {
    printf("Could not initialize statement\n");
    exit(1);
}

status = mysql_stmt_prepare(stmt, "CALL
inserimento_categoria3(?, ?)", strlen("CALL inserimento_categoria3(?, ?)"));
if(test_stmt_error(stmt, status) == 1){
    printf("\nPremi invio per continuare...\n");
    while(getchar() != '\n'){}
    goto top;
}

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

// `visualizza_aste_per_nome_oggetto` (in nomeoggetto
varchar(20))

ps_params[0].buffer_type = MYSQL_TYPE_VAR_STRING;
ps_params[0].buffer = &nome;
ps_params[0].buffer_length = 64;
ps_params[0].length = &length[0];
ps_params[0].is_null = 0;

ps_params[1].buffer_type = MYSQL_TYPE_VAR_STRING;
ps_params[1].buffer = &nomePadre;
ps_params[1].buffer_length = 64;
ps_params[1].length = &length[1];
ps_params[1].is_null = 0;

// bind input parameters
status = mysql_stmt_bind_param(stmt, ps_params);
if(test_stmt_error(stmt, status) == 1){
    printf("\nPremi invio per continuare...\n");
    while(getchar() != '\n'){}
    goto top;
}
// Run the stored procedure
status = mysql_stmt_execute(stmt);
if(test_stmt_error(stmt, status) == 1){
    printf("\nPremi invio per continuare...\n");
    while(getchar() != '\n'){}
    goto top;
}
} else{

    MYSQL_BIND ps_params[1]; // input parameter buffers
    unsigned long length[1]; // Can do like that because all IN
parameters have the same length

//define input variable
char nome[64];

```

```

printf("\nInserire il nome della nuova categoria: ");
getInput(64, nome, false);
length[0] = strlen(nome);

//inizializzazione statement procedurale.
stmt = mysql_stmt_init(con);
if (!stmt) {
    printf("Could not initialize statement\n");
    exit(1);
}

status = mysql_stmt_prepare(stmt, "CALL
inserimento_categoria3(?, null)", strlen("CALL inserimento_categoria3(?, null)"));
if(test_stmt_error(stmt, status) == 1){
    printf("\nPremi invio per continuare...\n");
    while(getchar() != '\n'){}
    goto top;
}

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

// `visualizza_aste_per_nome_oggetto` (in nomeoggetto
varchar(20))

ps_params[0].buffer_type = MYSQL_TYPE_VAR_STRING;
ps_params[0].buffer = &nome;
ps_params[0].buffer_length = 64;
ps_params[0].length = &length[0];
ps_params[0].is_null = 0;

// bind input parameters
status = mysql_stmt_bind_param(stmt, ps_params);
if(test_stmt_error(stmt, status) == 1){
    printf("\nPremi invio per continuare...\n");
    while(getchar() != '\n'){}
    goto top;
}

// Run the stored procedure
status = mysql_stmt_execute(stmt);
if(test_stmt_error(stmt, status) == 1){
    printf("\nPremi invio per continuare...\n");
    while(getchar() != '\n'){}
    goto top;
}
}
return;
break;

case 2:

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

printf("Si desidera associare la categoria a un genitore al
livello 3 ?\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
    unsigned long length[2]; // Can do like that because all IN
parameters have the same length

    //define input variable
    char nome[64];
    printf("\nInserire il nome della nuova categoria: ");
    getInput(64, nome, false);
    length[0] = strlen(nome);

    char nomePadre[64];
    printf("\nInserire il nome della categoria genitore: ");
    getInput(64, nomePadre, false);
    length[1] = strlen(nomePadre);

    //inizializzazione statement procedurale.
    stmt = mysql_stmt_init(con);
    if (!stmt) {
        printf("Could not initialize statement\n");
        exit(1);
    }

    status = mysql_stmt_prepare(stmt, "CALL
inserimento_categoria2(?, ?)", strlen("CALL inserimento_categoria2(?, ?)"));
    if(test_stmt_error(stmt, status) == 1){
        printf("\nPremi invio per continuare...\n");
        while(getchar() != '\n'){}
        goto top;
    }

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

    // `visualizza_aste_per_nome_oggetto` (in nomeoggetto
varchar(20))

    ps_params[0].buffer_type = MYSQL_TYPE_VAR_STRING;
    ps_params[0].buffer = &nome;
    ps_params[0].buffer_length = 64;
    ps_params[0].length = &length[0];
    ps_params[0].is_null = 0;

    ps_params[1].buffer_type = MYSQL_TYPE_VAR_STRING;
    ps_params[1].buffer = &nomePadre;
    ps_params[1].buffer_length = 64;
    ps_params[1].length = &length[1];
    ps_params[1].is_null = 0;

    // bind input parameters
    status = mysql_stmt_bind_param(stmt, ps_params);
    if(test_stmt_error(stmt, status) == 1){
        printf("\nPremi invio per continuare...\n");
        while(getchar() != '\n'){}
        goto top;
    }
    // Run the stored procedure
    status = mysql_stmt_execute(stmt);
    if(test_stmt_error(stmt, status) == 1){
        printf("\nPremi invio per continuare...\n");
        while(getchar() != '\n'){}
        goto top;
    }
} else{

```

```

        MYSQL_BIND ps_params[1]; // input parameter buffers
        unsigned long length[1]; // Can do like that because all IN
parameters have the same length

        //define input variable
        char nome[64];
        printf("\nInserire il nome della nuova categoria: ");
        getInput(64, nome, false);
        length[0] = strlen(nome);

        //inizializzazione statement procedurale.
        stmt = mysql_stmt_init(con);
        if (!stmt) {
            printf("Could not initialize statement\n");
            exit(1);
        }

        status = mysql_stmt_prepare(stmt, "CALL
inserimento_categoria2(?, null)", strlen("CALL inserimento_categoria2(?, null)"));
        if(test_stmt_error(stmt, status) == 1){
            printf("\nPremi invio per continuare...\n");
            while(getchar() != '\n'){}
            goto top;
        }

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

        // `visualizza_aste_per_nome_oggetto` (in nomeoggetto
varchar(20))

        ps_params[0].buffer_type = MYSQL_TYPE_VAR_STRING;
        ps_params[0].buffer = &nome;
        ps_params[0].buffer_length = 64;
        ps_params[0].length = &length[0];
        ps_params[0].is_null = 0;

        // bind input parameters
        status = mysql_stmt_bind_param(stmt, ps_params);
        if(test_stmt_error(stmt, status) == 1){
            printf("\nPremi invio per continuare...\n");
            while(getchar() != '\n'){}
            goto top;
        }

        // Run the stored procedure
        status = mysql_stmt_execute(stmt);
        if(test_stmt_error(stmt, status) == 1){
            printf("\nPremi invio per continuare...\n");
            while(getchar() != '\n'){}
            goto top;
        }
    }
    return;
    break;

case 3:

    printf("\e[1;1H\e[2J");
    printf("INSERIMENTO CATEGORIA: LIVELLO 3 \n");

    MYSQL_BIND ps_params[1]; // input parameter buffers
    unsigned long length[1]; // Can do like that because all IN
parameters have the same length

```

```

//define input variable
char nome[64];
printf("\nInserire il nome della nuova categoria: ");
getInput(64, nome, false);
length[0] = strlen(nome);

//inizializzazione statement procedurale.
stmt = mysql_stmt_init(con);
if (!stmt) {
    printf("Could not initialize statement\n");
    exit(1);
}

status = mysql_stmt_prepare(stmt, "CALL
inserimento_categorial(?)", strlen("CALL inserimento_categorial(?)));
if(test_stmt_error(stmt, status) == 1){
    printf("\nPremi invio per continuare...\n");
    while(getchar() != '\n'){}
    goto top;
}

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

// `visualizza_aste_per_nome_oggetto` (in nomeoggetto
varchar(20))

ps_params[0].buffer_type = MYSQL_TYPE_VAR_STRING;
ps_params[0].buffer = &nome;
ps_params[0].buffer_length = 64;
ps_params[0].length = &length[0];
ps_params[0].is_null = 0;

// bind input parameters
status = mysql_stmt_bind_param(stmt, ps_params);
if(test_stmt_error(stmt, status) == 1){
    printf("\nPremi invio per continuare...\n");
    while(getchar() != '\n'){}
    goto top;
}

// Run the stored procedure
status = mysql_stmt_execute(stmt);
if(test_stmt_error(stmt, status) == 1){
    printf("\nPremi invio per continuare...\n");
    while(getchar() != '\n'){}
    goto top;
}

return;
break;

default:
    goto top;
break;

}

}

//AGGIORNAMENTO CATEGORIA
void op_a3(){
    MYSQL_STMT *stmt;
    int status;

```





```

        goto top;
    }

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

    // `visualizza_aste_per_nome_oggetto` (in nomeoggetto
varchar(20))

    ps_params[0].buffer_type = MYSQL_TYPE_VAR_STRING;
    ps_params[0].buffer = &nome;
    ps_params[0].buffer_length = 64;
    ps_params[0].length = &length[0];
    ps_params[0].is_null = 0;

    ps_params[1].buffer_type = MYSQL_TYPE_VAR_STRING;
    ps_params[1].buffer = &nuovo_nome;
    ps_params[1].buffer_length = 64;
    ps_params[1].length = &length[1];
    ps_params[1].is_null = 0;

    // bind input parameters
    status = mysql_stmt_bind_param(stmt, ps_params);
    if(test_stmt_error(stmt, status) == 1){
        printf("\nPremi invio per continuare...\n");
        while(getchar() != '\n'){}
        goto top;
    }
    // Run the stored procedure
    status = mysql_stmt_execute(stmt);
    if(test_stmt_error(stmt, status) == 1){
        printf("\nPremi invio per continuare...\n");
        while(getchar() != '\n'){}
        goto top;
    }
    return;
    break;

case 2:
    //define input variable
    printf("\nInserire il nome della categoria : ");
    getInput(64, nome, false);
    length[0] = strlen(nome);

    printf("\nInserire il nome della categoria che si vuole rendere
padre : ");

    getInput(64, nomePadre, false);
    length[1] = strlen(nomePadre);

    status = mysql_stmt_prepare(stmt, "CALL
aggiornamento_categoria(?, null, null, ?)", strlen("CALL aggiornamento_categoria(?, null,
null, ?)"));

    if(test_stmt_error(stmt, status) == 1){
        printf("\nPremi invio per continuare...\n");
        while(getchar() != '\n'){}
        goto top;
    }

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

    // `visualizza_aste_per_nome_oggetto` (in nomeoggetto
varchar(20))

    ps_params[0].buffer_type = MYSQL_TYPE_VAR_STRING;
    ps_params[0].buffer = &nome;
    ps_params[0].buffer_length = 64;
    ps_params[0].length = &length[0];
    ps_params[0].is_null = 0;

```

```

ps_params[1].buffer_type = MYSQL_TYPE_VAR_STRING;
ps_params[1].buffer = &nomePadre;
ps_params[1].buffer_length = 64;
ps_params[1].length = &length[1];
ps_params[1].is_null = 0;

// bind input parameters
status = mysql_stmt_bind_param(stmt, ps_params);
if(test_stmt_error(stmt, status) == 1){
    printf("\nPremi invio per continuare...\n");
    while(getchar() != '\n'){}
    goto top;
}
// Run the stored procedure
status = mysql_stmt_execute(stmt);
if(test_stmt_error(stmt, status) == 1){
    printf("\nPremi invio per continuare...\n");
    while(getchar() != '\n'){}
    goto top;
}
return;
break;

```

case 3:

```

//define input variable
printf("\nInserire il nome della categoria : ");
getInput(64, nome, false);
length[0] = strlen(nome);

printf("\nInserire il nome della attuale categoria padre : ");
getInput(64, nomePadre, false);
length[1] = strlen(nomePadre);

printf("\nInserire il nome della categoria che si vuole rendere
padre : ");

getInput(64, new_nomePadre, false);
length[2] = strlen(new_nomePadre);

status = mysql_stmt_prepare(stmt, "CALL
aggiornamento_categoria(?, null, ?, ?)", strlen("CALL aggiornamento_categoria(?, null, ?,
?)"));

if(test_stmt_error(stmt, status) == 1){
    printf("\nPremi invio per continuare...\n");
    while(getchar() != '\n'){}
    goto top;
}

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

// `visualizza_aste_per_nome_oggetto` (in nomeoggetto
varchar(20))

ps_params[0].buffer_type = MYSQL_TYPE_VAR_STRING;
ps_params[0].buffer = &nome;
ps_params[0].buffer_length = 64;
ps_params[0].length = &length[0];
ps_params[0].is_null = 0;

ps_params[1].buffer_type = MYSQL_TYPE_VAR_STRING;
ps_params[1].buffer = &nomePadre;
ps_params[1].buffer_length = 64;
ps_params[1].length = &length[1];

```

```

        ps_params[1].is_null = 0;

        ps_params[2].buffer_type = MYSQL_TYPE_VAR_STRING;
        ps_params[2].buffer = &new_nomePadre;
        ps_params[2].buffer_length = 64;
        ps_params[2].length = &length[2];
        ps_params[2].is_null = 0;

        // bind input parameters
        status = mysql_stmt_bind_param(stmt, ps_params);
        if(test_stmt_error(stmt, status) == 1){
            printf("\nPremi invio per continuare...\n");
            while(getchar() != '\n'){}
            goto top;
        }
        // Run the stored procedure
        status = mysql_stmt_execute(stmt);
        if(test_stmt_error(stmt, status) == 1){
            printf("\nPremi invio per continuare...\n");
            while(getchar() != '\n'){}
            goto top;
        }
        return;
        break;
    }
}

//CANCELLAZIONE CATEGORIA ****
void op_a4(){
    MYSQL_STMT *stmt;
    int status;
    MYSQL_BIND ps_params[1]; // input parameter buffers
    unsigned long length[1]; // Can do like that because all IN parameters have the
    same length

    //define input variable
    char nome[64];
    top:
    printf("\nInserire il nome della categoria da eliminare: ");
    getInput(64, nome, false);
    length[0] = strlen(nome);

    //inizializzazione statement procedurale.
    stmt = mysql_stmt_init(con);
    if (!stmt) {
        printf("Could not initialize statement\n");
        exit(1);
    }

    status = mysql_stmt_prepare(stmt, "CALL cancellazione_categoria(?)", strlen("CALL
    cancellazione_categoria(?"));
    if(test_stmt_error(stmt, status) == 1){
        printf("\nPremi invio per continuare...\n");
        while(getchar() != '\n'){}
        goto top;
    }

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

    // `visualizza_aste_per_nome_oggetto` (in nomeoggetto varchar(20))
    ps_params[0].buffer_type = MYSQL_TYPE_VAR_STRING;

```

```

ps_params[0].buffer = &nome;
ps_params[0].buffer_length = 64;
ps_params[0].length = &length[0];
ps_params[0].is_null = 0;

// bind input parameters
status = mysql_stmt_bind_param(stmt, ps_params);
if(test_stmt_error(stmt, status) == 1){
    printf("\nPremi invio per continuare...\n");
    while(getchar() != '\n'){}
    goto top;
}

// Run the stored procedure
status = mysql_stmt_execute(stmt);
if(test_stmt_error(stmt, status) == 1){
    printf("\nPremi invio per continuare...\n");
    while(getchar() != '\n'){}
    goto top;
}
}

/*                                     *** Gestione aste ***
*/

//INIZIALIZZAZIONE NUOVA ASTA
void op_a5(){

    MYSQL_STMT *stmt;
    int status;
    MYSQL_BIND ps_params[9];          // input parameter buffers
    unsigned long length[9];          // Can do like that because all IN parameters have the
    same length

    top:
    //titolo : operazione
    printf("\e[1;1H\e[2J");
    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);
    length[0] = strlen(nome);

    char* cat = malloc(sizeof(char)*64);
    printf("\n\ninserire la categoria di afferenza dell'oggetto: ");
    getInput(64, cat, false);
    length[1] = strlen(cat);

    char* desc = malloc(sizeof(char)*64);
    printf("\n\ninserire una breve descrizione dell' oggetto: ");
    getInput(64, desc, false);
    length[2] = strlen(desc);

    char* stato = malloc(sizeof(char)*64);
    printf("\n\ninserire lo stato attuale dell'oggetto (nuovo/usato/buone
condizioni/etc...): ");
    getInput(64, stato, false);
    length[3] = strlen(stato);

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

```

```

getInput(64, colore, false);
length[4] = strlen(colore);

length[5] = 16;

char* dims = malloc(sizeof(char)*64);
printf("\ninserire le dimensioni dell'oggetto: ");
getInput(64, dims, false);
length[6] = strlen(dims);

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);
length[7] = sizeof(float);
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);
length[8] = sizeof(int);
free(_durata);

//inizializzazione statement procedurale.
stmt = mysql_stmt_init(con);
if (!stmt) {
    printf("Could not initialize statement\n");
    exit(1);
}

status = mysql_stmt_prepare(stmt, "CALL inizializzazione_asta(?, ?, ?, ?, ?, ?,
?, ?, ?)", strlen("CALL inizializzazione_asta(?, ?, ?, ?, ?, ?, ?, ?, ?)"));
if(test_stmt_error(stmt, status) == 1){
    printf("\nPremi invio per continuare...\n");
    while(getchar() != '\n'){}
    free(nome);
    free(cat);
    free(desc);
    free(stato);
    free(colore);
    free(dims);
    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 = 64;
ps_params[0].length = &length[0];
ps_params[0].is_null = 0;

ps_params[1].buffer_type = MYSQL_TYPE_STRING;
ps_params[1].buffer = cat;
ps_params[1].buffer_length = 64;
ps_params[1].length = &length[1];
ps_params[1].is_null = 0;

```

```
ps_params[2].buffer_type = MYSQL_TYPE_STRING;
ps_params[2].buffer = desc;
ps_params[2].buffer_length = 64;
ps_params[2].length = &length[2];
ps_params[2].is_null = 0;

ps_params[3].buffer_type = MYSQL_TYPE_STRING;
ps_params[3].buffer = stato;
ps_params[3].buffer_length = 64;
ps_params[3].length = &length[3];
ps_params[3].is_null = 0;

ps_params[4].buffer_type = MYSQL_TYPE_STRING;
ps_params[4].buffer = colore;
ps_params[4].buffer_length = 64;
ps_params[4].length = &length[4];
ps_params[4].is_null = 0;

ps_params[5].buffer_type = MYSQL_TYPE_STRING;
ps_params[5].buffer = logged_user.cf;
ps_params[5].buffer_length = 16;
ps_params[5].length = &length[5];
ps_params[5].is_null = 0;

ps_params[6].buffer_type = MYSQL_TYPE_STRING;
ps_params[6].buffer = dims;
ps_params[6].buffer_length = 64;
ps_params[6].length = &length[6];
ps_params[6].is_null = 0;

ps_params[7].buffer_type = MYSQL_TYPE_FLOAT;
ps_params[7].buffer = prezzobase;
ps_params[7].buffer_length = sizeof(float);
ps_params[7].length = &length[7];
ps_params[7].is_null = 0;

ps_params[8].buffer_type = MYSQL_TYPE_LONG;
ps_params[8].buffer = durata;
ps_params[8].buffer_length = sizeof(int);
ps_params[8].length = &length[8];
ps_params[8].is_null = 0;

// bind input parameters
status = mysql_stmt_bind_param(stmt, ps_params);
if(test_stmt_error(stmt, status) == 1){
    printf("\nPremi invio per continuare...\n");
    while(getchar() != '\n') {}
    free(nome);
    free(cat);
    free(desc);
    free(stato);
    free(colore);
    free(dims);
    free(prezzobase);
    free(durata);
    goto top;
}

// Run the stored procedure
status = mysql_stmt_execute(stmt);
if(test_stmt_error(stmt, status) == 1){
    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);
        return;
    }

    printf("L'Asta è stata correttamente inizializzata.\n");

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

//REPORT ASTA
void op_a6(){

    MYSQL_STMT *stmt;
    int status;
    MYSQL_BIND ps_params[1];          // input parameter buffers
    unsigned long length[1];          // Can do like that because all IN parameters have the
    same length

    top:
    //titolo : operazione
    printf("\e[1;1H\e[2J");
    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);
    length[0] = sizeof(int);

    //inizializzazione statement procedurale.
    stmt = mysql_stmt_init(con);
    if (!stmt) {
        printf("Could not initialize statement\n");
        exit(1);
    }

    status = mysql_stmt_prepare(stmt, "CALL report_asta(?)", strlen("CALL
report_asta(?)));
    if(test_stmt_error(stmt, status) == 1){
        printf("\nPremi invio per continuare...\n");
        while(getchar() != '\n'){}
        free(id);
        goto top;
    }

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

    // `visualizza_stato_asta` (in id int)
    ps_params[0].buffer_type = MYSQL_TYPE_LONG;
    ps_params[0].buffer = id;
    ps_params[0].buffer_length = sizeof(int);
    ps_params[0].length = &length[0];

```



```

ps_params[0].is_null = 0;

// bind input parameters
status = mysql_stmt_bind_param(stmt, ps_params);
if(test_stmt_error(stmt, status) == 1){
    printf("\nPremi invio per continuare...\n");
    while(getchar() != '\n'){}
    free(id);
    goto top;
}
// Run the stored procedure
status = mysql_stmt_execute(stmt);
if(test_stmt_error(stmt, status) == 1){
    printf("\nPremi invio per continuare...\n");
    while(getchar() != '\n'){}
    free(id);
    return;
}

procedure_output( stmt, status);
    free(id);
}

//VISUALIZZAZIONE ASTE INDETTE
void op_a7(){

    MYSQL_STMT *stmt;
    int status;
    MYSQL_BIND ps_params[1];      // input parameter buffers
    unsigned long length[1];      // Can do like that because all IN parameters have the
    same length

    top:
    //titolo : operazione
    printf("\e[1;1H\e[2J");
    printf("\n\n**** VISUALIZZAZIONE ASTE INDETTE ****\n\n");

    //define input variable
    length[0] = 16;

    //inizializzazione statement procedurale.
    stmt = mysql_stmt_init(con);
    if (!stmt) {
        printf("Could not initialize statement\n");
        exit(1);
    }

    status = mysql_stmt_prepare(stmt, "CALL visualizza_aste_per_espositore(?)",
    strlen("CALL visualizza_aste_per_espositore(?)));
    if(test_stmt_error(stmt, status) == 1){
        printf("\nPremi invio per continuare...\n");
        while(getchar() != '\n'){}
        goto top;
    }

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

    // `visualizza_stato_asta` (in id int)
    ps_params[0].buffer_type = MYSQL_TYPE_STRING;
    ps_params[0].buffer = logged_user.cf;
    ps_params[0].buffer_length = 17;

```

```

ps_params[0].length = &length[0];
ps_params[0].is_null = 0;

// bind input parameters
status = mysql_stmt_bind_param(stmt, ps_params);
if(test_stmt_error(stmt, status) == 1){
    printf("\nPremi invio per continuare...\n");
    while(getchar() != '\n'){}
    goto top;
}

// Run the stored procedure
status = mysql_stmt_execute(stmt);
if(test_stmt_error(stmt, status) == 1){
    printf("\nPremi invio per continuare...\n");
    while(getchar() != '\n'){}
    return;
}
procedure_output( stmt, status);
}

//INTERFACCIA UTENTE: UTENTE AMMINISTRATORE
void menu_utente_admin(){

    char *operation;
    start: operation = malloc(64*sizeof(char));
    int op_code;

    printf("\e[1;1H\e[2J");

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

    printf("_____ Gestione delle Categorie _____\n");
    printf("| \n");
    printf("| \n");
    printf("OUT: 0] | \n");
    printf("| \n");
    printf("| OPERAZIONE 1 : Visualizzazione titolario completo\n");
    printf("| \n");
    printf("| OPERAZIONE 2 : Inserimento categoria\n");
    printf("| \n");
    printf("| OPERAZIONE 3 : Aggiornamento categoria\n");
    printf("| \n");
    printf("| OPERAZIONE 4 : Cancellazione categoria\n");
    printf("| \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("|_____
|_____|\n\n");
        printf("    Inserisci il codice dell'operazione : ");
        getInput(64, operation, false);
        op_code = atoi(operation);
        switch((int)op_code){
            case 0:
                break;

        case 1:
            op_a1();
            printf("\n\n Premi invio per tornare al Menù Amministratore...\n");
            while(getchar() != '\n'){}
            free(operation);
                printf("\e[1;1H\e[2J");
            goto start;
            break;

            case 2:
                op_a2();
                printf("\n\n Premi invio per tornare al Menù Amministratore...\n");
                while(getchar() != '\n'){}
                free(operation);
                    printf("\e[1;1H\e[2J");
                goto start;
                break;

                case 3:
                    op_a3();
                    printf("\n\n Premi invio per tornare al Menù Amministratore...\n");
                    while(getchar() != '\n'){}
                    free(operation);
                        printf("\e[1;1H\e[2J");
                    goto start;
                    break;

                    case 4:
                        op_a4();
                        printf("\n\n Premi invio per tornare al Menù Amministratore...\n");
                        while(getchar() != '\n'){}
                        free(operation);
                            printf("\e[1;1H\e[2J");
                        goto start;
                        break;

                            case 5:
                                op_a5();
                                printf("\n\n Premi invio per tornare al Menù Amministratore...\n");
                                while(getchar() != '\n'){}
                                free(operation);
                                    printf("\e[1;1H\e[2J");
                                goto start;
                                break;

                                    case 6:
                                        op_a6();
                                        printf("\n\n Premi invio per tornare al Menù Amministratore...\n");
                                        while(getchar() != '\n'){}

```

```

        free(operation);
        printf("\e[1;1H\e[2J");
        goto start;
        break;

        case 7:
        op_a7();
        printf("\n\n Premi invio per tornare al Menù Amministratore...\n");
        while(getchar() != '\n') {}
        free(operation);
        printf("\e[1;1H\e[2J");
        goto start;
        break;

        case 8:
        op_3();
        printf("\n\n Premi invio per tornare al Menù Amministratore...\n");
        while(getchar() != '\n') {}
        free(operation);
        printf("\e[1;1H\e[2J");
        goto start;
        break;

    default:
        free(operation);
        printf("\e[1;1H\e[2J");
        goto start;
        break;
}

return;
}

```

#### IV. inout.c

```

#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <termios.h>
#include <sys/ioctl.h>
#include <pthread.h>
#include <signal.h>
#include <stdbool.h>

// Per la gestione dei segnali
static volatile sig_atomic_t signo;
typedef struct sigaction sigaction_t;
static void handler(int s);

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

    // Dichiarare le variabili necessarie ad un possibile mascheramento dell'input
    sigaction_t sa, savealrm, saveint, savehup, savequit, saveterm;
    sigaction_t 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, &savealarm);
(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;
int i;
for(i = 0; i < lung; i++) {
    (void) fread(&c, sizeof(char), 1, stdin);
    if(c == '\n') {
        stringa[i] = '\0';
        break;
    } else
        stringa[i] = c;

    // Gestisce gli asterischi
    if(hide) {
        if(c == '\b') // Backspace
            (void) write(fileno(stdout), &c, sizeof(char));
        else
            (void) write(fileno(stdout), "*", sizeof(char));
    }
}

// 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, &savealarm, 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 stringa;
}

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

bool yesOrNo(char *domanda, char yes, char no, bool predef, bool insensitive) {

    // I caratteri 'yes' e 'no' devono essere minuscoli
    yes = tolower(yes);
    no = tolower(no);

    // Decide quale delle due lettere mostrare come predefinite
    char s, n;
    if(predef) {
        s = toupper(yes);
        n = no;
    } else {
        s = yes;
        n = toupper(no);
    }

    // Richiesta della risposta
    while(true) {
        // Mostra la domanda
        printf("%s [%c/%c]: ", domanda, s, n);

        char c;
        getInput(1, &c, false);

        // Controlla quale risposta è stata data
        if(c == '\0') { // getInput() non può restituire '\n!'
            return predef;
        } else if(c == yes) {
            return true;
        } else if(c == no) {
            return false;
        } else if(c == toupper(yes)) {
            if(predef || insensitive) return true;
        } else if(c == toupper(no)) {
            if(!predef || insensitive) return false;
        }
    }
}

```

## V. parse.c

```

#include <stdio.h>

#include <stdlib.h>

```

```

#include <string.h>

#include "jsmn.h"
#include "program.h"

char *config;

static int jsoneq(const char *json, jsmntok_t *tok, const char *s)
{
    if (tok->type == JSMN_STRING
        && (int) strlen(s) == tok->end - tok->start
        && strncmp(json + tok->start, s, tok->end - tok->start) == 0) {
        return 0;
    }
    return -1;
}

int parse_config(void)
{
    int i;
    int r;
    jsmn_parser p;
    jsmntok_t t[128]; /* We expect no more than 128 tokens */

    jsmn_init(&p);
    r = jsmn_parse(&p, config, strlen(config), t, sizeof(t)/sizeof(t[0]));
    if (r < 0) {
        printf("Failed to parse JSON: %d\n", r);
        return 1;
    }

    /* Assume the top-level element is an object */
    if (r < 1 || t[0].type != JSMN_OBJECT) {
        printf("Object expected\n");
        return 1;
    }

    /* Loop over all keys of the root object */
    for (i = 1; i < r; i++) {
        if (jsoneq(config, &t[i], "host") == 0) {
            /* We may use strdup() to fetch string value */
            conf.host = strdup(config + t[i+1].start, t[i+1].end-
t[i+1].start);
            i++;
        } else if (jsoneq(config, &t[i], "username") == 0) {
            conf.username = strdup(config + t[i+1].start, t[i+1].end-
t[i+1].start);
            i++;
        } else if (jsoneq(config, &t[i], "password") == 0) {
            conf.password = strdup(config + t[i+1].start, t[i+1].end-
t[i+1].start);
            i++;
        } else if (jsoneq(config, &t[i], "port") == 0) {
            conf.port = strtol(config + t[i+1].start, NULL, 10);
            i++;
        } else if (jsoneq(config, &t[i], "database") == 0) {
            conf.database = strdup(config + t[i+1].start, t[i+1].end-
t[i+1].start);
            i++;
        } else {
            printf("Unexpected key: %.*s\n", t[i].end-t[i].start, config +
t[i].start);
        }
    }
    return EXIT_SUCCESS;
}

```

```

size_t load_file(char **buffer, char *filename)
{
    FILE *f = fopen(filename, "rb");
    if(f == NULL) {
        fprintf(stderr, "Unable to open file %s\n", filename);
        exit(1);
    }

    fseek(f, 0, SEEK_END);
    size_t fsize = ftell(f);
    fseek(f, 0, SEEK_SET); //same as rewind(f);

    *buffer = malloc(fsize + 1);
    fread(*buffer, fsize, 1, f);
    fclose(f);

    (*buffer)[fsize] = 0;
    return fsize;
}

void dump_config(void)
{
    puts("*** Current Configuration ***");
    printf("Host: %s\n", conf.host);
    printf("Username: %s\n", conf.username);
    printf("Password: %s\n", conf.password);
    printf("Port: %u\n", conf.port);
    printf("database: %s\n", conf.database);
}

```

## VI. config.json

```

{
    "host": "localhost",
    "username": "root",
    "password": "totigimmi",
    "port": 12345,
    "database": "sistemaaste"
}

```

## VII. program.h

```

#pragma once

#include <stdbool.h>

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

struct configuration {
    char *host;
    char *username;
    char *password;
    unsigned int port;
    char *database;
};

extern struct configuration conf;

```



```
extern char *config;

extern int parse_config();
extern size_t load_file(char **buffer, char *filename);
extern void dump_config(void);
extern char *getInput(unsigned int lung, char *stringa, bool hide);
extern bool yesOrNo(char *domanda, char yes, char no, bool predef, bool insensitive);
```

## VIII. jsmn.c

```
#include "jsmn.h"

/**
 * Allocates a fresh unused token from the token pool.
 */
static jsmntok_t *jsmn_alloc_token(jsmn_parser *parser,
                                   jsmntok_t *tokens, size_t num_tokens) {
    jsmntok_t *tok;
    if (parser->toknext >= num_tokens) {
        return NULL;
    }
    tok = &tokens[parser->toknext++];
    tok->start = tok->end = -1;
    tok->size = 0;
#ifdef JSMN_PARENT_LINKS
    tok->parent = -1;
#endif
    return tok;
}

/**
 * Fills token type and boundaries.
 */
static void jsmn_fill_token(jsmntok_t *token, jsmntype_t type,
                           int start, int end) {
    token->type = type;
    token->start = start;
    token->end = end;
    token->size = 0;
}

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

    start = parser->pos;

    for (; parser->pos < len && js[parser->pos] != '\\0'; parser->pos++) {
        switch (js[parser->pos]) {
#ifdef JSMN_STRICT
            /* In strict mode primitive must be followed by "," or "}" or
            "]" */
            case ':':
            case '\t':
            case '\r':
            case '\n':
            case ' ':
            case ',':
            case ']':
            case '}':
                goto found;
            default:
                /* ok */
                break;
#else
            /* not strict mode */
            case ':':
            case '\t':
            case '\r':
            case '\n':
            case ' ':
            case ',':
            case ']':
            case '}':
                goto found;
            default:
                /* ok */
                break;
#endif
        }
    }
    found:
    token = jsmn_alloc_token(parser, tokens, num_tokens);
    token->start = start;
    token->end = parser->pos;
    token->type = JSMN_PRIMITIVE;
    token->parent = -1;
    return 1;
}
```

```

        if (js[parser->pos] < 32 || js[parser->pos] >= 127) {
            parser->pos = start;
            return JSMN_ERROR_INVALID;
        }
    }
#ifdef JSMN_STRICT
    /* In strict mode primitive must be followed by a comma/object/array */
    parser->pos = start;
    return JSMN_ERROR_PART;
#endif

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

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

    int start = parser->pos;

    parser->pos++;

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

        /* Quote: end of string */
        if (c == '\"') {
            if (tokens == NULL) {
                return 0;
            }
            token = jsmn_alloc_token(parser, tokens, num_tokens);
            if (token == NULL) {
                parser->pos = start;
                return JSMN_ERROR_NOMEM;
            }
            jsmn_fill_token(token, JSMN_STRING, start+1, parser->pos);
#ifdef JSMN_PARENT_LINKS
            token->parent = parser->toksuper;
#endif
            return 0;
        }

        /* Backslash: Quoted symbol expected */
        if (c == '\\') && parser->pos + 1 < len) {
            int i;
            parser->pos++;
            switch (js[parser->pos]) {
                /* Allowed escaped symbols */
                case '\\': case '/': case '\"': case 'b' :

```

```

        case 'f' : case 'r' : case 'n' : case 't' :
            break;
        /* Allows escaped symbol \uXXXX */
        case 'u':
            parser->pos++;
            for(i = 0; i < 4 && parser->pos < len &&
js[parser->pos] != '\0'; i++) {
                /* If it isn't a hex character we have
an error */
                if(!((js[parser->pos] >= 48 &&
js[parser->pos] <= 57) || /* 0-9 */
                    (js[parser->pos] >= 65 && js[parser->pos] <= 70) || /* A-F */
                    (js[parser->pos] >= 97 && js[parser->pos] <= 102)))) { /* a-f */
                    parser->pos = start;
                    return JSMN_ERROR_INVALID;
                }
                parser->pos++;
            }
            parser->pos--;
            break;
        /* Unexpected symbol */
        default:
            parser->pos = start;
            return JSMN_ERROR_INVALID;
    }
}
parser->pos = start;
return JSMN_ERROR_PART;
}

/**
 * Parse JSON string and fill tokens.
 */
int jsmn_parse(jsmn_parser *parser, const char *js, size_t len,
               jsmntok_t *tokens, unsigned int num_tokens) {
    int r;
    int i;
    jsmntok_t *token;
    int count = parser->toknext;

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

        c = js[parser->pos];
        switch (c) {
            case '{': case '[':
                count++;
                if (tokens == NULL) {
                    break;
                }
                token = jsmn_alloc_token(parser, tokens, num_tokens);
                if (token == NULL)
                    return JSMN_ERROR_NOMEM;
                if (parser->toksuper != -1) {
                    tokens[parser->toksuper].size++;
#ifdef JSMN_PARENT_LINKS
                    token->parent = parser->toksuper;
#endif
                }
                token->type = (c == '{' ? JSMN_OBJECT : JSMN_ARRAY);
                token->start = parser->pos;
                parser->toksuper = parser->toknext - 1;
                break;
            case '}': case ']':

```

```

if (tokens == NULL)
    break;
type = (c == '}' ? JSMN_OBJECT : JSMN_ARRAY);

#ifdef JSMN_PARENT_LINKS
if (parser->toknext < 1) {
    return JSMN_ERROR_INVALID;
}
token = &tokens[parser->toknext - 1];
for (;;) {
    if (token->start != -1 && token->end == -1) {
        if (token->type != type) {
            return JSMN_ERROR_INVALID;
        }
        token->end = parser->pos + 1;
        parser->toksuper = token->parent;
        break;
    }
    if (token->parent == -1) {
        if (token->type != type || parser->
>toksuper == -1) {
            return JSMN_ERROR_INVALID;
        }
        break;
    }
    token = &tokens[token->parent];
}

#else

for (i = parser->toknext - 1; i >= 0; i--) {
    token = &tokens[i];
    if (token->start != -1 && token->end == -1) {
        if (token->type != type) {
            return JSMN_ERROR_INVALID;
        }
        parser->toksuper = -1;
        token->end = parser->pos + 1;
        break;
    }
}

/* Error if unmatched closing bracket */
if (i == -1) return JSMN_ERROR_INVALID;
for (; i >= 0; i--) {
    token = &tokens[i];
    if (token->start != -1 && token->end == -1) {
        parser->toksuper = i;
        break;
    }
}

#endif

break;
case '\"':
    r = jsmn_parse_string(parser, js, len, tokens,
num_tokens);
    if (r < 0) return r;
    count++;
    if (parser->toksuper != -1 && tokens != NULL)
        tokens[parser->toksuper].size++;
    break;
case '\t' : case '\r' : case '\n' : case ' ':
    break;
case ':':
    parser->toksuper = parser->toknext - 1;
    break;
case ',':
    if (tokens != NULL && parser->toksuper != -1 &&
        tokens[parser->toksuper].type !=
JSMN_ARRAY &&
        tokens[parser->toksuper].type !=
JSMN_OBJECT) {

```

```

#ifdef JSMN_PARENT_LINKS
    parser->toksuper = tokens[parser->toknext - 1];
#else
    for (i = parser->toknext - 1; i >= 0; i--) {
        if (tokens[i].type == JSMN_ARRAY ||
            tokens[i].type == JSMN_OBJECT) {
            if (tokens[i].start != -1 &&
                tokens[i].end == -1) {
                parser->toksuper = i;
                break;
            }
        }
    }
#endif

#ifdef JSMN_STRICT
    /* In strict mode primitives are: numbers and booleans */
    case '-': case '0': case '1': case '2': case '3': case '4':
    case '5': case '6': case '7': case '8': case '9':
    case 't': case 'f': case 'n' :
        /* And they must not be keys of the object */
        if (tokens != NULL && parser->toksuper != -1) {
            jsmntok_t *t = &tokens[parser->toksuper];
            if (t->type == JSMN_OBJECT ||
                (t->type == JSMN_STRING && t->size != 0)) {
                return JSMN_ERROR_INVALID;
            }
        }
    }
#else
    /* In non-strict mode every unquoted value is a primitive */
    default:
        r = jsmn_parse_primitive(parser, js, len, tokens,
                                num_tokens);
        if (r < 0) return r;
        count++;
        if (parser->toksuper != -1 && tokens != NULL)
            tokens[parser->toksuper].size++;
        break;
#endif

#ifdef JSMN_STRICT
    /* Unexpected char in strict mode */
    default:
        return JSMN_ERROR_INVALID;
#endif

    }

    if (tokens != NULL) {
        for (i = parser->toknext - 1; i >= 0; i--) {
            /* Unmatched opened object or array */
            if (tokens[i].start != -1 && tokens[i].end == -1) {
                return JSMN_ERROR_PART;
            }
        }
    }

    return count;
}

/**
 * Creates a new parser based over a given buffer with an array of tokens
 * available.
 */
void jsmn_init(jsmn_parser *parser) {

```

```

    parser->pos = 0;
    parser->toknext = 0;
    parser->toksupper = -1;
}

```

## IX. jsmn.h

```

#pragma once

#include <stddef.h>

#ifdef __cplusplus
extern "C" {
#endif

/**
 * JSON type identifier. Basic types are:
 *   o Object
 *   o Array
 *   o String
 *   o Other primitive: number, boolean (true/false) or null
 */
typedef enum {
    JSMN_UNDEFINED = 0,
    JSMN_OBJECT = 1,
    JSMN_ARRAY = 2,
    JSMN_STRING = 3,
    JSMN_PRIMITIVE = 4
} jsmntype_t;

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

/**
 * JSON token description.
 *   type           type (object, array, string etc.)
 *   start          start position in JSON data string
 *   end            end position in JSON data string
 */
typedef struct {
    jsmntype_t type;
    int start;
    int end;
    int size;
#ifdef JSMN_PARENT_LINKS
    int parent;
#endif
} jsmntok_t;

/**
 * JSON parser. Contains an array of token blocks available. Also stores
 * the string being parsed now and current position in that string
 */
typedef struct {
    unsigned int pos; /* offset in the JSON string */
    unsigned int toknext; /* next token to allocate */

```

```
        int toksuper; /* superior token node, e.g parent object or array */
    } jsmn_parser;

/**
 * Create JSON parser over an array of tokens
 */
void jsmn_init(jsmn_parser *parser);

/**
 * Run JSON parser. It parses a JSON data string into an array of tokens, each describing
 * a single JSON object.
 */
int jsmn_parse(jsmn_parser *parser, const char *js, size_t len,
               jsmntok_t *tokens, unsigned int num_tokens);

#ifdef __cplusplus
}
#endif
```