Basi di Dati e Conoscenza

Progetto A.A. 2020/2021

Sistema di gioco Risiko online

0252159

Dominique Toce

Indice

1. Descrizione del Minimondo

Si vuole realizzare un servizio online che consenta di giocare ad un clone del famoso gioco Risiko, in modalità "conquista del mondo".

Al sistema hanno accesso due tipologie di utenti: i giocatori e i moderatori. I moderatori hanno la possibilità di creare stanze di gioco, in funzione della quantità di giocatori che utilizzano attualmente il sistema. In particolare, i moderatori hanno la possibilità di visualizzare, tramite un report, quante stanze hanno attualmente partite in corso e quanti giocatori stanno partecipando alla partita. Inoltre, in questo report, gli amministratori possono visualizzare il numero totale di giocatori che hanno effettuato almeno un'operazione negli ultimi 15 minuti che non sono all'interno di alcuna stanza di gioco.

Una stanza permette ad un numero massimo di sei giocatori di entrare e partecipare alla partita. Una partita coinvolge almeno tre giocatori. Quando il terzo giocatore entra in una stanza, viene attivato un countdown tale da avviare la partita dopo due minuti. In questi due minuti altri giocatori possono entrare, fino al massimo concesso.

All'avvio della partita, gli stati del tabellone vengono assegnati casualmente ai giocatori. I turni "girano" in funzione del tempo di ingresso dei giocatori nella stanza (chi è entrato prima gioca prima). Un turno prevede che il giocatore possa compiere una delle seguenti azioni:

* Posizionare un numero arbitrario di carri armati in uno stato
* Scegliere uno stato da cui fare partire un attacco verso uno stato adiacente
* Spostare carri armati da uno stato ad un altro adiacente (almeno un carro armato deve restare nello stato di partenza)

L'attacco viene svolto nel seguente modo. La fase di attacco si svolge tra il giocatore che attacca e quello che difende attraverso il lancio dei dadi. Il numero dei dadi da lanciare è stabilito dal numero di armate che si decide di schierare in guerra meno una, fino ad un massimo di tre dadi per volta. Ognuno dei giocatori lancia il numero di dadi corrispondenti, e poi si confrontano i valori ottenuti, il più alto dell'attaccante con il più alto del difensore, il

secondo con il secondo e così via. Per ogni punteggio più alto, il perdente deve togliere un'armata dal tabellone. In caso di parità il punto va al difensore. Il lancio dei dadi viene simulato mediante la generazione di numeri pseudocasuali.

Se lo stato attaccato perde tutte le armate, questo viene conquistato e vengono spostate automaticamente in esso un numero di armate pari a quelle sopravvissute all'attacco.

Un apposito timer determina quando il tempo per svolgere un'azione da parte di un giocatore scade e il turno passa quindi al giocatore successivo.

Al termine del turno, se è stata svolta almeno un'azione, il giocatore riceve un numero di carri armati da posizionare pari al numero di stati posseduti diviso tre, arrotondato per eccesso.

Un'apposita procedura consente al client di sapere, quando è il turno del giocatore, tutto lo stato di gioco e quindi far scegliere quale azione effettuare. Si ricorda, comunque, che tutta la logica applicativa è implementata nel DBMS.

Un giocatore può sempre visualizzare lo storico di tutte le partite giocate.

2. Analisi dei Requisiti

Identificazione dei termini ambigui e correzioni possibili

Linea	Termine	Nuovo termine	Motivo correzione
7	amministratori	moderatori	Unifico termini con stesso significato (sinonimi)
8	operazione	azione	Unifico termini con stesso significato (sinonimi)
6, 10, 12, 15	stanza	stanza di gioco	Unifico termini con stesso significato (sinonimi)
18, 20, 35	carro armato	armata	Unifico termini con stesso significato (sinonimi)
26	attaccante	giocatore che attacca	Unifico termini con stesso significato (sinonimi)
26, 28	difensore	giocatore che difende	Unifico termini con stesso significato (sinonimi)
34	almeno	-	Elimino il termine perché crea ambiguità riguardo al fatto che, durante un turno, un giocatore può svolgere una e una sola azione

Specifica disambiguata

Si vuole realizzare un servizio online che consenta di giocare ad un clone del famoso gioco Risiko, in modalità "conquista del mondo".

Al sistema hanno accesso due tipologie di utenti: i giocatori e i moderatori. I moderatori hanno la possibilità di creare stanze di gioco, in funzione della quantità di giocatori che utilizzano attualmente il sistema. In particolare, i moderatori hanno la possibilità di visualizzare, tramite un report, quante stanze di gioco hanno attualmente partite in corso e quanti giocatori stanno partecipando alla partita. Inoltre, in questo report, i moderatori possono visualizzare il numero totale di giocatori che hanno effettuato almeno un'azione negli ultimi 15 minuti che non sono all'interno di alcuna stanza di gioco.

Una stanza di gioco permette ad un numero massimo di sei giocatori di entrare e partecipare alla partita. Una partita coinvolge almeno tre giocatori. Quando il terzo giocatore entra in una stanza di gioco, viene attivato un countdown tale da avviare la partita dopo due minuti. In questi due

minuti altri giocatori possono entrare, fino al massimo concesso.

All'avvio della partita, gli stati del tabellone vengono assegnati casualmente ai giocatori. I turni "girano" in funzione del tempo di ingresso dei giocatori nella stanza di gioco (chi è entrato prima gioca prima). Un turno prevede che il giocatore possa compiere una delle seguenti azioni:

* Posizionare un numero arbitrario di armate in uno stato
* Scegliere uno stato da cui fare partire un attacco verso uno stato adiacente
* Spostare armate da uno stato ad un altro adiacente (almeno un’armata deve restare nello stato di partenza)

L'attacco viene svolto nel seguente modo. La fase di attacco si svolge tra il giocatore che attacca e quello che difende attraverso il lancio dei dadi. Il numero dei dadi da lanciare è stabilito dal numero di armate che si decide di schierare in guerra meno una, fino ad un massimo di tre dadi per volta. Ognuno dei giocatori lancia il numero di dadi corrispondenti, e poi si confrontano i valori ottenuti, il più alto del giocatore che attacca con il più alto del giocatore che difende, il secondo con il secondo e così via. Per ogni punteggio più alto, il perdente deve togliere un'armata dal tabellone. In caso di parità il punto va al giocatore che difende.
Il lancio dei dadi viene simulato mediante la generazione di numeri pseudocasuali.

Se lo stato attaccato perde tutte le armate, questo viene conquistato e vengono spostate automaticamente in esso un numero di armate pari a quelle sopravvissute all'attacco.

Un apposito timer determina quando il tempo per svolgere un'azione da parte di un giocatore scade e il turno passa quindi al giocatore successivo.

Al termine del turno, se è stata svolta un'azione, il giocatore riceve un numero di armate da posizionare pari al numero di stati posseduti diviso tre, arrotondato per eccesso.

Un'apposita procedura consente al client di sapere, quando è il turno del giocatore, tutto lo

stato di gioco e quindi far scegliere quale azione effettuare. Si ricorda, comunque, che tutta la logica applicativa è implementata nel DBMS.

Un giocatore può sempre visualizzare lo storico di tutte le partite giocate.

Glossario dei Termini

Termine	Descrizione	Sinonimi	Collegamenti
Stanza di gioco	Luogo virtuale che ospita una partita	Stanza	Partita, Moderatore
Partita	Competizione alla quale partecipano da 3 a 6 giocatori	-	Stanza di gioco, Giocatore, Turno, Stato
Giocatore	Colui che gioca una partita	-	Partita, Turno, Stato
Turno	Periodo della partita in cui un giocatore può svolgere un'azione (spostamento, attacco, posizionamento)	-	Giocatore, Partita
Moderatore	Colui che gestisce le stanze di gioco	Amministratore	Stanza di gioco
Stato	Rappresentazione di una nazione sul tabellone di gioco; è adiacente ad altri stati	-	Giocatore, Partita

Raggruppamento dei requisiti in insiemi omogenei

Frasi di carattere generale
Si vuole realizzare un servizio online che consenta di giocare ad un clone del famoso gioco Risiko, in modalità "conquista del mondo". Al sistema hanno accesso due tipologie di utenti: i giocatori e i moderatori.
Frasi relative a Stanza di gioco
Una stanza di gioco permette ad un numero massimo di sei giocatori di entrare e partecipare alla partita. Quando il terzo giocatore entra in una stanza di gioco, viene attivato un countdown tale da

avviare la partita dopo due minuti. In questi due minuti altri giocatori possono entrare, fino al massimo concesso.

Frasi relative a Partita

Una partita coinvolge almeno tre giocatori.

All'avvio della partita, gli stati del tabellone vengono assegnati casualmente ai giocatori.

Frasi relative a Giocatore

Un giocatore può compiere una delle seguenti azioni:

* Posizionare un numero arbitrario di armate in uno stato

* Scegliere uno stato da cui fare partire un attacco verso uno stato adiacente

* Spostare armate da uno stato ad un altro adiacente (almeno un’armata deve restare nello stato di partenza).

L'attacco viene svolto nel seguente modo. La fase di attacco si svolge tra il giocatore che attacca e quello che difende attraverso il lancio dei dadi. Il numero dei dadi da lanciare è stabilito dal numero di armate che si decide di schierare in guerra meno una, fino ad un massimo di tre dadi per volta. Ognuno dei giocatori lancia il numero di dadi corrispondenti, e poi si confrontano i valori ottenuti, il più alto del giocatore che attacca con il più alto del giocatore che difende, il secondo con il secondo e così via. Per ogni punteggio più alto, il perdente deve togliere un'armata dal tabellone. In caso di parità il punto va al giocatore che difende.

Al termine del turno, se è stata svolta un'azione, il giocatore riceve un numero di armate da posizionare pari al numero di stati posseduti diviso tre, arrotondato per eccesso.

Un giocatore può sempre visualizzare lo storico di tutte le partite giocate.

Frasi relative a Moderatore

I moderatori hanno la possibilità di creare stanze di gioco, in funzione della quantità di giocatori che utilizzano attualmente il sistema. In particolare, i moderatori hanno la possibilità di visualizzare, tramite un report, quante stanze di gioco hanno attualmente partite in corso e quanti giocatori stanno partecipando alla partita. Inoltre, in questo report, i moderatori possono visualizzare il numero totale di giocatori che hanno effettuato almeno un'azione negli ultimi 15 minuti che non sono all'interno di alcuna stanza di gioco.

Frasi relative a Turno

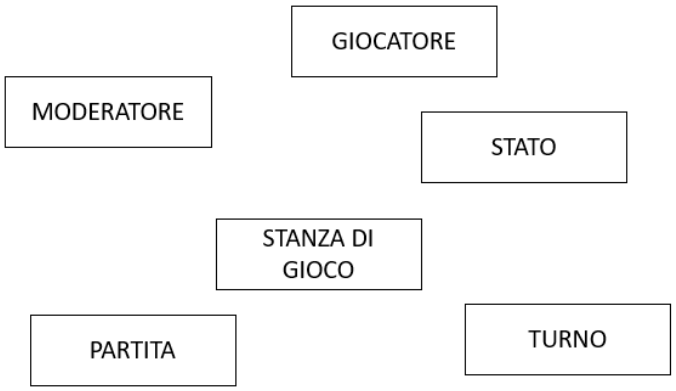
I turni "girano" in funzione del tempo di ingresso dei giocatori nella stanza di gioco (chi è entrato

prima gioca prima). Un turno prevede che il giocatore possa compiere una delle tre azioni possibili. Un apposito timer determina quando il tempo per svolgere un'azione da parte di un giocatore scade e il turno passa quindi al giocatore successivo. Al termine del turno, se è stata svolta un'azione, il giocatore riceve un numero di armate da posizionare pari al numero di stati posseduti diviso tre, arrotondato per eccesso.
Frasi relative a Stato
All'avvio della partita, gli stati del tabellone vengono assegnati casualmente ai giocatori. Se lo stato attaccato perde tutte le armate, questo viene conquistato e vengono spostate automaticamente in esso un numero di armate pari a quelle sopravvissute all'attacco.

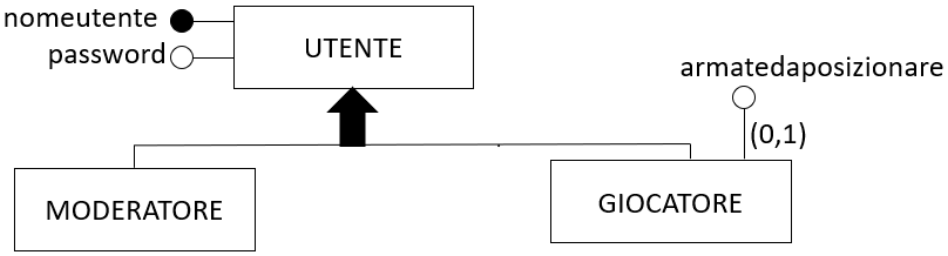
3. Progettazione concettuale

Costruzione dello schema E-R

Sulla base dell'analisi dei requisiti precedentemente effettuata, rappresento come entità i concetti autonomi e maggiormente rilevanti riportati anche nel glossario dei termini:

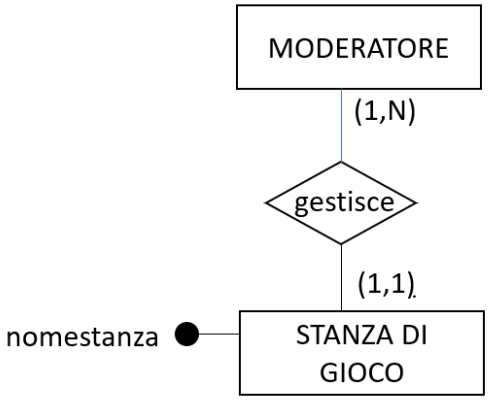


Utilizzando una generalizzazione, distinguo le due tipologie di utente che sono previste nel sistema:

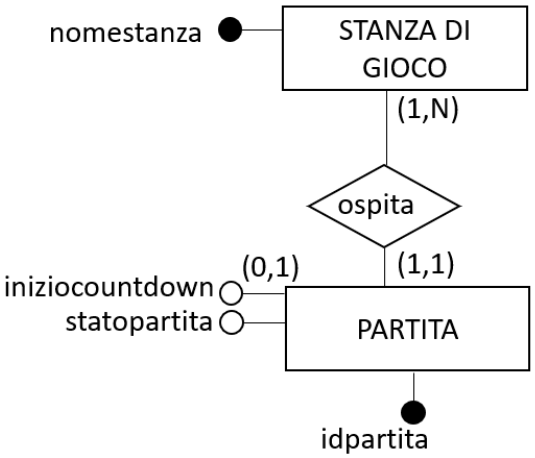


GIOCATORE e MODERATORE sono quindi una specializzazione dell'entità UTENTE sulla quale, seppur non indicato nelle specifiche, aggiungo due attributi: 'nomeutente' (come identificatore) e 'password', sfruttando la proprietà che ogni attributo dell'entità padre viene ereditato dalle entità figlie. In più, sull'entità GIOCATORE aggiungo l'attributo opzionale 'armatedaposizionare' per indicare il numero di armate che un giocatore possiede (finché non gioca la sua prima partita questo attributo non sarà specificato).

A partire da questa prima rappresentazione procedo a "macchia d'olio" (per comodità riporto solo parti nuove del diagramma), inserendo tra le entità MODERATORE e STANZA DI GIOCO una associazione uno a molti che vuole evidenziare la creazione nonché gestione di queste ultime da parte di un moderatore:



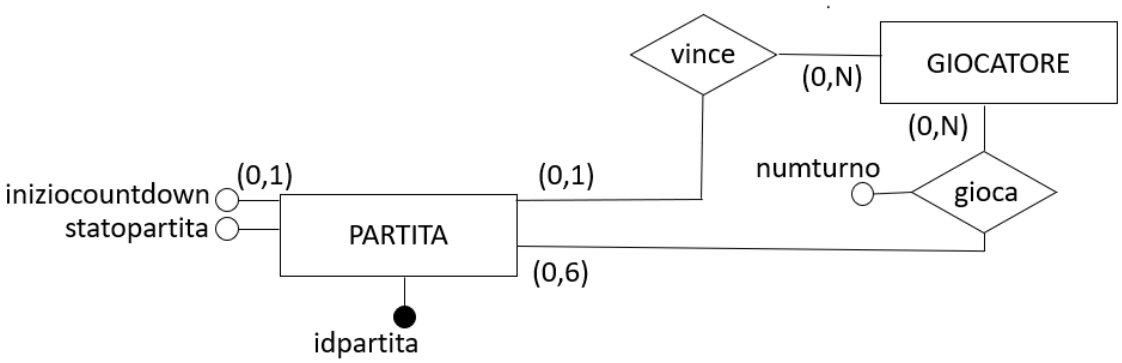
Tra le entità STANZA DI GIOCO e PARTITA inserisco un'altra associazione uno a molti per indicare che ogni partita si svolge in una stanza di gioco; sull'entità PARTITA aggiungo gli attributi: 'idpartita' come identificatore, 'statopartita' per salvare informazioni riguardo al fatto che la partita sia in attesa di iniziare o in corso oppure sia terminata e infine 'iniziocountdown' che è un attributo opzionale che salverà il momento in cui partecipa alla partita un terzo giocatore (finché un terzo giocatore non partecipa alla partita questo attributo non sarà specificato).



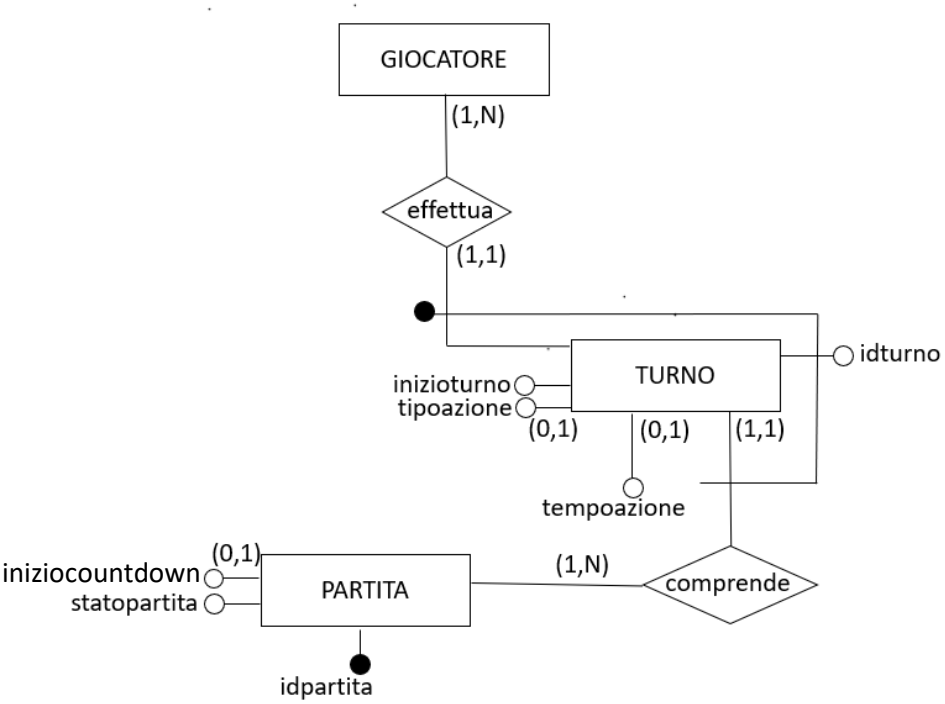
Inoltre, aggiungo due associazioni tra l'entità PARTITA e l'entità GIOCATORE:

- una per via della partecipazione dei giocatori a partite, su cui aggiungo un attributo 'numturno' per mantenere appunto il numero del turno di un certo giocatore all'interno di una certa partita
- un'altra, seppur non richiesto nella specifica, per tenere traccia del giocatore che ha vinto una certa partita e mostrarlo nel momento in cui un giocatore vuole visualizzare lo storico di tutte le partite da lui giocate, in quanto ritengo che sia particolarmente interessato a sapere quali

partite ha vinto e quali no (oltre alle altre informazioni di base come id partita, stanza di gioco, ecc.).



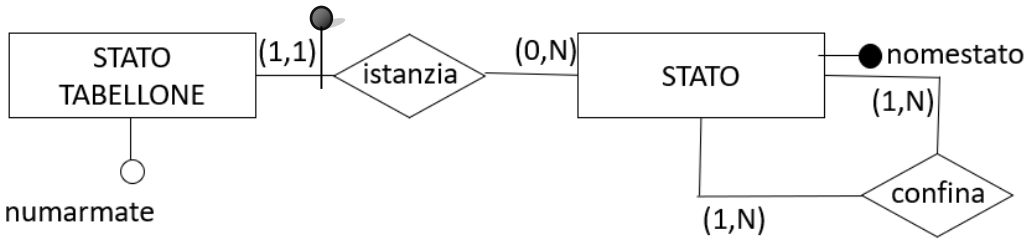
Continuando la "navigazione" tra le specifiche, sposto l'attenzione sull'entità TURNO che è un concetto collegato sia al GIOCATORE che alla PARTITA. Su di essa aggiungo l'attributo 'idturno' che però da solo non basta come identificatore; in particolare visto che un turno è relativo a una certa partita e deve essere giocato da un certo giocatore all'interno della partita, TURNO è un'entità debole sulle entità PARTITA e GIOCATORE. Inoltre, sempre su TURNO aggiungo altri attributi: 'tipoazione' per salvare quale azione viene svolta all'interno di quel turno, 'inizioturno' per tenere traccia del momento in cui un turno inizia e 'tempoazione' per salvare il momento in cui viene effettuata un'azione nel turno.



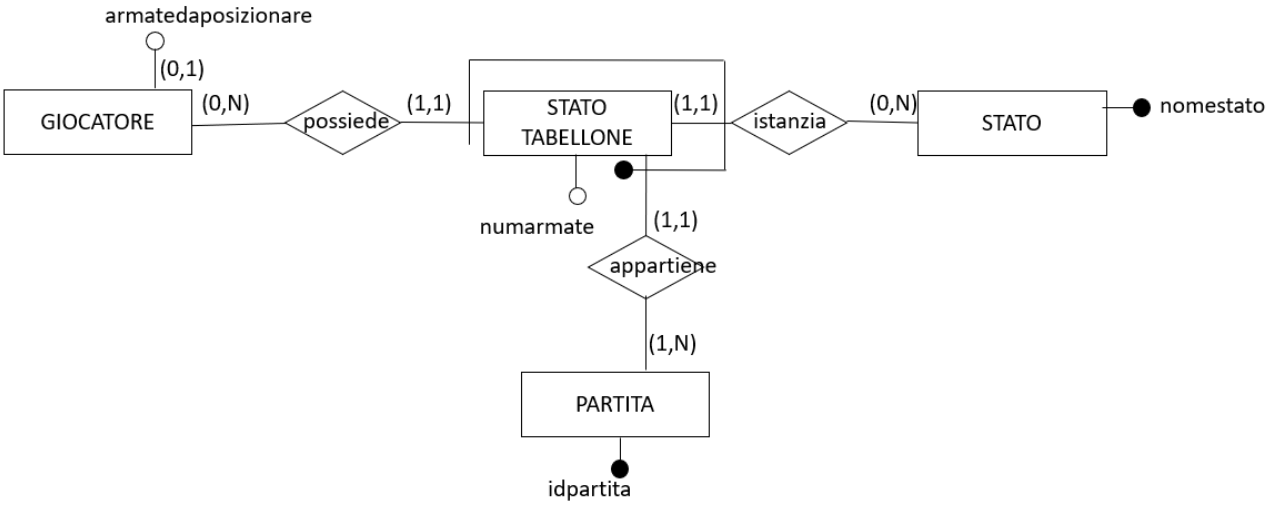
A questo punto resta da occuparsi del concetto di STATO. L'entità STATO, da sola, rischia di confondere due concetti ben distinti per il nostro minimondo di interesse e pertanto distinguo tra:

- STATO → rappresenta una entità astratta ovvero il concetto di Stato, con nome dello Stato (attributo 'nomestato') e i suoi confinanti (che rappresento con una associazione ricorsiva simmetrica, molti a molti, sull'entità STATO stessa)
- STATO TABELLONE → rappresenta, invece, un'istanza di STATO in un certo tabellone di gioco e pertanto posseduta effettivamente da un giocatore durante una partita e con un certo numero di armate disposte su di essa (attributo 'numarmate').

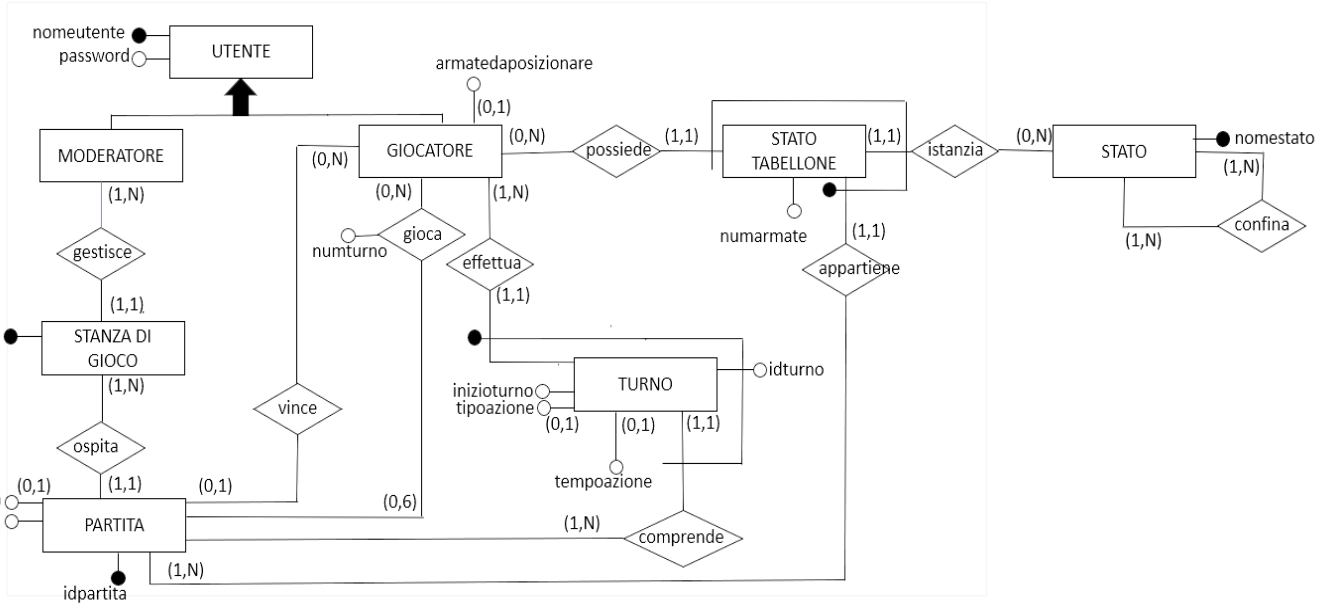
Lego queste due entità con una associazione di tipo "instance of" uno a molti.



Infine, l'entità STATO TABELLONE oltre che dallo STATO è identificata anche dalle entità GIOCATORE e PARTITA (è un'entità debole):



Integrazione finale



Regole aziendali

RV = regola di vincolo

(RV1) Una partita deve coinvolgere almeno 3 giocatori per essere giocata.

(RV2) L'attributo 'statopartita' di PARTITA deve assumere uno dei seguenti valori:

'inattesa'

'incorso'

'terminata'

(RV3) L'attributo opzionale 'tipoazione' di TURNO deve assumere uno dei seguenti valori:

'posizionamento'

'attacco'

'spostamento'

'nonsvolta'

(RV4) L'attributo 'numturno' dell'associazione Gioca deve assumere un valore compreso nell'intervallo [1,6].

(RV5) Durante un'azione di tipo spostamento sullo stato da cui parte l'azione deve rimanere almeno un'armata.

(RV6) Al termine di un turno in cui è stata svolta un'azione, il numero di armate che riceve il giocatore deve essere pari al numero di stati posseduti dal giocatore diviso 3, arrotondato per eccesso.

(RV7) Un giocatore può partecipare a una sola partita per volta.

(RV8) Una stanza di gioco può ospitare una sola partita per volta.

(RV9) Il valore dell'attributo 'tempoazione' di TURNO deve essere maggiore del valore dell'attributo 'inizioturno' di TURNO.

Dizionario dei dati

Entità	Descrizione	Attributi	Identificatori
Utente	Utente che ha accesso al gioco	nomeutente, password	nomeutente
Moderatore	Tipo di utente che crea e gestisce le stanze di gioco	nomeutente, password	nomeutente
Giocatore	Tipo di utente che gioca una partita	nomeutente, password, armatedaposizionare	nomeutente
Stanza di gioco	Stanza che ospita una partita del gioco per volta	nomestanza	nomestanza
Partita	Partita alla quale possono partecipare da 3 a 6 giocatori; può essere in attesa, in corso o terminata	idpartita, statopartita, iniziocountdown	idpartita
Turno	Turno di un giocatore in una partita, durante il quale egli può svolgere un'azione (spostamento, attacco o posizionamento)	idturno, tipoazione, tempoazione, inizioturno	idturno, giocatore, partita
Stato tabellone	Istanza di Stato presente sul tabellone di una certa partita	numarmate	partita, stato, giocatore
Stato	Stato confinante con altri Stati	nomestato	nomestato

Associazione	Descrizione	Entità coinvolte	Attributi
Gestisce	Un moderatore gestisce una o più stanze di gioco	Moderatore, Stanza di gioco	-
Ospita	Una stanza di gioco ospita una o più partite	Stanza di gioco, Partita	-
Vince	Un giocatore può vincere nessuna o più partite	Giocatore, Partita	-
Gioca	Un giocatore può giocare nessuna o più partite	Giocatore, Partita	numturno
Comprende	Una partita comprende uno o più turni	Partita, Turno	-
Effettua	Un giocatore effettua uno o più turni	Giocatore, Turno	-
Possiede	Un giocatore possiede nessuno o più Stati sul tabellone	Giocatore, Stato tabellone	-
Appartiene	Uno Stato del tabellone appartiene a una e una sola partita	Stato tabellone, Partita	-
Istanzia	Uno Stato del tabellone è un'istanza di uno e un solo Stato	Stato tabellone, Stato	-
Confina	Uno Stato confina con uno o più Stati	Stato	-

4. Progettazione logica

Volume dei dati

Concetto nello schema	Tipo[1]	Volume atteso
Utente	E	10.000
Moderatore	E	500
Giocatore	E	9.500
Stanza di gioco	E	1.000 (in media 2 per moderatore)
Partita	E	1.000.000
Turno	E	150.000.000 (in media 150 turni a partita)
Stato tabellone	E	42.000.000 (42 per partita)
Stato	E	42
Gestisce	R	1.000 (= num stanze di gioco)
Ospita	R	1.000.000 (= num partite)
Gioca	R	4.000.000 (in media 4 giocatori per partita: 4.000.000)
Vince	R	990.000
Effettua	R	150.000.000 (= num turni)
Comprende	R	150.000.000 (= num turni)
Possiede	R	42.000.000 (= num stati tabellone)
Confina	R	150 (42 stati ciascuno con in media 3 confinanti)
Appartiene	R	42.000.000 (= num stati tabellone)
Istanzia	R	42.000.000 (= num stati tabellone)

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

Tavola delle operazioni

Cod.	Descrizione	Frequenza attesa
01	Creare una stanza di gioco	4/settimana
02	Aggiungere un nuovo moderatore	2/settimana
03	Aggiungere un nuovo giocatore	6/g
04	Creare una partita	500/g
05	Partecipare ad una partita	2000/g
06	Conoscere il n° di stanze con partite attualmente in corso e il numero di giocatori che stanno partecipando ad ognuna di queste partite. Inoltre, conoscere il numero di giocatori che non sono in nessuna stanza e hanno effettuato un'azione negli ultimi 15 minuti	50/settimana
07	Effettuare un turno in cui si posiziona un certo n° di armate in uno stato	20.000/g (di 70.00)
08	Conoscere lo stato di gioco di una partita	70.000/g (in media 150 turni per partita)
09	Effettuare un turno in cui si realizza un attacco da uno stato ad un altro adiacente	40.000/g (di 70.000)
10	Effettuare un turno in cui si sposta un certo numero di armate da uno stato ad un altro adiacente	50.000/g (40.000 derivanti da un attacco + 10.000 di 70.000)
11	Visualizzare lo storico di tutte le partite giocate da un dato giocatore	100/settimana

Costo delle operazioni

Costo in scrittura = 2*costo in lettura

(01) Una scrittura in STANZA DI GIOCO → (1*2) *4 = 8 accessi/settimana

(02) Una scrittura in MODERATORE → (1*2) *2 = 4 accessi/settimana

(03) Una scrittura in GIOCATORE → (1*2) * 6 = 12 accessi/g

(04) Una scrittura in PARTITA → (1*2) * 500 = 1000 accessi/g

(05) Una scrittura in Gioca → (1*2) *2000 = 4000 accessi/g

(06) 100 letture in PARTITA (supponendo ci siano in totale 100 partite), 60 letture in Ospita e altre 60 letture in Gioca (supponendo che delle 100 partite, 60 siano in corso); inoltre, 160 letture in GIOCATORE (supponendo in media 4 giocatori per le 40 partite non in corso), 6400 letture in Effettua e 6400 letture in TURNO (supponendo che in media un giocatore effettua 40 turni per partita) → (100+60+60+160+6400+6400) * 50 = 659.000 accessi/settimana

(07) Una lettura in GIOCATORE per leggere l'attributo 'armatedaposizionare', una lettura in Possiede, una lettura in STATO TABELLONE, una scrittura in Effettua, una scrittura in TURNO, una scrittura in STATO TABELLONE per modificare l'attributo 'numarmate', una scrittura in GIOCATORE per modificare l'attributo 'armatedaposizionare' → (1+1+1+1*2+1*2+1*2+1*2) *20.000 = 220.000 accessi/giorno

(08) Una lettura in PARTITA, 42 letture in Appartiene, 42 letture in STATO TABELLONE → (1+42+42) *70.000 = 5.950.000 accessi/g

(09) Due letture in Possiede, due letture in STATO TABELLONE, una scrittura in Effettua, una scrittura in TURNO, una scrittura in STATO TABELLONE per modificare l'attributo 'numarmate' dello stato perdente, una scrittura in Possiede (solo se lo stato attaccato viene conquistato; supponiamo che ciò avvenga, in media, nella metà dei turni in cui viene svolta un'azione di attacco) → (2+2+1*2+1*2+1*2) *40.000 + (1*2) *20.000 = 440.000 accessi/g

(10) Due letture in Possiede, due letture in STATO TABELLONE, una scrittura in Effettua, una scrittura in TURNO, due scritture in STATO TABELLONE per modificare l'attributo 'numarmate' → (2+2+1*2+1*2+2*2) * 50.000 = 600.000 accessi/g

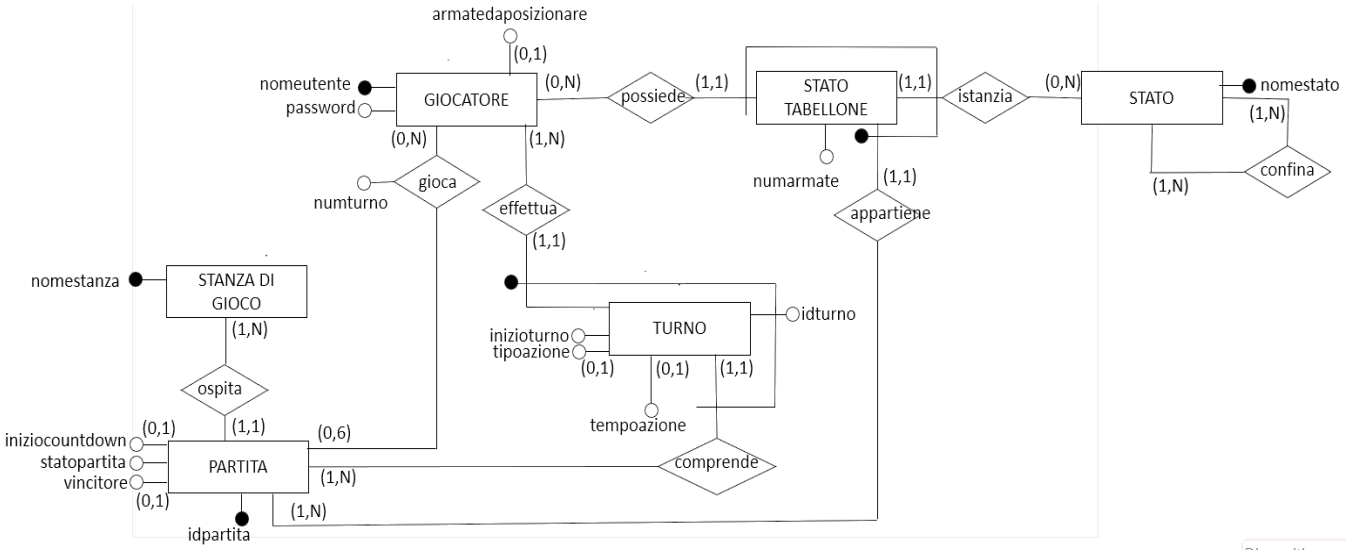
(11) 10 letture in Gioca, 10 letture in PARTITA, 10 letture in Vince, 10 letture in Ospita (supponendo che in media un giocatore partecipi a 10 partite) → (10+10+10+10) *100 = 4000 accessi/settimana

Ristrutturazione dello schema E-R

Per la ristrutturazione dello schema E-R, innanzitutto elimino la generalizzazione presente, inglobando l'entità padre UTENTE nell'entità figlia GIOCATORE, che quindi erediterà gli attributi 'nomeutente' e 'password'; l'altra entità figlia MODERATORE con la relativa associazione Gestisce, invece, non le considero più in quanto non mi interessa mantenere informazioni su quale moderatore crea/gestisce una determinata stanza di gioco.

Proseguendo con la ristrutturazione mi accorgo che, introducendo una ridondanza e cioè un nuovo attributo opzionale 'vincitore' su PARTITA, il costo dell'operazione (11) sarebbe di 3000 accessi/settimana che, paragonato al calcolo del costo dell'operazione (11) in assenza di ridondanza di 4000 accessi/settimana e un risparmio di pochi kilobyte di memoria aggiuntivi (che servirebbero per memorizzare il nuovo attributo), risulta conveniente.
Quindi elimino l'associazione Vince tra PARTITA e GIOCATORE e la sostituisco con un attributo opzionale 'vincitore' sull'entità PARTITA.

Di seguito, il diagramma E-R ristrutturato:



Trasformazione di attributi e identificatori

Traduzione di entità e associazioni

STANZA DI GIOCO (<u>nomestanza</u>)

PARTITA (<u>idpartita</u>, stanzadigioco, statopartita, vincitore*, iniziocountdown*)

GIOCATORE (<u>nomeutente</u>, password, armatedaposizionare*)

GIOCA (numturno, <u>giocatore</u>, <u>partita</u>)

TURNO (idturno, giocatore, partita, tipoazione*, tempoazione*, inizioturno)

STATO TABELLONE (stato, partita, giocatore, numarmate)

STATO (nomestato)

CONFINA (stato, confinante)

Vincoli di integrità referenziale presenti:

PARTITA (stanzadigioco) ⊆ STANZA DI GIOCO (nomestanza)

GIOCA (giocatore) ⊆ GIOCATORE (nomeutente)

GIOCA (partita) ⊆ PARTITA (idpartita)

TURNO (giocatore) ⊆ GIOCATORE (nomeutente)

TURNO (partita) ⊆ PARTITA (idpartita)

STATO TABELLONE (stato) ⊆ STATO (nomestato)

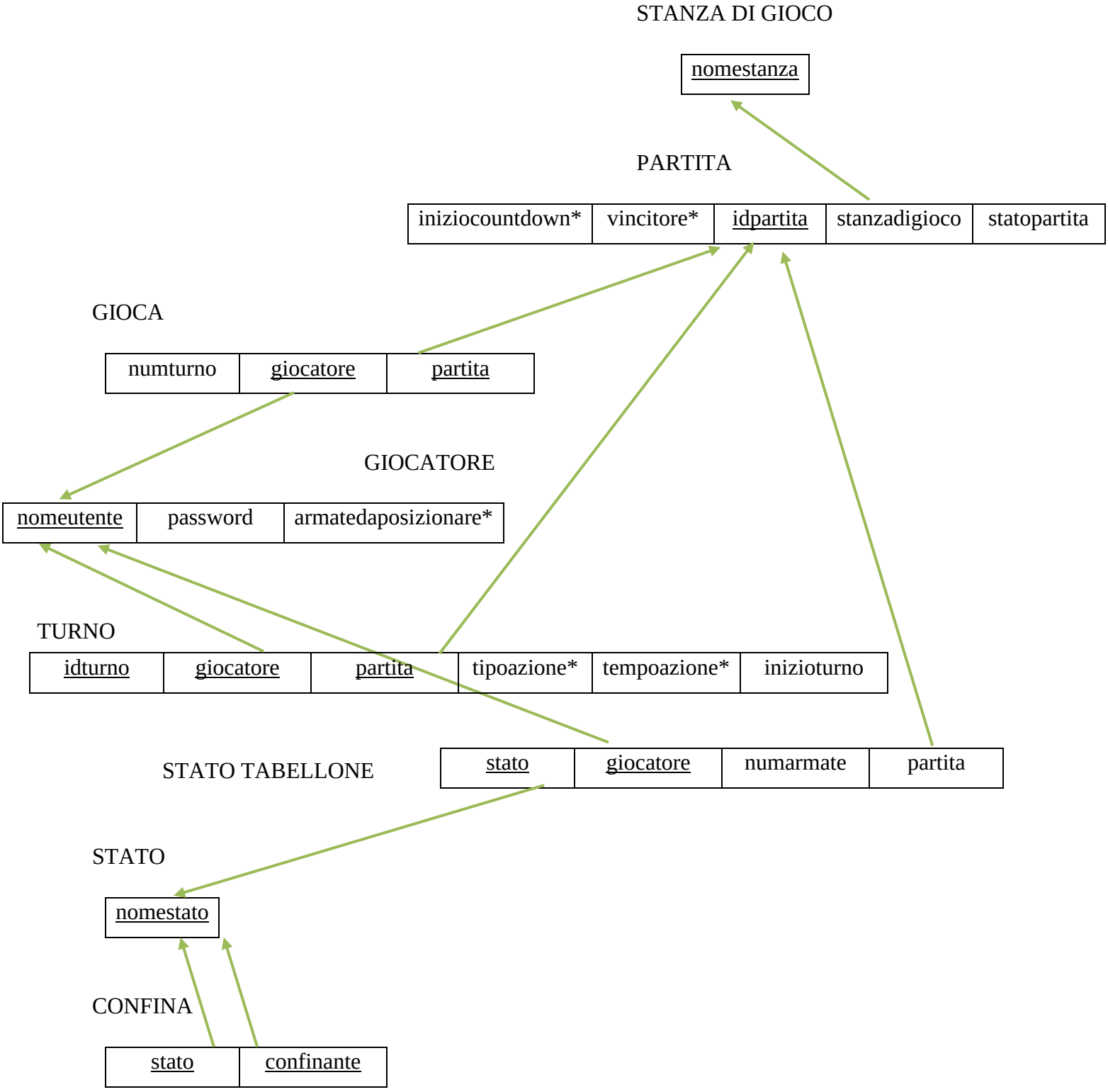
STATO TABELLONE (partita) ⊆ PARTITA (idpartita)

STATO TABELLONE (giocatore) ⊆ GIOCATORE (nomeutente)

CONFINA (stato) ⊆ STATO (nomestato)

CONFINA (confinante) ⊆ STATO (nomestato)

Rappresentazione grafica del modello relazionale completo:



Normalizzazione del modello relazionale

1NF: Il modello relazionale è in 1FN, infatti risulta che ogni attributo è definito su un dominio con valori atomici e contiene un valore singolo di quel dominio.

2NF: Il modello relazionale è in 2NF, infatti è innanzitutto in 1NF e tutti gli attributi non-chiave dipendono funzionalmente dall'intera chiave e non solo da una parte di essa.

3NF: Il modello relazionale è in 3NF, infatti è innanzitutto in 2NF e tutti gli attributi non-chiave dipendono direttamente dalla chiave.

5. Progettazione fisica

Utenti e privilegi

All'interno dell'applicazione sono stati previsti tre tipi di utenti: login, giocatore e moderatore. A ognuno di questi è associato un ruolo (omonimo all'utente stesso) che ha unicamente il privilegio di eseguire delle stored procedures. Tali privilegi sono stati assegnati cercando di rimanere il più fedele possibile al POLP (Principle Of Least Privilege) e contemporaneamente di garantire a ciascun utente il necessario per poter operare al meglio sulla base di dati.

Utente login: tale utente è l'utente con il quale si accede preliminarmente alla base di dati e ha il privilegio di tipo EXECUTE sulla stored procedure 'login' che permette appunto di effettuare una lettura nella tabella 'Utente' per poter consentire l'accesso come uno degli altri due tipi di utenti previsti o rifiutare l'accesso in caso di credenziali errate. Inoltre, ha anche il privilegio di tipo EXECUTE sulla stored procedure 'aggiugi_giocatore'.

Utente giocatore: questo tipo di utente sarà utilizzato dai giocatori dell'applicazione e ha privilegi di tipo EXECUTE sulle seguenti stored procedures:

- 'crea_partita'
- 'visualizza_stanze_libere'
- 'partecipa_partita'
- 'posiziona_armate'
- 'effettua_attacco'
- 'sposta_armate'
- 'storico_partite'
- 'visualizza_partite_in_attesa'
- 'stato_di_gioco'

Utente moderatore: questo tipo di utente sarà utilizzato dai moderatori dell'applicazione e ha privilegi di tipo EXECUTE sulle seguenti stored procedures:

- 'aggiungi_moderatore' → assegno il privilegio di eseguire questa procedura all'utente moderatore in modo da non permettere a qualsiasi utente di potersi registrare come tale; cioè per permettere solamente ad un moderatore già registrato di poter registrare un nuovo moderatore

- ‘crea_stanzadigioco’
- ‘report_moderatore’

Strutture di memorizzazione

Tabella <Utente>		
Attributo	**Tipo di dato**	**Attributi[2]**
nomeutente	VARCHAR(45)	PK, NN
password	VARCHAR(45)	NN
armatedaposizionare	INT	UN
tipo	ENUM(‘giocatore’, ‘moderatore’)	NN

Tabella <Stanza di gioco>		
Attributo	**Tipo di dato**	**Attributi[2]**
nomestanza	VARCHAR(15)	PK, NN

Tabella <Gioca>		
Attributo	**Tipo di dato**	**Attributi[2]**
numturno	INT	NN, UN
giocatore	VARCHAR(45)	PK, NN
partita	INT	PK, NN, UN

Tabella <Partita>		
Attributo	**Tipo di dato**	**Attributi[2]**
idpartita	INT	PK, NN, AI, UN
stanzadigioco	VARCHAR(15)	NN

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

statopartita	ENUM(‘inattesa’, ‘incorso’, ‘terminata’)	NN
iniziocountdown	DATETIME	
vincitore	VARCHAR(45)	

Tabella <Turno>		
Attributo	**Tipo di dato**	**Attributi[2]**
idturno	INT	PK, NN, AI, UN
giocatore	VARCHAR(45)	PK, NN
partita	INT	PK, NN, UN
tipoazione	ENUM(‘posizionamento’, ‘attacco’, ‘spostamento’, ‘nonsvolta’)	
tempoazione	DATETIME	
inizioturno	DATETIME	NN

Tabella <Stato tabellone>		
Attributo	**Tipo di dato**	**Attributi[2]**
partita	INT	PK, NN, UN
Stato	VARCHAR(25)	PK, NN
giocatore	VARCHAR(45)	PK, NN
numarmate	INT	NN, UN

Tabella <Stato>		
Attributo	**Tipo di dato**	**Attributi[2]**
nomestato	VARCHAR(25)	PK, NN

Tabella <Confina>		
Attributo	**Tipo di dato**	**Attributi[2]**
stato	VARCHAR(25)	PK, NN
confinante	VARCHAR(25)	PK, NN

Indici

Sono stati inseriti, in ogni tabella, indici di tipo PRIMARY e dove necessario per implementare le Foreign Key anche di tipo INDEX. In questo modo si ottimizzano enormemente le prestazioni delle queries, specialmente quelle con una struttura più articolata e che toccano più dati; inoltre si permette anche al DBMS di controllare più rapidamente che i vincoli di chiave e di Foreign Key vengano rispettati.

Tabella <Utente>	
Indice <PRIMARY>	**Tipo[3]:**
nomeutente	PR

Tabella <Stanzadigioco>	
Indice <PRIMARY>	**Tipo[3]:**
nomestanza	PR

Tabella <Gioca>	
Indice <PRIMARY>	**Tipo[3]:**
giocatore, partita	PR
Indice <fk_Gioca_Partita1_idx>	**Tipo[3]:**
partita	IDX

Tabella <Partita>	
Indice <PRIMARY>	**Tipo[3]:**
idpartita	PR
Indice <fk_partita_Stanzadigioco1_idx>	**Tipo[3]:**
stanzadigioco	IDX

[3] IDX = index, UQ = unique, FT = full text, PR = primary.

Tabella <Turno>	
Indice <PRIMARY>	**Tipo[3]:**
idturno, partita, giocatore	PR
Indice <fk_Turno_partita1_idx>	**Tipo[3]:**
partita	IDX
Indice <fk_Turno_Utente1_idx>	**Tipo[3]:**
giocatore	IDX

Tabella <Statotabellone>	
Indice <PRIMARY>	**Tipo[3]:**
stato, partita, giocatore	PR
Indice <fk_Statotabellone_partita1_idx>	**Tipo[3]:**
partita	IDX
Indice <fk_Statotabellone_Stato1_idx>	**Tipo[3]:**
stato	IDX

Tabella <Stato>	
Indice <PRIMARY>	**Tipo[3]:**
nomestato	PR

Tabella <Confina>	
Indice <PRIMARY>	**Tipo[3]:**
stato, confinante	PR
Indice <fk_Confina_Stato2_idx>	**Tipo[3]:**
confinante	IDX

Trigger

Per questa applicazione sono stati realizzati in tutto quattro trigger, due dei quali (`Gioca_BEFORE_INSERT` e `Turno_BEFORE_UPDATE`) riguardano semplicemente l'implementazione dei check necessari per rispettare alcune regole aziendali proposte. In dettaglio:

- `Partita_AFTER_UPDATE` → dopo un update nella tabella 'Partita', se il nuovo statopartita è 'incorso' (cioè la partita è stata avviata) viene chiamata la procedura 'assegna_stati' per assegnare gli Stati del tabellone casualmente ai giocatori e subito dopo viene chiamata anche la procedura 'gioca_turno' per inserire un nuovo turno, nell'omonima tabella, relativo al giocatore con numturno = 1 (cioè al primo giocatore che è entrato nella partita in questione)
- `Gioca_BEFORE_INSERT` → prima di un insert nella tabella 'Gioca', controlla che l'attributo numturno sia compreso nell'intervallo [1,6] visto che ad una partita possono partecipare da 1 a 6 giocatori
- `Turno_BEFORE_INSERT` → prima di un insert nella tabella 'Turno' controlla che i turni stiano "girando" in funzione del tempo di ingresso dei giocatori nella stanza
- `Turno_BEFORE_UPDATE` → prima di un update nella tabella 'Turno', controlla che il tempo di inizio dell'azione sia più grande del tempo di inizio del turno

Di seguito il codice SQL dei trigger:

```
DELIMITER $$


USE `mydb`$$

DROP TRIGGER IF EXISTS `mydb`.`Partita_AFTER_UPDATE` $$

USE `mydb`$$

CREATE DEFINER = CURRENT_USER TRIGGER `mydb`.`Partita_AFTER_UPDATE` AFTER UPDATE ON `Partita` FOR EACH ROW

BEGIN

    declare var_primogiocatore VARCHAR(45);
```

```
    if new.statopartita = 'incorso' then
                    call assegna_stati(old.idpartita);

    select giocatore
                    from Gioca
        where Gioca.partita = old.idpartita and numturno = 1
        into var_primogiocatore;

            call gioca_turno(var_primogiocatore);
    end if;
END$$

USE `mydb`$$
DROP TRIGGER IF EXISTS `mydb`.`Gioca_BEFORE_INSERT` $$
USE `mydb`$$
CREATE DEFINER = CURRENT_USER TRIGGER `mydb`.`Gioca_BEFORE_INSERT` BEFORE INSERT ON `Gioca` FOR EACH ROW
BEGIN
    if(new.numturno>6 or new.numturno<1) then
        signal sqlstate '45000' set message_text="E' stato raggiunto il numero massimo di giocatori per questa partita.";
    end if;
END$$
```

```
USE `mydb`$$

DROP TRIGGER IF EXISTS `mydb`.`Turno_BEFORE_INSERT` $$

USE `mydb`$$

CREATE DEFINER = CURRENT_USER TRIGGER `mydb`.`Turno_BEFORE_INSERT` BEFORE INSERT ON `Turno` FOR EACH ROW

BEGIN

        declare var_numturno INT;

    declare var_numturnoprecedente INT;

    declare var_partita INT;

    declare var_numgiocatori INT;


    set var_numturnoprecedente = 0;


        select restituisci_partita_incorso(new.giocatore) into var_partita;


    select numturno

                from Gioca

        where Gioca.giocatore = new.giocatore and Gioca.partita = var_partita

        into var_numturno;


    select count(*)

                from Gioca join Partita on Gioca.partita = Partita.idpartita

                where idpartita = var_partita

        into var_numgiocatori;
```

```
   select numturno
                from Turno join Utente on Turno.giocatore = Utente.nomeutente
                        join Gioca on Utente.nomeutente = Gioca.giocatore
                where Turno.partita = var_partita
      order by idturno desc limit 1
                into var_numturnoprecedente;

   if (var_numturnoprecedente % var_numgiocatori) + 1 <> var_numturno then
                signal sqlstate '45000' set message_text = "Non e' il tuo turno.";
         end if;
END$$
```

```
USE `mydb`$$
DROP TRIGGER IF EXISTS `mydb`.`Turno_BEFORE_UPDATE` $$
USE `mydb`$$
CREATE DEFINER = CURRENT_USER TRIGGER `mydb`.`Turno_BEFORE_UPDATE` BEFORE UPDATE ON `Turno` FOR EACH ROW
BEGIN
        if(new.tempoazione < old.inizioturno) then
    signal sqlstate '45000' set message_text="Il tempo di inizio dell'azione non puo' essere più piccolo del tempo di inizio del turno.";
   end if;
END$$
```

Eventi

Sono stati realizzati due eventi:

- `avvia_partita` → quando il countdown di 2 minuti associato ad una partita scade, la partita viene avviata
- `controlla_turno` → quando il timer di 2 minuti associato a un turno scade e non e' stata svolta alcuna azione, il turno passa al giocatore successivo

Di seguito il codice SQL degli eventi:

```
set global event_scheduler = ON;


delimiter !
CREATE EVENT if not exists `mydb`.`avvia_partita`
ON SCHEDULE EVERY 2 second STARTS CURRENT_TIMESTAMP ON COMPLETION PRESERVE
comment "Quando il countdown di 2 minuti associato ad una partita scade, la partita viene avviata."
DO
BEGIN
	update Partita set statopartita = 'incorso'
		where now() > interval 2 minute + iniziocountdown and statopartita = 'inattesa';
END !


delimiter !
CREATE EVENT if not exists `mydb`.`controlla_turno`
```

```
ON SCHEDULE EVERY 2 second STARTS CURRENT_TIMESTAMP ON COMPLETION PRESERVE

comment "Quando il timer di 2 minuti associato a un turno scade e non e' stata svolta alcuna azione, il turno passa al giocatore successivo."

DO

BEGIN

        declare var_tipoazione ENUM('posizionamento', 'spostamento', 'attacco', 'nonsvolta');

        declare var_giocatore VARCHAR(45);

    declare var_partita INT;


        update Turno set tipoazione = 'nonsvolta', tempoazione = now()

                where now() > interval 2 minute + inizioturno and tipoazione is null and tipoazione is null;


        select tipoazione, giocatore, partita

                from Turno

       order by tempoazione desc limit 1

       into var_tipoazione, var_giocatore, var_partita;


        if var_tipoazione = 'nonsvolta' then

                call turno_successivo(var_giocatore, var_partita);

        end if;

END !
```

Inoltre, per questa applicazione sono state realizzate anche due **funzioni** che vengono chiamate all'interno di diverse stored procedure e che riporto qui non essendoci una sezione apposita:

- `restituisci_partita` → a partire da un giocatore, restitusce l'idpartita relativo all'ultima partita a cui il giocatore in questione ha partecipato
- `restituisci_partita_incorso` → a partire da un giocatore, restuituisce l'idpartita relativo alla partita in corso a cui il giocatore sta partecipando

Di seguito il codice SQL delle funzioni:

```
delimiter !
create function `mydb`.`restituisci_partita` (var_giocatore VARCHAR(45))
returns INT
reads sql data
begin
	declare var_partita INT;

	select idpartita
		from Gioca join Partita on Gioca.partita = Partita.idpartita
    where Gioca.giocatore = var_giocatore
    order by idpartita desc limit 1
    into var_partita;
	return var_partita;
END !

delimiter !
create function `mydb`.`restituisci_partita_incorso` (var_giocatore VARCHAR(45))
```

```
returns INT
reads sql data
begin
    declare var_partita INT;

    select idpartita
        from Gioca join Partita on Gioca.partita = Partita.idpartita
  where Gioca.giocatore = var_giocatore and statopartita = 'incorso'
  into var_partita;
    return var_partita;
END !
```

Viste

Non sono state previste delle viste.

Stored Procedures e transazioni

Per questa applicazione sono state realizzate 19 stored procedure:

- ‘login’ → permette di effettuare una lettura nella tabella ‘Utente’ per poter consentire l’accesso (come giocatore o come moderatore) o rifiutare l’accesso in caso di credenziali errate
- ‘crea_stanzadigioco’ → permette di creare una nuova stanza di gioco
- ‘aggiungi_giocatore’ → permette a un nuovo utente di registrarsi come giocatore
- ‘partecipa_partita’ → permette di partecipare a una partita che è in attesa di iniziare; in questa stored procedure è stata inserita una transazione con livello di isolamento READ COMMITTED per evitare dirty read (letture sporche) e garantire quindi che le letture, effettuate al suo interno, siano relative solo a dati committati
- ‘assegna_stati’ → permette di assegnare gli Stati del tabellone casualmente ai giocatori nel momento in cui una partita viene avviata

- ‘report_moderatore’ → permette di visualizzare un report che mostri quante stanze di gioco hanno attualmente partite in corso, quanti giocatori stanno partecipando alla partita e il numero totale di giocatori che hanno effettuato almeno un’azione negli ultimi 15 minuti, che non sono all’interno di alcuna stanza di gioco. In questa stored procedure è stata inserita una transazione con livello di isolamento REPEATABLE READ per garantire le letture ripetibili; in particolare per garantire che i risultati della query relativa al numero di stanze di gioco che hanno attualmente partite in corso e della query relativa a stanza di gioco, id partita in corso e numero di giocatori al suo interno siano coerenti tra loro. Inoltre, dato che all’interno di questa transazione vengono effettuate soltanto letture, posso migliorare le prestazioni, marcando la transazione stessa con la modalità d’accesso read only
- ‘storico_partite’ → permette ad un giocatore di visualizzare lo storio di tutte le partite da lui giocate, mostrando cioè per ognuna id partita, stanza di gioco e vincitore; in questa stored procedure è stata inserita una transazione con livello di isolamento READ COMMITTED per evitare letture sporche
- ‘stato_di_gioco’ → permette di visualizzare lo stato di gioco di una partita, ovvero per ogni Stato del tabellone di gioco il numero di armate su di esso presenti, il suo possessore e uno Stato ad esso adiacente con, anche per esso, numero di armate presenti e possessore; in questa stored procedure è stata inserita una transazione con livello di isolamento READ COMMITTED per evitare letture sporche e inoltre dato che all’interno di questa transazione vengono effettuate soltanto letture, posso migliorare le prestazioni, marcando la transazione stessa con la modalità d’accesso read only
- ‘posiziona_armate’ → permette di effettuare un’azione di posizionamento armate; in particolare permette al giocatore, nel corso di un suo turno, di posizionare un certo numero di armate in uno Stato da lui posseduto
- ‘assegna_nuove_armate’ → viene chiamata all’interno della procedura ‘turno_successivo’ e permette, al termine di un turno e se è stata svolta un’azione durante esso, di assegnare al giocatore un numero di nuove armate da posizionare pari al numero di Stati posseduti diviso tre, arrotondato per eccesso; in questa stored procedure è stata inserita una transazione con livello di isolamento READ COMMITTED per evitare letture sporche
- ‘gioca_turno’ → permette di inserire un nuovo turno, nella tabella ‘Turno’ appunto
- ‘aggiungi_moderatore’ → permette di inserire nel sistema un nuovo moderatore

- ‘effettua_attacco’ → permette di effettuare un’azione di attacco; in particolare permette al giocatore, nel corso di un suo turno, di far partire un attacco da uno Stato da lui posseduto verso uno adiacente, posseduto da un altro giocatore
- ‘sposta_armate’ → permette di effettuare un’azione di spostamento armate; in particolare permette al giocatore, nel corso di un suo turno, di spostare un certo numero di armate da uno Stato da lui posseduto ad un altro ad esso adiacente posseduto sempre da lui
- ‘controlla_vittoria_lancio_dadi’ → viene chiamata all’interno della procedura ‘effettua_attacco’ e permette di conoscere il numero di armate perse dall’attaccante e il numero di armate perse dal difensore in seguito al lancio dei dadi che simulano un’azione di attacco; in questa stored procedure è stata inserita una transazione con livello di isolamento READ UNCOMMITTED, che è il livello di protezione da anomalie più basso, perché al suo interno non viene fatta alcuna lettura
- ‘visualizza_stanze_libere’ → permette di visualizzare le stanze di gioco libere, cioè quelle che non ospitano alcuna partita in corso o in attesa di iniziare; in questa stored procedure è stata inserita una transazione con livello di isolamento READ COMMITTED per evitare letture sporche e inoltre dato che all’interno di questa transazione vengono effettuate soltanto letture, posso migliorare le prestazioni, marcando la transazione stessa con la modalità d’accesso read only
- ‘visualizza_partite_in_attesa’ → permette di visualizzare tutte le partite in attesa di iniziare, ciascuna con id partita, stanza di gioco e numero di giocatori già entrati; in questa stored procedure è stata inserita una transazione con livello di isolamento READ COMMITTED per evitare letture sporche e modalità d’accesso read only
- ‘crea_partita’ → permette di creare una nuova partita all’interno di una stanza di gioco libera; in questa stored procedure è stata inserita una transazione con livello di isolamento READ COMMITTED per evitare dirty read
- ‘turno_successivo’ → viene chiamata all’interno dell’evento ‘controlla_turno’ e nelle procedure ‘effettua_attacco’, ‘sposta_armate’, ‘posiziona_armate’ e permette di conoscere il giocatore che dovrà giocare il turno successivo all’interno di una certa partita e lo usa come parametro quando chiama la procedura ‘gioca_turno’. Inoltre, se è stata svolta un’azione nel turno appena terminato chiama la procedura ‘assegna_nuove_armate’

Di seguito il codice SQL delle stored procedure, nello stesso ordine in cui sono state descritte sopra:

```
-- -----------------------------------------------------
-- procedure login
-- -----------------------------------------------------

USE `mydb`;
DROP procedure IF EXISTS `mydb`.`login`;

DELIMITER $$
USE `mydb`$$
CREATE PROCEDURE `login` (in var_nomeutente VARCHAR(45), in var_password VARCHAR(45), out var_ruolo INT)
BEGIN
	declare var_ruolo_utente ENUM('giocatore', 'moderatore');
  select tipo from Utente
		where nomeutente = var_nomeutente
    and password = md5(var_password)
    into var_ruolo_utente;

  -- enum corrispondente nel client
	if var_ruolo_utente = 'giocatore' then set var_ruolo = 1;
  elseif var_ruolo_utente = 'moderatore' then set var_ruolo = 2;
  else set var_ruolo = 3;
  end if;
END$$
```

```
DELIMITER ;

-- -----------------------------------------------------
-- procedure crea_stanzadigioco
-- -----------------------------------------------------

USE `mydb`;
DROP procedure IF EXISTS `mydb`.`crea_stanzadigioco`;

DELIMITER $$
USE `mydb`$$
CREATE PROCEDURE `crea_stanzadigioco` (in var_nomestanza VARCHAR(15))
BEGIN
	insert into Stanzadigioco VALUES (var_nomestanza);
END$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure aggiungi_giocatore
-- -----------------------------------------------------

USE `mydb`;
```

```
DROP procedure IF EXISTS `mydb`.`aggiungi_giocatore`;


DELIMITER $$

USE `mydb`$$

CREATE PROCEDURE `aggiungi_giocatore` (in var_nomeutente VARCHAR(45), in var_password VARCHAR(45))

BEGIN

    insert into Utente (nomeutente, password, tipo) VALUES(var_nomeutente, md5(var_password), 'giocatore');

END$$


DELIMITER ;


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

-- procedure partecipa_partita

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


USE `mydb`;

DROP procedure IF EXISTS `mydb`.`partecipa_partita`;


DELIMITER $$

USE `mydb`$$

CREATE PROCEDURE `partecipa_partita` (in var_giocatore VARCHAR(45), in var_partita INT)

BEGIN

  declare var_partecipare INT;
```

```
declare var_numturno INT;
declare var_statopartita ENUM('inattesa', 'incorso', 'terminata');

        declare exit handler for sqlexception
begin
                rollback;  -- rollback any changes made in the transaction
                resignal;  -- raise again the sql exception to the caller
end;

        set transaction isolation level read committed;
        start transaction;

-- controllo se il giocatore sta già partecipando a una partita
select count(*)
                from Gioca join Partita on Gioca.partita = Partita.idpartita
                where giocatore = var_giocatore and statopartita <> 'terminata'
    into var_partecipare;

if var_partecipare > 0 then
    signal sqlstate '45000' set message_text = "Il giocatore sta gia' partecipando a un'altra partita";
end if;

-- controllo se var_partita è una partita in attesa (se in corso o terminata non è possibile partecipare)
select statopartita from Partita
```

```
            where idpartita = var_partita
    into var_statopartita;

  if var_statopartita <> 'inattesa' then
            signal sqlstate '45001' set message_text = "Non e' possibile partecipare a questa partita, perche' in corso o terminata";
        end if;

  -- azzero il numero di armatedaposizionare per il giocatore, che magari può aver giocato precedentemente
  -- un'altra partita e avere ancora delle armate vecchie
  update Utente set armatedaposizionare = 0 where nomeutente = var_giocatore;

      -- salvo in var_numturno il numero assegnato all'ultimo giocatore entrato nella partita in esame e poi lo incremento di 1
  select count(giocatore) from Gioca
            where partita = var_partita
    into var_numturno;

  set var_numturno = var_numturno +1;
  insert into Gioca VALUES(var_numturno, var_giocatore, var_partita);

  -- quando entra il terzo giocatore viene avviato il countdown per far iniziare la partita dopo 2 minuti
  if var_numturno = 3 then
```

```
                update Partita set iniziocountdown = now() where idpartita = var_partita;
        end if;
        commit;
END$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure assegna_stati
-- -----------------------------------------------------

USE `mydb`;
DROP procedure IF EXISTS `mydb`.`assegna_stati`;

DELIMITER $$
USE `mydb`$$
CREATE PROCEDURE `assegna_stati` (in var_partita INT)
BEGIN
        declare var_numturno INT;
    declare var INT;
    declare var_nomegiocatore VARCHAR(45);
    declare var_numgiocatori INT;
    declare var_nomestato VARCHAR(25);
```

```
   set var = 0;


        select count(giocatore)
                from Gioca
      where Gioca.partita = var_partita
                into var_numgiocatori;


   while var < 42 do
                select nomestato
                        from Stato
        where nomestato not in (select stato from Statotabellone where partita = var_partita)
        order by rand() limit 1
        into var_nomestato;


                -- numturno del giocatore a cui devo assegnare lo stato
      set var_numturno = (var % var_numgiocatori) + 1;
                select giocatore
                        from Gioca
                        where numturno = var_numturno and Gioca.partita = var_partita
        into var_nomegiocatore;
                insert into Statotabellone (giocatore, partita, stato) values (var_nomegiocatore, var_partita, var_nomestato);
                set var = var+1;
   end while;
END$$
```

```
DELIMITER ;

-- -----------------------------------------------------
-- procedure report_moderatore
-- -----------------------------------------------------

USE `mydb`;
DROP procedure IF EXISTS `mydb`.`report_moderatore`;

DELIMITER $$
USE `mydb`$$
CREATE PROCEDURE `report_moderatore` ()
BEGIN
        set transaction read only;
        set transaction isolation level repeatable read;
        start transaction;

   select count(stanzadigioco) as numstanze
                from Stanzadigioco join Partita on Stanzadigioco.nomestanza = Partita.stanzadigioco
      where statopartita = 'incorso';

        select stanzadigioco, idpartita, count(giocatore) as numgiocatori
                from Partita join Gioca on Partita.idpartita = Gioca.partita
```

```
                where Partita.statopartita = "incorso"
                group by idpartita;

        select count(distinct nomeutente) as giocatoriusciti
                from Partita join Gioca on Partita.idpartita = Gioca.partita
                        join Utente on Gioca.giocatore = Utente.nomeutente
                        join Turno on Utente.nomeutente = Turno.giocatore
                where Partita.statopartita = "terminata" and Turno.tempoazione >= (now()-interval 15
minute);

    commit;
END$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure storico_partite
-- -----------------------------------------------------

USE `mydb`;
DROP procedure IF EXISTS `mydb`.`storico_partite`;

DELIMITER $$
USE `mydb`$$
CREATE PROCEDURE `storico_partite` (in var_giocatore VARCHAR(45))
```

```
BEGIN
        declare var_numpartite INT;

   declare exit handler for sqlexception
   begin
                rollback;  -- rollback any changes made in the transaction
                resignal;  -- raise again the sql exception to the caller
   end;

        set transaction isolation level read committed;
   start transaction;

   select count(idpartita)
                from Gioca join Partita on Gioca.partita = Partita.idpartita
      where giocatore = var_giocatore and statopartita = 'terminata'
      into var_numpartite;

   if var_numpartite < 1 then
                signal sqlstate '45000' set message_text = "L'utente non ha partite terminate";
        end if;

        select idpartita, stanzadigioco, vincitore
                from Gioca join Partita on Gioca.partita = Partita.idpartita
      where giocatore = var_giocatore and statopartita = 'terminata';
```

```
        commit;
END$$


DELIMITER ;


-- -----------------------------------------------------
-- procedure stato_di_gioco
-- -----------------------------------------------------


USE `mydb`;
DROP procedure IF EXISTS `mydb`.`stato_di_gioco`;


DELIMITER $$
USE `mydb`$$
CREATE PROCEDURE `stato_di_gioco` (in var_giocatore VARCHAR(45))
BEGIN
   declare var_partita INT;
   declare var_statopartita ENUM('inattesa', 'incorso', 'terminata');
   declare var_numturno INT;
   declare var_numturnoprecedente INT;
   declare var_numgiocatori INT;


   declare var_statiposseduti INT;
```

```
declare exit handler for sqlexception
begin
  rollback;  -- rollback any changes made in the transaction
            resignal;  -- raise again the sql exception to the caller
end;

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

select restituisci_partita(var_giocatore) into var_partita;

      -- controllo se la partita e' stata avviata o e' ancora in attesa o è gia' terminata
      select statopartita
            from Partita
            where idpartita = var_partita
            into var_statopartita;

      if var_statopartita = 'inattesa' then
            signal sqlstate '45000' set message_text = "Attendere che la partita inizi";
      end if;

      if var_statopartita = 'terminata' then
            signal sqlstate '45001' set message_text = "La partita e' terminata";
```

```
        end if;

        -- controllo che il giocatore non abbia perso
        select count(*)
                from Statotabellone
                where giocatore = var_giocatore and partita = var_partita
                into var_statiposseduti;

        if var_statiposseduti = 0 then
                signal sqlstate '45002' set message_text = "Hai perso la partita!";
        end if;

        -- controllo che sia il turno del giocatore che vuole visualizzare lo stato di gioco
    select numturno
                from Gioca
       where Gioca.giocatore = var_giocatore and Gioca.partita = var_partita
       into var_numturno;

    select count(*)
                from Gioca join Partita on Gioca.partita = Partita.idpartita
                where idpartita = var_partita
       into var_numgiocatori;

    select numturno
```

```
                from Turno join Utente on Turno.giocatore = Utente.nomeutente
                        join Gioca on Utente.nomeutente = Gioca.giocatore
                where Turno.partita = var_partita
        order by idturno desc limit 1
                into var_numturnoprecedente;

    if (var_numturnoprecedente % var_numgiocatori) <> var_numturno then
                signal sqlstate '45000' set message_text = "Non e' il tuo turno";
            end if;

            select s1.giocatore, s1.numarmate, nomestato, confinante, s2.giocatore, s2.numarmate
                from Statotabellone as s1 join Stato on s1.stato = Stato.nomestato
                        join Confina on Stato.nomestato = Confina.stato
                        join Statotabellone as s2 on Confina.confinante = s2.stato
                where s1.partita = var_partita and s2.partita = var_partita;

            commit;
END$$

DELIMITER ;

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

-- procedure posiziona_armate

-- ----------------------------------------------------
```

```
USE `mydb`;

DROP procedure IF EXISTS `mydb`.`posiziona_armate`;


DELIMITER $$

USE `mydb`$$

CREATE PROCEDURE `posiziona_armate` (in var_giocatore VARCHAR(45), in var_numarmate INT, in var_stato VARCHAR(25))

BEGIN

        declare var_partita INT;

   declare var_statopartita ENUM('inattesa', 'incorso', 'terminata');

   declare var_possiede INT;

   declare var_idturno INT;

   declare var_armatedaposiz INT;

   declare var_tipoazione ENUM('posizionamento', 'attacco', 'spostamento', 'nonsvolta');

   declare var_statiposseduti INT;


   declare exit handler for sqlexception

   begin

     rollback;  -- rollback any changes made in the transaction

     resignal;  -- raise again the sql exception to the caller

   end;


   select restituisci_partita(var_giocatore) into var_partita;
```

```
-- controllo se la partita e' stata avviata o e' ancora in attesa o e' gia' terminata
select statopartita
        from Partita
        where idpartita = var_partita
        into var_statopartita;

if var_statopartita = 'inattesa' then
        signal sqlstate '45000' set message_text = "Attendere che la partita inizi";
end if;

if var_statopartita = 'terminata' then
        signal sqlstate '45001' set message_text = "La partita e' terminata";
end if;

-- controllo che il giocatore non abbia perso
select count(*)
        from Statotabellone
        where giocatore = var_giocatore and partita = var_partita
        into var_statiposseduti;

if var_statiposseduti = 0 then
        signal sqlstate '45002' set message_text = "Hai perso la partita!";
end if;
```

```
        -- controllo che non sia già stata effettuata un'azione nello stesso turno
        select idturno
                from Turno
                where partita = var_partita and Turno.giocatore = var_giocatore
    order by idturno desc limit 1
    into var_idturno;

  select tipoazione
                from Turno
    where idturno = var_idturno
    into var_tipoazione;

  if var_tipoazione is not null then
                signal sqlstate '45003' set message_text = "Hai gia' giocato il turno";
        end if;

        -- controllo se lo stato appartiene al giocatore
        select count(*)
                from Statotabellone
    where Statotabellone.giocatore = var_giocatore and partita = var_partita and stato = upper(var_stato)
                into var_possiede;

  if var_possiede <> 1 then
                signal sqlstate '45004' set message_text = "Lo stato non appartiene al giocatore";
```

```
        end if;

        -- controllo se il giocatore effettivamente possiede il numarmate che vuole posizionare
        select armatedaposizionare
                from Utente join Gioca on Utente.nomeutente = Gioca.giocatore
                where nomeutente = var_giocatore and partita = var_partita
                into var_armatedaposiz;

        if var_numarmate > var_armatedaposiz then
                signal sqlstate '45005' set message_text = "Il giocatore non possiede il numarmate che vuole posizionare";
        end if;

        update Statotabellone set numarmate = (numarmate + var_numarmate)
                        where Statotabellone.giocatore = var_giocatore and partita = var_partita and stato = upper(var_stato);

    update Utente set armatedaposizionare = armatedaposizionare - var_numarmate
                where nomeutente = var_giocatore;

    update Turno set tipoazione = 'posizionamento', tempoazione = now()
                where idturno = var_idturno and tipoazione is null;

    call turno_successivo(var_giocatore, var_partita);
END$$
```

```
DELIMITER ;

-- -----------------------------------------------------
-- procedure assegna_nuove_armate
-- -----------------------------------------------------

USE `mydb`;
DROP procedure IF EXISTS `mydb`.`assegna_nuove_armate`;

DELIMITER $$
USE `mydb`$$
CREATE PROCEDURE `assegna_nuove_armate` (in var_giocatore VARCHAR(45))
BEGIN
        declare var_partita INT;
    declare var_numstati INT;
    declare var_nuovearmate INT;

    set transaction isolation level read committed;
    start transaction;

        select restituisci_partita_incorso(var_giocatore) into var_partita;

    -- calcolo il numero di nuove armate da assegnare
```

```
    select count(stato)
                    from Statotabellone
        where Statotabellone.giocatore = var_giocatore and Statotabellone.partita = var_partita
        into var_numstati;

            set var_nuovearmate = var_numstati/3;
            set var_nuovearmate = ceiling(var_nuovearmate);

            update Utente set armatedaposizionare = (armatedaposizionare + var_nuovearmate)
                        where nomeutente = var_giocatore;

            commit;
END$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure gioca_turno
-- -----------------------------------------------------

USE `mydb`;
DROP procedure IF EXISTS `mydb`.`gioca_turno`;

DELIMITER $$
```

```
USE `mydb`$$

CREATE PROCEDURE `gioca_turno` (in var_giocatore VARCHAR(45))

BEGIN

	declare var_partita INT;


	select restituisci_partita_incorso(var_giocatore) into var_partita;


	insert into Turno (partita, giocatore, inizioturno)

		values(var_partita, var_giocatore, now());

END$$


DELIMITER ;


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

-- procedure aggiungi_moderatore

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


USE `mydb`;

DROP procedure IF EXISTS `mydb`.`aggiungi_moderatore`;


DELIMITER $$

USE `mydb`$$

CREATE PROCEDURE `aggiungi_moderatore` (in var_nomeutente VARCHAR(45), in var_password VARCHAR(45))

BEGIN
```

```
        insert into Utente (nomeutente, password, tipo) VALUES(var_nomeutente, md5(var_password), 'moderatore');
END$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure effettua_attacco
-- -----------------------------------------------------

USE `mydb`;
DROP procedure IF EXISTS `mydb`.`effettua_attacco`;

DELIMITER $$
USE `mydb`$$
CREATE PROCEDURE `effettua_attacco` (in var_attaccante VARCHAR(45), in var_numarmateatt INT,
in var_statoatt VARCHAR(25), in var_statodif VARCHAR(25))
BEGIN
        declare var_partita INT;
        declare var_possiede INT;
    declare var_confinanti INT;
        declare var_armateposseduteatt INT;
    declare var_difensore VARCHAR(45);
    declare var_armatepossedutedif INT;
```

```
declare var_numarmatedif INT;
declare var_numdadiatt INT;
declare var_numdadidif INT;
declare var_idturno INT;

    declare dado1_att INT;
    declare dado2_att INT default null;
    declare dado3_att INT default null;
declare dado1_dif INT;
    declare dado2_dif INT default null;
    declare dado3_dif INT default null;

    declare armate_perse_att INT default 0;
declare armate_perse_dif INT default 0;

    declare var_statopartita ENUM('inattesa', 'incorso', 'terminata');
declare var_tipoazione ENUM('posizionamento', 'attacco', 'spostamento', 'nonsvolta');
    declare var_statiposseduti INT;

    declare exit handler for sqlexception
    begin
        rollback;  -- rollback any changes made in the transaction
        resignal;  -- raise again the sql exception to the caller
    end;
```

```
select restituisci_partita(var_attaccante) into var_partita;

    -- controllo se la partita e' stata avviata o e' ancora in attesa o e' gia' terminata
    select statopartita
        from Partita
        where idpartita = var_partita
        into var_statopartita;

    if var_statopartita = 'inattesa' then
        signal sqlstate '45000' set message_text = "Attendere che la partita inizi";
    end if;

    if var_statopartita = 'terminata' then
        signal sqlstate '45001' set message_text = "La partita e' terminata";
    end if;

    -- controllo che il giocatore non abbia perso
    select count(*)
        from Statotabellone
        where giocatore = var_attaccante and partita = var_partita
        into var_statiposseduti;

    if var_statiposseduti = 0 then
```

```
            signal sqlstate '45002' set message_text = "Hai perso la partita!";
        end if;

        -- controllo che non sia già stata effettuata un'azione nello stesso turno
    select idturno
            from Turno
            where partita = var_partita and Turno.giocatore = var_attaccante
        order by idturno desc limit 1
        into var_idturno;

    select tipoazione
            from Turno
        where idturno = var_idturno
        into var_tipoazione;

    if var_tipoazione is not null then
            signal sqlstate '45003' set message_text = "Hai gia' giocato il turno";
        end if;

        -- controllo se lo statoatt appartiene all'attaccante
        select count(*)
            from Statotabellone
        where giocatore = var_attaccante and partita = var_partita and stato = upper(var_statoatt)
            into var_possiede;
```

```
    if var_possiede <> 1 then
            signal sqlstate '45004' set message_text = "Lo statoatt non appartiene all'attaccante";
        end if;

        -- controllo se statoatt e statodif confinano e se appartengono a giocatori diversi
        select count(*)
            from Confina
      where stato = upper(var_statoatt) and confinante = upper(var_statodif)
      into var_confinanti;

        if var_confinanti <> 1 then
            signal sqlstate '45005' set message_text = "Stato attaccante e stato attaccato non confinano.";
        end if;

        select count(*)
            from Statotabellone
      where giocatore = var_attaccante and partita = var_partita and stato = upper(var_statodif)
            into var_possiede;

    if var_possiede = 1 then
            signal sqlstate '45006' set message_text = "Lo stato attaccato appartiene già all'attaccante";
        end if;
```

```
-- controllo che l'attaccante possieda il numero di armate che vuole schierare
-- e che questo numero sia al massimo pari a 4 (cioè 3 dadi)
-- e almeno pari a 2 (cioè un dado)

if var_numarmateatt > 4 then
        signal sqlstate '45007' set message_text = "Non e' possibile schierare piu' di 4 armate";
    end if;
    if var_numarmateatt < 2 then
        signal sqlstate '45008' set message_text = "Non e' possibile schierare meno di 2 armate";
    end if;

select numarmate
        from Statotabellone
  where stato = upper(var_statoatt) and giocatore = var_attaccante and partita = var_partita
  into var_armateposseduteatt;

if var_numarmateatt > var_armateposseduteatt then
        signal sqlstate '45009' set message_text = "L'attaccante non possiede il numero di armate che ha chiesto di schierare";
    end if;

-- conoscere il difensore
```

```
select giocatore
        from Statotabellone
    where stato = upper(var_statodif) and partita = var_partita
    into var_difensore;

-- controllo le armate possedute dal difensore e, se ne ha di piu' rispetto al numarmateatt
-- ne prendo solo in numero pari ad numarmateatt, altrimenti se ne ha di meno le prendo tutte
select numarmate
        from Statotabellone
    where stato = upper(var_statodif) and Statotabellone.giocatore = var_difensore and partita = var_partita
    into var_armatepossedutedif;

    if var_armatepossedutedif > var_numarmateatt then
        set var_numarmatedif = var_numarmateatt;
    else
        set var_numarmatedif = var_armatepossedutedif;
    end if;

-- calcolo il numero di dadi da lanciare
    set var_numdadiatt = var_numarmateatt - 1;
set var_numdadidif = var_numarmatedif - 1;

    -- lancio dadi
    if var_numdadiatt = 3 then
```

```
            set dado1_att = ceiling(rand()*6);
            set dado2_att = ceiling(rand()*6);
            set dado3_att = ceiling(rand()*6);

    elseif var_numdadiatt = 2 then
            set dado1_att = ceiling(rand()*6);
            set dado2_att = ceiling(rand()*6);

        elseif var_numdadiatt = 1 then
            set dado1_att = ceiling(rand()*6);
        end if;

        if var_numdadidif = 3 then
            set dado1_dif = ceiling(rand()*6);
            set dado2_dif = ceiling(rand()*6);
            set dado3_dif = ceiling(rand()*6);

    elseif var_numdadidif = 2 then
            set dado1_dif = ceiling(rand()*6);
            set dado2_dif = ceiling(rand()*6);

        elseif var_numdadidif = 1 then
            set dado1_dif = ceiling(rand()*6);
        end if;
```

```
-- controllo vittoria lancio dadi

call controlla_vittoria_lancio_dadi(dado1_att, dado2_att, dado3_att, dado1_dif, dado2_dif, dado3_dif, armate_perse_att, armate_perse_dif);


if (var_armateposseduteatt - armate_perse_att > 0 and armate_perse_att <> 0) then

        update Statotabellone set numarmate = (numarmate - armate_perse_att)

            where giocatore = var_attaccante and partita = var_partita and stato = upper(var_statoatt);

elseif (var_armateposseduteatt - armate_perse_att = 0) then

        update Statotabellone set Statotabellone.giocatore = var_difensore, numarmate = var_numarmatedif - armate_perse_dif

            where giocatore = var_attaccante and partita = var_partita and stato = upper(var_statoatt);

end if;


    if (var_armatepossedutedif - armate_perse_dif > 0 and armate_perse_dif <> 0) then

        update Statotabellone set numarmate = (numarmate - armate_perse_dif)

            where Statotabellone.giocatore = var_difensore and partita = var_partita and stato = upper(var_statodif);

    elseif (var_armatepossedutedif - armate_perse_dif = 0) then

        update Statotabellone set giocatore = var_attaccante, numarmate = var_numarmateatt - armate_perse_att

            where Statotabellone.giocatore = var_difensore and partita = var_partita and stato = upper(var_statodif);

end if;
```

```
        update Turno set tipoazione = 'attacco', tempoazione = now()
                where idturno = var_idturno and Turno.giocatore = var_attaccante
    and partita = var_partita and tempoazione is null;

-- se gli stati appartengono tutti allo stesso giocatore (attaccante o difensore), la partita termina
select count(*)
                from Statotabellone
                where giocatore = var_attaccante
                into var_statiposseduti;

        if var_statiposseduti = 42 then
    update Partita set vincitore = var_attaccante, statopartita = 'terminata'
                        where idpartita = var_partita;
                signal sqlstate '45000' set message_text = "L'attaccante ha vinto la partita!";
end if;

select count(*)
                from Statotabellone
                where giocatore = var_difensore
                into var_statiposseduti;

if var_statiposseduti = 42 then
    update Partita set vincitore = var_difensore, statopartita = 'terminata'
```

```
                    where idpartita = var_partita;
                signal sqlstate '45001' set message_text = "Il difensore ha vinto la partita!";
    end if;

    call turno_successivo(var_attaccante, var_partita);
END$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure sposta_armate
-- -----------------------------------------------------

USE `mydb`;
DROP procedure IF EXISTS `mydb`.`sposta_armate`;

DELIMITER $$
USE `mydb`$$
CREATE PROCEDURE `sposta_armate` (in var_giocatore VARCHAR(45), in var_numarmate INT,
in var_statopart VARCHAR(25), in var_statodest VARCHAR(25))
BEGIN
        declare var_possiede INT;
    declare var_confinanti INT;
    declare var_partita INT;
    declare var_armatetot INT;
```

```
declare var_idturno INT;
	declare var_statopartita ENUM('inattesa', 'incorso', 'terminata');
declare var_tipoazione ENUM('posizionamento', 'attacco', 'spostamento', 'nonsvolta');
declare var_statiposseduti INT;

declare exit handler for sqlexception
begin
  rollback;  -- rollback any changes made in the transaction
  resignal;  -- raise again the sql exception to the caller
end;

select restituisci_partita(var_giocatore) into var_partita;

	-- controllo se la partita e' stata avviata o e' ancora in attesa o e' gia' terminata
	select statopartita
		from Partita
		where idpartita = var_partita
		into var_statopartita;

	if var_statopartita = 'inattesa' then
		signal sqlstate '45000' set message_text = "Attendere che la partita inizi";
	end if;

	if var_statopartita = 'terminata' then
```

```
            signal sqlstate '45001' set message_text = "La partita e' terminata";
        end if;

        -- controllo che il giocatore non abbia perso
        select count(*)
            from Statotabellone
            where giocatore = var_giocatore and partita = var_partita
            into var_statiposseduti;

        if var_statiposseduti = 0 then
            signal sqlstate '45002' set message_text = "Hai perso la partita!";
        end if;

        -- controllo che non sia già stata effettuata un'azione nello stesso turno
        select idturno
            from Turno
            where partita = var_partita and Turno.giocatore = var_giocatore
    order by idturno desc limit 1
    into var_idturno;

select tipoazione
            from Turno
    where idturno = var_idturno
    into var_tipoazione;
```

```
if var_tipoazione is not null then
        signal sqlstate '45003' set message_text = "Hai gia' giocato il turno";
    end if;

    -- controllo se lo statopart e lo statodest appartengono al giocatore
    select count(*)
        from Statotabellone
  where Statotabellone.giocatore = var_giocatore and Statotabellone.partita = var_partita and Statotabellone.stato = upper(var_statopart)
        into var_possiede;

if var_possiede <> 1 then
        signal sqlstate '45004' set message_text = "Lo stato di partenza non appartiene al giocatore";
    end if;

select count(*)
        from Statotabellone
  where Statotabellone.giocatore = var_giocatore and Statotabellone.partita = var_partita and Statotabellone.stato = upper(var_statodest)
        into var_possiede;

if var_possiede <> 1 then
```

```
            signal sqlstate '45005' set message_text = "Lo stato di destinazione non appartiene al giocatore";
        end if;

    -- controllo che statopart e statodest siano confinanti
    select count(*)
            from Confina
        where Confina.stato = upper(var_statopart) and confinante = upper(var_statodest)
        into var_confinanti;

        if var_confinanti <> 1 then
            signal sqlstate '45006' set message_text = "Stato di partenza e stato di destinazione non confinano.";
        end if;

    -- controllo che sullo statopart rimanga almeno un'armata

    select numarmate
            from Statotabellone
            where stato = upper(var_statopart) and Statotabellone.partita = var_partita
        into var_armatetot;

        if var_armatetot - var_numarmate < 1 then
            signal sqlstate '45007' set message_text = "Sullo stato di partenza deve rimanere almeno un'armata";
```

```
        end if;

        update Statotabellone set numarmate = (numarmate + var_numarmate)
                        where Statotabellone.giocatore = var_giocatore and partita = var_partita and stato=var_statodest;

    update Statotabellone set numarmate = (numarmate - var_numarmate)
                        where Statotabellone.giocatore = var_giocatore and partita = var_partita and stato = upper(var_statopart);

        update Turno set tipoazione = 'spostamento', tempoazione = now()
                where idturno = var_idturno and tempoazione is null;

        call turno_successivo(var_giocatore, var_partita);
END$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure controlla_vittoria_lancio_dadi
-- -----------------------------------------------------

USE `mydb`;
DROP procedure IF EXISTS `mydb`.`controlla_vittoria_lancio_dadi`;
```

```
DELIMITER $$

USE `mydb`$$

CREATE PROCEDURE `controlla_vittoria_lancio_dadi` (in dado1_att INT, in dado2_att INT, in dado3_att INT, in dado1_dif INT, in dado2_dif INT, in dado3_dif INT, out armate_perse_att INT, out armate_perse_dif INT)

BEGIN

        -- Variabili che rappresentano il dado più alto, quello medio e quello più basso rispettivamente dell'attaccante e del difensore

        declare max_dado_att INT;


        declare med_dado_att INT;

        declare min_dado_att INT;

        declare max_dado_dif INT;

        declare med_dado_dif INT;

        declare min_dado_dif INT;


    set transaction isolation level read uncommitted;

    start transaction;


    -- In base alla presenza o meno dei valori dei dadi, riordino i loro valori in modo che siano nelle variabili giuste

        -- se c'è un solo dado diverso da null

        if (dado2_att is null) and (dado3_att is null) then

                set max_dado_att = dado1_att;

    -- se ci sono due dadi diversi da null

        elseif dado3_att is null then
```

```
            if (dado1_att >= dado2_att) then
                    set max_dado_att = dado1_att;
                    set med_dado_att = dado2_att;
            else
                    set max_dado_att = dado2_att;
                    set med_dado_att = dado1_att;
            end if;
    -- Tutti e tre i dadi non sono null
        else
            if((dado1_att >= dado2_att) and (dado1_att >= dado3_att)) then
                    set max_dado_att = dado1_att;
                    if(dado2_att > dado3_att) then
                            set med_dado_att = dado2_att;
                            set min_dado_att = dado3_att;
                    else
                            set med_dado_att = dado3_att;
                            set min_dado_att = dado2_att;
                    end if;
            elseif (dado2_att >= dado1_att) and (dado2_att >= dado3_att) then
                    set max_dado_att = dado2_att;
                    if(dado1_att > dado3_att) then
                            set med_dado_att = dado1_att;
                            set min_dado_att = dado3_att;
                    else
```

```
                    set med_dado_att = dado3_att;
                    set min_dado_att = dado1_att;
                end if;
            else
                set max_dado_att = dado3_att;
                if(dado1_att > dado2_att) then
                    set med_dado_att = dado1_att;
                    set min_dado_att = dado2_att;
                else
                    set med_dado_att = dado2_att;
                    set min_dado_att = dado1_att;
                end if;
            end if;
        end if;

        -- ripeto la stessa cosa per i dadi del difensore
        -- se c'è un solo dado diverso da null
        if (dado2_dif is null) and (dado3_dif is null) then
            set max_dado_dif = dado1_dif;
    -- se ci sono due dadi diversi da null
        elseif dado3_dif is null then
            if (dado1_dif >= dado2_dif) then
                set max_dado_dif = dado1_dif;
                set med_dado_dif = dado2_dif;
```

```
            else
                set max_dado_dif = dado2_dif;
                set med_dado_dif = dado1_dif;
            end if;
    -- Tutti e tre i dadi non sono null
        else
            if(dado1_dif >= dado2_dif) and (dado1_dif >= dado3_dif) then
                set max_dado_dif = dado1_dif;
                if(dado2_dif > dado3_dif) then
                    set med_dado_dif = dado2_dif;
                    set min_dado_dif = dado3_dif;
                else
                    set med_dado_dif = dado3_dif;
                    set min_dado_dif = dado2_dif;
                end if;
            elseif(dado2_dif >= dado1_dif) and (dado2_dif >= dado3_dif) then
                set max_dado_dif = dado2_dif;
                if(dado1_dif > dado3_dif) then
                    set med_dado_dif = dado1_dif;
                    set min_dado_dif = dado3_dif;
                else
                    set med_dado_dif = dado3_dif;
                    set min_dado_dif = dado1_dif;
                end if;
```

```
        else
                set max_dado_dif = dado3_dif;
                if(dado1_dif > dado2_dif) then
                        set med_dado_dif = dado1_dif;
                        set min_dado_dif = dado2_dif;
                else
                        set med_dado_dif = dado2_dif;
                        set min_dado_dif = dado1_dif;
                end if;
        end if;
end if;

-- Controllo dei valori MAX MED e MIN; il valore 1 indica che si è persa un'armata

if(max_dado_dif >= max_dado_att) then
        set armate_perse_att = 1; -- perde un'armata l'attaccante
        set armate_perse_dif = 0;
else
        set armate_perse_dif = 1; -- perde un'armata il difensore
        set armate_perse_att = 0;
end if;
if((med_dado_att  is not null) and (med_dado_dif is not null)) then
        if(med_dado_dif >= med_dado_att) then
                set armate_perse_att = armate_perse_att + 1; -- perde un'armata l'attaccante
```

```
                else
                        set armate_perse_dif = armate_perse_dif + 1; -- perde un'armata il difensore
                end if;
                if((min_dado_att is not null) and (min_dado_dif is not null)) then
                        if(min_dado_dif >= min_dado_att) then
                                set armate_perse_att = armate_perse_att + 1; -- perde un'armata l'attaccante
                        else
                                set armate_perse_dif = armate_perse_dif + 1; -- perde un'armata il difensore
                        end if;
                end if;
        end if;
 commit;
END$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure visualizza_stanze_libere
-- -----------------------------------------------------

USE `mydb`;
DROP procedure IF EXISTS `mydb`.`visualizza_stanze_libere`;
```

```
DELIMITER $$
USE `mydb`$$
CREATE PROCEDURE `visualizza_stanze_libere` ()
BEGIN
	set transaction read only;
	set transaction isolation level read committed;
	start transaction;
	select nomestanza
		from Stanzadigioco
	where nomestanza not in (select nomestanza
		from Stanzadigioco join Partita on Stanzadigioco.nomestanza = Partita.stanzadigioco
	where statopartita = 'incorso' or statopartita = 'inattesa');
	commit;
END$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure visualizza_partite_in_attesa
-- -----------------------------------------------------

USE `mydb`;
DROP procedure IF EXISTS `mydb`.`visualizza_partite_in_attesa`;
```

```
DELIMITER $$
USE `mydb`$$
CREATE PROCEDURE `visualizza_partite_in_attesa` ()
BEGIN
        set transaction read only;
        set transaction isolation level read committed;
        start transaction;
        select idpartita, stanzadigioco,  count(giocatore) as numgiocatori
          from Partita join Gioca on Partita.idpartita = Gioca.partita
    where statopartita = 'inattesa'
    group by idpartita;
        commit;
END$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure crea_partita
-- -----------------------------------------------------

USE `mydb`;
DROP procedure IF EXISTS `mydb`.`crea_partita`;

DELIMITER $$
```

```
USE `mydb`$$

CREATE PROCEDURE `crea_partita` (in var_stanzadigioco VARCHAR(15), in var_giocatore VARCHAR(45), out var_idpartita INT)

BEGIN

	declare var_numpartite INT;


	declare exit handler for sqlexception

	begin

		rollback;  -- rollback any changes made in the transaction

		resignal;  -- raise again the sql exception to the caller

  end;


  set transaction isolation level read committed;

  start transaction;


	-- controllo che nella stessa stanza non ci siano altre partite inattesa o incorso

  select count(*)

		from Partita

    where stanzadigioco = var_stanzadigioco and (statopartita = 'incorso' or statopartita = 'inattesa')

    into var_numpartite;


	if var_numpartite <> 0 then

		signal sqlstate '45000' set message_text = "In questa stanza esiste gia' una partita inattesa o incorso";

	end if;
```

```
        insert into Partita (statopartita, stanzadigioco) values ('inattesa', var_stanzadigioco);

        select idpartita
                from Partita
    where statopartita = 'inattesa' and stanzadigioco = var_stanzadigioco
    into var_idpartita;

        insert into Gioca (giocatore, partita, numturno) values (var_giocatore, var_idpartita, 1);
        update Utente set armatedaposizionare = 0 where nomeutente = var_giocatore;
        commit;
END$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure turno_successivo
-- -----------------------------------------------------

USE `mydb`;
DROP procedure IF EXISTS `mydb`.`turno_successivo`;

DELIMITER $$
USE `mydb`$$
```

```
CREATE PROCEDURE `turno_successivo` (in var_giocatore VARCHAR(45), in var_partita INT)
BEGIN
	declare var_numprossimoturno INT;
    declare var_prossimogiocatore VARCHAR(45);
    declare var_numgiocatori INT;
    declare var_numturno INT;
    declare var_tipoazione ENUM('spostamento', 'attacco', 'posizionamento', 'nonsvolta');

	select count(*)
		from Gioca join Partita on Gioca.partita = Partita.idpartita
		where idpartita = var_partita
	into var_numgiocatori;

	select numturno
		from Gioca
		where Gioca.giocatore = var_giocatore and Gioca.partita = var_partita
		into var_numturno;

	set var_numprossimoturno = (var_numturno % var_numgiocatori) + 1;

	select giocatore
		from Gioca
		where numturno = var_numprossimoturno and partita = var_partita
	into var_prossimogiocatore;
```

```
    select tipoazione
                    from Turno
        where giocatore = var_giocatore and partita = var_partita
        order by idturno desc limit 1
        into var_tipoazione;

    if var_tipoazione <> 'nonsvolta' then
                    call assegna_nuove_armate(var_giocatore);
            end if;

            call gioca_turno(var_prossimogiocatore);
END$$
```

Appendice: Implementazione

Codice SQL per instanziare il database

```
SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0;

SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS,
FOREIGN_KEY_CHECKS=0;

SET @OLD_SQL_MODE=@@SQL_MODE, SQL_MODE='ONLY_FULL_GROUP_BY,
STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZERO_DATE,ERROR_FOR_DIVISION_
BY_ZERO,NO_ENGINE_SUBSTITUTION';


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

-- Schema mydb

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

DROP SCHEMA IF EXISTS `mydb` ;


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

-- Schema mydb

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

CREATE SCHEMA IF NOT EXISTS `mydb` DEFAULT CHARACTER SET utf8mb4 ;

USE `mydb` ;


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

-- Table `mydb`.`Utente`

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

DROP TABLE IF EXISTS `mydb`.`Utente` ;
```

```
CREATE TABLE IF NOT EXISTS `mydb`.`Utente` (
  `nomeutente` VARCHAR(45) NOT NULL,
  `password` VARCHAR(45) NOT NULL,
  `tipo` ENUM('giocatore', 'moderatore') NOT NULL,
  `armatedaposizionare` INT UNSIGNED NULL,
  PRIMARY KEY (`nomeutente`))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `mydb`.`Stanzadigioco`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `mydb`.`Stanzadigioco` ;

CREATE TABLE IF NOT EXISTS `mydb`.`Stanzadigioco` (
  `nomestanza` VARCHAR(15) NOT NULL,
  PRIMARY KEY (`nomestanza`))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `mydb`.`Partita`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `mydb`.`Partita` ;
```

```
CREATE TABLE IF NOT EXISTS `mydb`.`Partita` (
  `idpartita` INT UNSIGNED NOT NULL AUTO_INCREMENT,
  `statopartita` ENUM('inattesa', 'incorso', 'terminata') NOT NULL,
  `stanzadigioco` VARCHAR(15) NOT NULL,
  `iniziocountdown` DATETIME NULL,
  `vincitore` VARCHAR(45) NULL,
  PRIMARY KEY (`idpartita`),
  CONSTRAINT `fk_partita_Stanzadigioco1`
    FOREIGN KEY (`stanzadigioco`)
    REFERENCES `mydb`.`Stanzadigioco` (`nomestanza`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;

CREATE INDEX `fk_partita_Stanzadigioco1_idx` ON `mydb`.`Partita` (`stanzadigioco` ASC) VISIBLE;


-- -----------------------------------------------------
-- Table `mydb`.`Gioca`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `mydb`.`Gioca` ;

CREATE TABLE IF NOT EXISTS `mydb`.`Gioca` (
```

```
  `numturno` INT UNSIGNED NOT NULL,
  `giocatore` VARCHAR(45) NOT NULL,
  `partita` INT UNSIGNED NOT NULL,
  PRIMARY KEY (`giocatore`, `partita`),
  CONSTRAINT `fk_gioca_Utente`
    FOREIGN KEY (`giocatore`)
    REFERENCES `mydb`.`Utente` (`nomeutente`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Gioca_Partita1`
    FOREIGN KEY (`partita`)
    REFERENCES `mydb`.`Partita` (`idpartita`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


CREATE INDEX `fk_Gioca_Partita1_idx` ON `mydb`.`Gioca` (`partita` ASC) VISIBLE;



-- -----------------------------------------------------
-- Table `mydb`.`Turno`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `mydb`.`Turno` ;
```

```
CREATE TABLE IF NOT EXISTS `mydb`.`Turno` (
  `idturno` INT UNSIGNED NOT NULL AUTO_INCREMENT,
  `tempoazione` DATETIME NULL,
  `tipoazione` ENUM('posizionamento', 'attacco', 'spostamento', 'nonsvolta') NULL,
  `partita` INT UNSIGNED NOT NULL,
  `inizioturno` DATETIME NOT NULL,
  `giocatore` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`idturno`, `partita`, `giocatore`),
  CONSTRAINT `fk_Turno_partita1`
    FOREIGN KEY (`partita`)
    REFERENCES `mydb`.`Partita` (`idpartita`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Turno_Utente1`
    FOREIGN KEY (`giocatore`)
    REFERENCES `mydb`.`Utente` (`nomeutente`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;

CREATE INDEX `fk_Turno_partita1_idx` ON `mydb`.`Turno` (`partita` ASC) VISIBLE;

CREATE INDEX `fk_Turno_Utente1_idx` ON `mydb`.`Turno` (`giocatore` ASC) VISIBLE;
```

```
-- -----------------------------------------------------
-- Table `mydb`.`Stato`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `mydb`.`Stato` ;

CREATE TABLE IF NOT EXISTS `mydb`.`Stato` (
  `nomestato` VARCHAR(25) NOT NULL,
  PRIMARY KEY (`nomestato`))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `mydb`.`Statotabellone`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `mydb`.`Statotabellone` ;

CREATE TABLE IF NOT EXISTS `mydb`.`Statotabellone` (
  `numarmate` INT UNSIGNED NOT NULL DEFAULT 3,
  `giocatore` VARCHAR(45) NOT NULL,
  `partita` INT UNSIGNED NOT NULL,
  `stato` VARCHAR(25) NOT NULL,
  PRIMARY KEY (`giocatore`, `partita`, `stato`),
  CONSTRAINT `fk_Statotabellone_Utente1`
```

```
  FOREIGN KEY (`giocatore`)
  REFERENCES `mydb`.`Utente` (`nomeutente`)
  ON DELETE NO ACTION
  ON UPDATE NO ACTION,
 CONSTRAINT `fk_Statotabellone_partita1`
  FOREIGN KEY (`partita`)
  REFERENCES `mydb`.`Partita` (`idpartita`)
  ON DELETE NO ACTION
  ON UPDATE NO ACTION,
 CONSTRAINT `fk_Statotabellone_Stato1`
  FOREIGN KEY (`stato`)
  REFERENCES `mydb`.`Stato` (`nomestato`)
  ON DELETE NO ACTION
  ON UPDATE NO ACTION)
ENGINE = InnoDB;

CREATE INDEX `fk_Statotabellone_partita1_idx` ON `mydb`.`Statotabellone` (`partita` ASC) VISIBLE;

CREATE INDEX `fk_Statotabellone_Stato1_idx` ON `mydb`.`Statotabellone` (`stato` ASC) VISIBLE;

-- -----------------------------------------------------
-- Table `mydb`.`Confina`
```

```
-- -----------------------------------------------------
DROP TABLE IF EXISTS `mydb`.`Confina` ;

CREATE TABLE IF NOT EXISTS `mydb`.`Confina` (
  `stato` VARCHAR(25) NOT NULL,
  `confinante` VARCHAR(25) NOT NULL,
  PRIMARY KEY (`stato`, `confinante`),
  CONSTRAINT `fk_Confina_Stato1`
    FOREIGN KEY (`stato`)
    REFERENCES `mydb`.`Stato` (`nomestato`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Confina_Stato2`
    FOREIGN KEY (`confinante`)
    REFERENCES `mydb`.`Stato` (`nomestato`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;

CREATE INDEX `fk_Confina_Stato2_idx` ON `mydb`.`Confina` (`confinante` ASC) VISIBLE;

USE `mydb` ;
DELIMITER ;
SET SQL_MODE = '';
```

```
DROP USER IF EXISTS giocatore;

SET
SQL_MODE='ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO
_ZERO_DATE,ERROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION';

CREATE USER 'giocatore' IDENTIFIED BY 'giocatore';


GRANT EXECUTE ON procedure `mydb`.`partecipa_partita` TO 'giocatore';

GRANT EXECUTE ON procedure `mydb`.`storico_partite` TO 'giocatore';

GRANT EXECUTE ON procedure `mydb`.`posiziona_armate` TO 'giocatore';

GRANT EXECUTE ON procedure `mydb`.`sposta_armate` TO 'giocatore';

GRANT EXECUTE ON procedure `mydb`.`effettua_attacco` TO 'giocatore';

GRANT EXECUTE ON procedure `mydb`.`visualizza_stanze_libere` TO 'giocatore';

GRANT EXECUTE ON procedure `mydb`.`visualizza_partite_in_attesa` TO 'giocatore';

GRANT EXECUTE ON procedure `mydb`.`crea_partita` TO 'giocatore';

GRANT EXECUTE ON procedure `mydb`.`stato_di_gioco` TO 'giocatore';

SET SQL_MODE = '';

DROP USER IF EXISTS moderatore;

SET
SQL_MODE='ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO
_ZERO_DATE,ERROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION';

CREATE USER 'moderatore' IDENTIFIED BY 'moderatore';


GRANT EXECUTE ON procedure `mydb`.`crea_stanzadigioco` TO 'moderatore';

GRANT EXECUTE ON procedure `mydb`.`report_moderatore` TO 'moderatore';

GRANT EXECUTE ON procedure `mydb`.`aggiungi_moderatore` TO 'moderatore';
```

```
SET SQL_MODE = '';

DROP USER IF EXISTS login;

SET
SQL_MODE='ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO
_ZERO_DATE,ERROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION';

CREATE USER 'login' IDENTIFIED BY 'login';


GRANT EXECUTE ON procedure `mydb`.`login` TO 'login';

GRANT EXECUTE ON procedure `mydb`.`aggiungi_giocatore` TO 'login';


SET SQL_MODE=@OLD_SQL_MODE;

SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS;

SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS;


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

-- Data for table `mydb`.`Utente`

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

START TRANSACTION;

USE `mydb`;

INSERT INTO `mydb`.`Utente` (`nomeutente`, `password`, `tipo`, `armatedaposizionare`) VALUES
('luca', '6e6bc4e49dd477ebc98ef4046c067b5f', 'moderatore', NULL);

INSERT INTO `mydb`.`Utente` (`nomeutente`, `password`, `tipo`, `armatedaposizionare`) VALUES
('dominique', '6e6bc4e49dd477ebc98ef4046c067b5f', 'giocatore', NULL);

INSERT INTO `mydb`.`Utente` (`nomeutente`, `password`, `tipo`, `armatedaposizionare`) VALUES
('sissi', '6e6bc4e49dd477ebc98ef4046c067b5f', 'giocatore', NULL);
```

```
INSERT INTO `mydb`.`Utente` (`nomeutente`, `password`, `tipo`, `armatedaposizionare`) VALUES ('elis', '6e6bc4e49dd477ebc98ef4046c067b5f', 'giocatore', NULL);

INSERT INTO `mydb`.`Utente` (`nomeutente`, `password`, `tipo`, `armatedaposizionare`) VALUES ('leti', '6e6bc4e49dd477ebc98ef4046c067b5f', 'giocatore', NULL);

INSERT INTO `mydb`.`Utente` (`nomeutente`, `password`, `tipo`, `armatedaposizionare`) VALUES ('fra', '6e6bc4e49dd477ebc98ef4046c067b5f', 'giocatore', NULL);

INSERT INTO `mydb`.`Utente` (`nomeutente`, `password`, `tipo`, `armatedaposizionare`) VALUES ('mimmo', '6e6bc4e49dd477ebc98ef4046c067b5f', 'giocatore', NULL);


COMMIT;



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

-- Data for table `mydb`.`Stanzadigioco`

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

START TRANSACTION;

USE `mydb`;

INSERT INTO `mydb`.`Stanzadigioco` (`nomestanza`) VALUES ('verde');

INSERT INTO `mydb`.`Stanzadigioco` (`nomestanza`) VALUES ('blu');


COMMIT;



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

-- Data for table `mydb`.`Stato`
```

```
-- -----------------------------------------------------
START TRANSACTION;
USE `mydb`;
INSERT INTO `mydb`.`Stato` (`nomestato`) VALUES ('ALASKA');
INSERT INTO `mydb`.`Stato` (`nomestato`) VALUES ('TERRITORI DEL NORD OVEST');
INSERT INTO `mydb`.`Stato` (`nomestato`) VALUES ('GROENLANDIA');
INSERT INTO `mydb`.`Stato` (`nomestato`) VALUES ('ALBERTA');
INSERT INTO `mydb`.`Stato` (`nomestato`) VALUES ('ONTARIO');
INSERT INTO `mydb`.`Stato` (`nomestato`) VALUES ('QUEBEC');
INSERT INTO `mydb`.`Stato` (`nomestato`) VALUES ('STATI UNITI OCCIDENTALI');
INSERT INTO `mydb`.`Stato` (`nomestato`) VALUES ('STATI UNITI ORIENTALI');
INSERT INTO `mydb`.`Stato` (`nomestato`) VALUES ('AMERICA CENTRALE');
INSERT INTO `mydb`.`Stato` (`nomestato`) VALUES ('VENEZUELA');
INSERT INTO `mydb`.`Stato` (`nomestato`) VALUES ('BRASILE');
INSERT INTO `mydb`.`Stato` (`nomestato`) VALUES ('PERU');
INSERT INTO `mydb`.`Stato` (`nomestato`) VALUES ('ARGENTINA');
INSERT INTO `mydb`.`Stato` (`nomestato`) VALUES ('ISLANDA');
INSERT INTO `mydb`.`Stato` (`nomestato`) VALUES ('GRAN BRETAGNA');
INSERT INTO `mydb`.`Stato` (`nomestato`) VALUES ('SCANDINAVIA');
INSERT INTO `mydb`.`Stato` (`nomestato`) VALUES ('EUROPA OCCIDENTALE');
INSERT INTO `mydb`.`Stato` (`nomestato`) VALUES ('EUROPA SETTENTRIONALE');
INSERT INTO `mydb`.`Stato` (`nomestato`) VALUES ('UCRAINA');
INSERT INTO `mydb`.`Stato` (`nomestato`) VALUES ('EUROPA MERIDIONALE');
INSERT INTO `mydb`.`Stato` (`nomestato`) VALUES ('AFRICA DEL NORD');
```

```
INSERT INTO `mydb`.`Stato` (`nomestato`) VALUES ('EGITTO');

INSERT INTO `mydb`.`Stato` (`nomestato`) VALUES ('CONGO');

INSERT INTO `mydb`.`Stato` (`nomestato`) VALUES ('AFRICA ORIENTALE');

INSERT INTO `mydb`.`Stato` (`nomestato`) VALUES ('AFRICA DEL SUD');

INSERT INTO `mydb`.`Stato` (`nomestato`) VALUES ('MADAGASCAR');

INSERT INTO `mydb`.`Stato` (`nomestato`) VALUES ('URALI');

INSERT INTO `mydb`.`Stato` (`nomestato`) VALUES ('SIBERIA');

INSERT INTO `mydb`.`Stato` (`nomestato`) VALUES ('JACUZIA');

INSERT INTO `mydb`.`Stato` (`nomestato`) VALUES ('KAMCHATKA');

INSERT INTO `mydb`.`Stato` (`nomestato`) VALUES ('CITA');

INSERT INTO `mydb`.`Stato` (`nomestato`) VALUES ('MEDIO ORIENTE');

INSERT INTO `mydb`.`Stato` (`nomestato`) VALUES ('INDIA');

INSERT INTO `mydb`.`Stato` (`nomestato`) VALUES ('SIAM');

INSERT INTO `mydb`.`Stato` (`nomestato`) VALUES ('INDONESIA');

INSERT INTO `mydb`.`Stato` (`nomestato`) VALUES ('NUOVA GUINEA');

INSERT INTO `mydb`.`Stato` (`nomestato`) VALUES ('AUSTRALIA OCCIDENTALE');

INSERT INTO `mydb`.`Stato` (`nomestato`) VALUES ('AUSTRALIA ORIENTALE');

INSERT INTO `mydb`.`Stato` (`nomestato`) VALUES ('GIAPPONE');

INSERT INTO `mydb`.`Stato` (`nomestato`) VALUES ('CINA');

INSERT INTO `mydb`.`Stato` (`nomestato`) VALUES ('MONGOLIA');

INSERT INTO `mydb`.`Stato` (`nomestato`) VALUES ('AFGHANISTAN');


COMMIT;
```

```
-- -----------------------------------------------------

-- Data for table `mydb`.`Confina`

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

START TRANSACTION;

USE `mydb`;

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('ALASKA', 'TERRITORI DEL NORD OVEST');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('ALASKA', 'ALBERTA');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('ALASKA', 'KAMCHATKA');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('TERRITORI DEL NORD OVEST', 'ALASKA');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('TERRITORI DEL NORD OVEST', 'GROENLANDIA');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('TERRITORI DEL NORD OVEST', 'ALBERTA');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('TERRITORI DEL NORD OVEST', 'ONTARIO');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('GROENLANDIA', 'TERRITORI DEL NORD OVEST');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('GROENLANDIA', 'ONTARIO');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('GROENLANDIA', 'QUEBEC');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('GROENLANDIA', 'ISLANDA');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('ALBERTA', 'ALASKA');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('ALBERTA', 'TERRITORI DEL NORD OVEST');
```

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('ALBERTA', 'ONTARIO');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('ALBERTA', 'STATI UNITI OCCIDENTALI');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('ONTARIO', 'GROENLANDIA');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('ONTARIO', 'TERRITORI DEL NORD OVEST');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('ONTARIO', 'QUEBEC');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('ONTARIO', 'STATI UNITI OCCIDENTALI');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('ONTARIO', 'STATI UNITI ORIENTALI');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('ONTARIO', 'ALBERTA');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('QUEBEC', 'GROENLANDIA');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('QUEBEC', 'ONTARIO');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('QUEBEC', 'STATI UNITI ORIENTALI');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('STATI UNITI OCCIDENTALI', 'ALBERTA');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('STATI UNITI OCCIDENTALI', 'ONTARIO');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('STATI UNITI OCCIDENTALI', 'STATI UNITI ORIENTALI');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('STATI UNITI OCCIDENTALI', 'AMERICA CENTRALE');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('STATI UNITI ORIENTALI', 'STATI UNITI OCCIDENTALI');

```
INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('STATI UNITI ORIENTALI', 'ONTARIO');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('STATI UNITI ORIENTALI', 'QUEBEC');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('STATI UNITI ORIENTALI', 'AMERICA CENTRALE');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('AMERICA CENTRALE', 'STATI UNITI OCCIDENTALI');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('AMERICA CENTRALE', 'STATI UNITI ORIENTALI');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('AMERICA CENTRALE', 'VENEZUELA');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('VENEZUELA', 'AMERICA CENTRALE');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('VENEZUELA', 'BRASILE');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('VENEZUELA', 'PERU');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('BRASILE', 'VENEZUELA');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('BRASILE', 'PERU');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('BRASILE', 'ARGENTINA');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('BRASILE', 'AFRICA DEL NORD');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('PERU', 'VENEZUELA');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('PERU', 'BRASILE');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('PERU', 'ARGENTINA');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('ARGENTINA', 'PERU');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('ARGENTINA', 'BRASILE');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('ISLANDA', 'GROENLANDIA');
```

```
INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('ISLANDA', 'GRAN BRETAGNA');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('ISLANDA', 'SCANDINAVIA');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('GRAN BRETAGNA', 'ISLANDA');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('GRAN BRETAGNA', 'SCANDINAVIA');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('GRAN BRETAGNA', 'EUROPA OCCIDENTALE');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('GRAN BRETAGNA', 'EUROPA SETTENTRIONALE');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('SCANDINAVIA', 'ISLANDA');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('SCANDINAVIA', 'GRAN BRETAGNA');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('SCANDINAVIA', 'EUROPA SETTENTRIONALE');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('SCANDINAVIA', 'UCRAINA');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('EUROPA OCCIDENTALE', 'GRAN BRETAGNA');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('EUROPA OCCIDENTALE', 'EUROPA SETTENTRIONALE');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('EUROPA OCCIDENTALE', 'EUROPA MERIDIONALE');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('EUROPA OCCIDENTALE', 'AFRICA DEL NORD');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('EUROPA SETTENTRIONALE', 'GRAN BRETAGNA');
```

```
INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('EUROPA SETTENTRIONALE', 'SCANDINAVIA');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('EUROPA SETTENTRIONALE', 'EUROPA OCCIDENTALE');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('EUROPA SETTENTRIONALE', 'UCRAINA');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('EUROPA SETTENTRIONALE', 'EUROPA MERIDIONALE');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('UCRAINA', 'SCANDINAVIA');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('UCRAINA', 'EUROPA SETTENTRIONALE');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('UCRAINA', 'EUROPA MERIDIONALE');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('UCRAINA', 'URALI');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('UCRAINA', 'AFGHANISTAN');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('UCRAINA', 'MEDIO ORIENTE');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('EUROPA MERIDIONALE', 'EUROPA OCCIDENTALE');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('EUROPA MERIDIONALE', 'EUROPA SETTENTRIONALE');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('EUROPA MERIDIONALE', 'UCRAINA');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('EUROPA MERIDIONALE', 'MEDIO ORIENTE');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('EUROPA MERIDIONALE', 'AFRICA DEL NORD');
```

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('EUROPA MERIDIONALE', 'EGITTO');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('AFRICA DEL NORD', 'BRASILE');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('AFRICA DEL NORD', 'EUROPA OCCIDENTALE');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('AFRICA DEL NORD', 'EUROPA MERIDIONALE');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('AFRICA DEL NORD', 'EGITTO');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('AFRICA DEL NORD', 'CONGO');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('AFRICA DEL NORD', 'AFRICA ORIENTALE');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('EGITTO', 'EUROPA MERIDIONALE');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('EGITTO', 'AFRICA DEL NORD');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('EGITTO', 'AFRICA ORIENTALE');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('EGITTO', 'MEDIO ORIENTE');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('CONGO', 'AFRICA DEL NORD');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('CONGO', 'AFRICA ORIENTALE');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('CONGO', 'AFRICA DEL SUD');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('AFRICA ORIENTALE', 'AFRICA DEL NORD');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('AFRICA ORIENTALE', 'EGITTO');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('AFRICA ORIENTALE', 'CONGO');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('AFRICA ORIENTALE', 'AFRICA DEL SUD');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('AFRICA ORIENTALE', 'MADAGASCAR');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('AFRICA DEL SUD', 'CONGO');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('AFRICA DEL SUD', 'AFRICA ORIENTALE');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('AFRICA DEL SUD', 'MADAGASCAR');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('MADAGASCAR', 'AFRICA ORIENTALE');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('MADAGASCAR', 'AFRICA DEL SUD');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('URALI', 'UCRAINA');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('URALI', 'SIBERIA');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('URALI', 'AFGHANISTAN');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('URALI', 'CINA');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('SIBERIA', 'JACUZIA');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('SIBERIA', 'URALI');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('SIBERIA', 'CITA');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('SIBERIA', 'MONGOLIA');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('SIBERIA', 'CINA');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('JACUZIA', 'SIBERIA');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('JACUZIA', 'KAMCHATKA');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('JACUZIA', 'CITA');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('KAMCHATKA', 'ALASKA');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('KAMCHATKA', 'JACUZIA');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('KAMCHATKA', 'CITA');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('KAMCHATKA', 'MONGOLIA');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('KAMCHATKA', 'GIAPPONE');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('CITA', 'SIBERIA');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('CITA', 'JACUZIA');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('CITA', 'KAMCHATKA');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('CITA', 'MONGOLIA');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('GIAPPONE', 'KAMCHATKA');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('GIAPPONE', 'MONGOLIA');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('AFGHANISTAN', 'UCRAINA');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('AFGHANISTAN', 'URALI');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('AFGHANISTAN', 'CINA');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('AFGHANISTAN', 'MEDIO ORIENTE');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('MONGOLIA', 'SIBERIA');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('MONGOLIA', 'CITA');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('MONGOLIA', 'KAMCHATKA');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('MONGOLIA', 'CINA');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('MONGOLIA', 'GIAPPONE');

```
INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('CINA', 'AFGHANISTAN');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('CINA', 'URALI');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('CINA', 'SIBERIA');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('CINA', 'MONGOLIA');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('CINA', 'SIAM');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('CINA', 'INDIA');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('CINA', 'MEDIO ORIENTE');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('MEDIO ORIENTE', 'EUROPA MERIDIONALE');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('MEDIO ORIENTE', 'AFGHANISTAN');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('MEDIO ORIENTE', 'UCRAINA');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('MEDIO ORIENTE', 'CINA');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('MEDIO ORIENTE', 'INDIA');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('MEDIO ORIENTE', 'EGITTO');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('INDIA', 'CINA');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('INDIA', 'SIAM');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('INDIA', 'MEDIO ORIENTE');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('SIAM', 'INDIA');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('SIAM', 'CINA');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('SIAM', 'INDONESIA');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('INDONESIA', 'SIAM');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('INDONESIA', 'NUOVA GUINEA');
```

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('INDONESIA', 'AUSTRALIA OCCIDENTALE');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('NUOVA GUINEA', 'INDONESIA');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('NUOVA GUINEA', 'AUSTRALIA OCCIDENTALE');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('NUOVA GUINEA', 'AUSTRALIA ORIENTALE');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('AUSTRALIA OCCIDENTALE', 'NUOVA GUINEA');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('AUSTRALIA OCCIDENTALE', 'INDONESIA');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('AUSTRALIA OCCIDENTALE', 'AUSTRALIA ORIENTALE');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('AUSTRALIA ORIENTALE', 'AUSTRALIA OCCIDENTALE');

INSERT INTO `mydb`.`Confina` (`stato`, `confinante`) VALUES ('AUSTRALIA ORIENTALE', 'NUOVA GUINEA');

COMMIT;

Codice del Front-End

Il codice in c è strutturato in più file quindi riporto di seguito il codice per ogni file, preceduto dal nome del file stesso.

defines.h:

```
#pragma once

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

```
struct configuration {
	char* host;
	char* db_username;
	char* db_password;
	unsigned int port;
	char* database;

	char username[128];
	char password[128];
};

extern struct configuration conf;

extern int parse_config(char* path, struct configuration* conf);
extern bool yesOrNo(char* domanda, char yes, char no, bool predef, bool insensitive);
extern char multiChoice(char* domanda, char choices[], int num);
extern void insertPassword(char password[]);
extern void print_error(MYSQL* conn, char* message);
extern void print_stmt_error(MYSQL_STMT* stmt, char* message);
extern void finish_with_error(MYSQL* conn, char* message);
extern void finish_with_stmt_error(MYSQL* conn, MYSQL_STMT* stmt, char* message, bool
close_stmt);
extern bool setup_prepared_stmt(MYSQL_STMT** stmt, char* statement, MYSQL* conn);
extern void dump_result_set(MYSQL* conn, MYSQL_STMT* stmt, char* title);
extern void run_as_moderatore(MYSQL* conn);
extern void run_as_giocatore(MYSQL* conn);
```

utils.c:

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

#include "defines.h"

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

```
void print_error(MYSQL* conn, char* message)
{
	fprintf(stderr, "%s\n", message);
	if (conn != NULL) {
#if MYSQL_VERSION_ID >= 40101
		fprintf(stderr, "Error %u (%s): %s\n",
			mysql_errno(conn), mysql_sqlstate(conn), mysql_error(conn));
#else
		fprintf(stderr, "Error %u: %s\n",
			mysql_errno(conn), mysql_error(conn));
#endif
	}
}

bool setup_prepared_stmt(MYSQL_STMT** stmt, char* statement, MYSQL* conn)
{
	bool update_length = true;

	*stmt = mysql_stmt_init(conn);
	if (*stmt == NULL)
	{
		print_error(conn, "Could not initialize statement handler");
		return false;
	}

	if (mysql_stmt_prepare(*stmt, statement, strlen(statement)) != 0) {
		print_stmt_error(*stmt, "Could not prepare statement");
		return false;
	}

	mysql_stmt_attr_set(*stmt, STMT_ATTR_UPDATE_MAX_LENGTH, &update_length);

	return true;
}

void finish_with_error(MYSQL* conn, char* message)
{
	print_error(conn, message);
	mysql_close(conn);
	exit(EXIT_FAILURE);
}

void finish_with_stmt_error(MYSQL* conn, MYSQL_STMT* stmt, char* message, bool
close_stmt)
{
	print_stmt_error(stmt, message);
	if (close_stmt)		mysql_stmt_close(stmt);
	mysql_close(conn);
	exit(EXIT_FAILURE);
}
```

```
static void print_dashes(MYSQL_RES* res_set)
{
	MYSQL_FIELD* field;
	unsigned int i, j;

	mysql_field_seek(res_set, 0);
	putchar('+');
	for (i = 0; i < mysql_num_fields(res_set); i++) {
		field = mysql_fetch_field(res_set);
		for (j = 0; j < field->max_length + 2; j++)
			putchar('-');
		putchar('+');
	}
	putchar('\n');
}

static void dump_result_set_header(MYSQL_RES* res_set)
{
	MYSQL_FIELD* field;
	unsigned long col_len;
	unsigned int i;

	/* determine column display widths -- requires result set to be */
	/* generated with mysql_store_result(), not mysql_use_result() */

	mysql_field_seek(res_set, 0);

	for (i = 0; i < mysql_num_fields(res_set); i++) {
		field = mysql_fetch_field(res_set);
		col_len = strlen(field->name);

		if (col_len < field->max_length)
			col_len = field->max_length;
		if (col_len < 4 && !IS_NOT_NULL(field->flags))
			col_len = 4; /* 4 = length of the word "NULL" */
		field->max_length = col_len; /* reset column info */
	}

	print_dashes(res_set);
	putchar('|');
	mysql_field_seek(res_set, 0);
	for (i = 0; i < mysql_num_fields(res_set); i++) {
		field = mysql_fetch_field(res_set);
		printf(" %-*s |", (int)field->max_length, field->name);
	}
	putchar('\n');

	print_dashes(res_set);
}
```

```
void dump_result_set(MYSQL* conn, MYSQL_STMT* stmt, char* title)
{
	int i;
	int status;
	int num_fields;	/* number of columns in result */
	MYSQL_FIELD* fields;  /* for result set metadata */
	MYSQL_BIND* rs_bind;  /* for output buffers */
	MYSQL_RES* rs_metadata;
	MYSQL_TIME* date;
	size_t attr_size;

	/* Prefetch the whole result set. This in conjunction with
	 * STMT_ATTR_UPDATE_MAX_LENGTH set in `setup_prepared_stmt`
	 * updates the result set metadata which are fetched in this
	 * function, to allow to compute the actual max length of
	 * the columns.
	 */
	if (mysql_stmt_store_result(stmt)) {
		fprintf(stderr, " mysql_stmt_execute(), 1 failed\n");
		fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
		exit(0);
	}

	/* the column count is > 0 if there is a result set */
	/* 0 if the result is only the final status packet */
	num_fields = mysql_stmt_field_count(stmt);

	if (num_fields > 0) {
		/* there is a result set to fetch */
		printf("%s\n", title);

		if ((rs_metadata = mysql_stmt_result_metadata(stmt)) == NULL) {
			finish_with_stmt_error(conn, stmt, "Unable to retrieve result metadata\n",
true);
		}

		dump_result_set_header(rs_metadata);

		fields = mysql_fetch_fields(rs_metadata);

		rs_bind = (MYSQL_BIND*)malloc(sizeof(MYSQL_BIND) * num_fields);
		if (!rs_bind) {
			finish_with_stmt_error(conn, stmt, "Cannot allocate output buffers\n", true);
		}
		memset(rs_bind, 0, sizeof(MYSQL_BIND) * num_fields);

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

```
                // Properly size the parameter buffer
                switch (fields[i].type) {
                case MYSQL_TYPE_DATE:
                case MYSQL_TYPE_TIMESTAMP:
                case MYSQL_TYPE_DATETIME:
                case MYSQL_TYPE_TIME:
                        attr_size = sizeof(MYSQL_TIME);
                        break;
                case MYSQL_TYPE_FLOAT:
                        attr_size = sizeof(float);
                        break;
                case MYSQL_TYPE_DOUBLE:
                        attr_size = sizeof(double);
                        break;
                case MYSQL_TYPE_TINY:
                        attr_size = sizeof(signed char);
                        break;
                case MYSQL_TYPE_SHORT:
                case MYSQL_TYPE_YEAR:
                        attr_size = sizeof(short int);
                        break;
                case MYSQL_TYPE_LONG:
                case MYSQL_TYPE_INT24:
                        attr_size = sizeof(int);
                        break;
                case MYSQL_TYPE_LONGLONG:
                        attr_size = sizeof(long long int);
                        break;
                default:
                        attr_size = fields[i].max_length;
                        break;
                }

                // Setup the binding for the current parameter
                rs_bind[i].buffer_type = fields[i].type;
                rs_bind[i].buffer = malloc(attr_size + 1);
                rs_bind[i].buffer_length = attr_size + 1;

                if (rs_bind[i].buffer == NULL) {
                        finish_with_stmt_error(conn, stmt, "Cannot allocate output buffers\n",
true);
                }
        }

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

        /* fetch and display result set rows */
```

```
		while (true) {
			status = mysql_stmt_fetch(stmt);

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

			putchar('|');

			for (i = 0; i < num_fields; i++) {

				if (rs_bind[i].is_null_value) {
					printf(" %-*s |", (int)fields[i].max_length, "NULL");
					continue;
				}

				switch (rs_bind[i].buffer_type) {

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

				case MYSQL_TYPE_DATE:
				case MYSQL_TYPE_TIMESTAMP:
					date = (MYSQL_TIME*)rs_bind[i].buffer;
					printf(" %d-%02d-%02d |", date->year, date->month, date-
>day);
					break;

				case MYSQL_TYPE_STRING:
					printf(" %-*s |", (int)fields[i].max_length,
(char*)rs_bind[i].buffer);
					break;

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

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

				case MYSQL_TYPE_LONGLONG:
					printf(" %-*lld |", (long long int)fields[i].max_length, *(long
long int*)rs_bind[i].buffer);
```

```
                                        break;

                                case MYSQL_TYPE_NEWDECIMAL:
                                        printf(" %-*.02lf |", (int)fields[i].max_length,
*(float*)rs_bind[i].buffer);
                                        break;

                                default:
                                        printf("ERROR: Unhandled type (%d)\n",
rs_bind[i].buffer_type);
                                        abort();
                                }
                        }
                        putchar('\n');
                        print_dashes(rs_metadata);
                }

                mysql_free_result(rs_metadata); /* free metadata */

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

parse.c:

```
#pragma warning(disable : 4996)
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "defines.h"

#define BUFF_SIZE 4096

// The final config struct will point into this
static char config[BUFF_SIZE];


char* strndup(char* str, int chars)
{
        char* buffer;
```

```
	int n;

	buffer = (char*)malloc((chars + 1)*sizeof(char));
	if (buffer)
	{
		for (n = 0; ((n < chars) && (str[n] != 0)); n++)
			buffer[n] = str[n];

		buffer[n] = 0;
	}
	return buffer;
}


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

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

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

```
	int parent;
#endif
} jsmntok_t;

/**
 * JSON parser. Contains an array of token blocks available. Also stores
 * the string being parsed now and current position in that string
 */
typedef struct {
	unsigned int pos; /* offset in the JSON string */
	unsigned int toknext; /* next token to allocate */
	int toksuper; /* superior token node, e.g parent object or array */
} jsmn_parser;

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

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

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

```
	start = parser->pos;

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

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

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

	int start = parser->pos;
```

```
	parser->pos++;

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

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

		/* Backslash: Quoted symbol expected */
		if (c == '\\' && parser->pos + 1 < len) {
			int i;
			parser->pos++;
			switch (js[parser->pos]) {
				/* Allowed escaped symbols */
			case '\"': case '/': case '\\': case 'b':
			case 'f': case 'r': case 'n': case 't':
				break;
				/* Allows escaped symbol \uXXXX */
			case 'u':
				parser->pos++;
				for (i = 0; i < 4 && parser->pos < len && js[parser->pos] != '\0'; i++)
{
					/* If it isn't a hex character we have an error */
					if (!((js[parser->pos] >= 48 && js[parser->pos] <= 57) || /* 0-9
*/
						(js[parser->pos] >= 65 && js[parser->pos] <= 70) || /*
A-F */
						(js[parser->pos] >= 97 && js[parser->pos] <= 102))) {
/* a-f */
						parser->pos = start;
						return JSMN_ERROR_INVAL;
					}
					parser->pos++;
				}
				parser->pos--;
```

```
                                break;
                                /* Unexpected symbol */
                        default:
                                parser->pos = start;
                                return JSMN_ERROR_INVAL;
                        }
                }
        }
        parser->pos = start;
        return JSMN_ERROR_PART;
}

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

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

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

```
#ifdef JSMN_PARENT_LINKS
                        if (parser->toknext < 1) {
                                return JSMN_ERROR_INVAL;
                        }
                        token = &tokens[parser->toknext - 1];
                        for (;;) {
                                if (token->start != -1 && token->end == -1) {
                                        if (token->type != type) {
                                                return JSMN_ERROR_INVAL;
                                        }
                                        token->end = parser->pos + 1;
                                        parser->toksuper = token->parent;
                                        break;
                                }
                                if (token->parent == -1) {
                                        if (token->type != type || parser->toksuper == -1) {
                                                return JSMN_ERROR_INVAL;
                                        }
                                        break;
                                }
                                token = &tokens[token->parent];
                        }
#else
                        for (i = parser->toknext - 1; i >= 0; i--) {
                                token = &tokens[i];
                                if (token->start != -1 && token->end == -1) {
                                        if (token->type != type) {
                                                return JSMN_ERROR_INVAL;
                                        }
                                        parser->toksuper = -1;
                                        token->end = parser->pos + 1;
                                        break;
                                }
                        }
                        /* Error if unmatched closing bracket */
                        if (i == -1) return JSMN_ERROR_INVAL;
                        for (; i >= 0; i--) {
                                token = &tokens[i];
                                if (token->start != -1 && token->end == -1) {
                                        parser->toksuper = i;
                                        break;
                                }
                        }
#endif
                        break;
                case '\"':
                        r = jsmn_parse_string(parser, js, len, tokens, num_tokens);
                        if (r < 0) return r;
                        count++;
                        if (parser->toksuper != -1 && tokens != NULL)
```

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

```
#ifdef JSMN_STRICT
                        /* Unexpected char in strict mode */
                default:
                        return JSMN_ERROR_INVAL;
#endif
                }
        }

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

        return count;
}

/**
 * Creates a new parser based over a given  buffer with an array of tokens
 * available.
 */
static void jsmn_init(jsmn_parser* parser) {
        parser->pos = 0;
        parser->toknext = 0;
        parser->toksuper = -1;
}

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

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

        fseek(f, 0, SEEK_END);
        size_t fsize = ftell(f);
```

```
	fseek(f, 0, SEEK_SET);  //same as rewind(f);

	if (fsize >= BUFF_SIZE) {
		fprintf(stderr, "Configuration file too large\n");
		abort();
	}

	fread(config, fsize, 1, f);
	fclose(f);

	config[fsize] = 0;
	return fsize;
}


int parse_config(char* path, struct configuration* conf)
{
	int i;
	int r;
	jsmn_parser p;
	jsmntok_t t[128]; /* We expect no more than 128 tokens */

	load_file(path);

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

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

	/* Loop over all keys of the root object */
	for (i = 1; i < r; i++) {
		if (jsoneq(config, &t[i], "host") == 0) {
			/* We may use strndup() to fetch string value */
			conf->host = strndup(config + t[i + 1].start, t[i + 1].end - t[i + 1].start);
			i++;
		}
		else if (jsoneq(config, &t[i], "username") == 0) {
			conf->db_username = strndup(config + t[i + 1].start, t[i + 1].end - t[i +
1].start);
			i++;
		}
		else if (jsoneq(config, &t[i], "password") == 0) {
```

```
                conf->db_password = strndup(config + t[i + 1].start, t[i + 1].end - t[i +
1].start);
                i++;
            }
            else if (jsoneq(config, &t[i], "port") == 0) {
                conf->port = strtol(config + t[i + 1].start, NULL, 10);
                i++;
            }
            else if (jsoneq(config, &t[i], "database") == 0) {
                conf->database = strndup(config + t[i + 1].start, t[i + 1].end - t[i + 1].start);
                i++;
            }
            else {
                printf("Unexpected key: %.*s\n", t[i].end - t[i].start, config + t[i].start);
            }
        }
        return 1;
}
```

inout.c:

```
#include <stdio.h>
#include <stdlib.h>
#include <Windows.h>
#include <stdbool.h>

#include "defines.h"


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


void insertPassword(char password[]) {
    int ch;
    int i = 0;
    while ((ch = _getch()) != EOF && ch != '\n' && ch != '\r' && i < sizeof(password) - 1) {
        if (ch == '\b' && i > 0) {
            printf("\b \b"); // Destructive backspace
            fflush(stdout);
            i--;
            password[i] = '\0';
        }
        else if (ch != '\b') {
            putchar('*');
            password[i++] = (char)ch;
        }
```

```
	}
	password[i] = '\0';
	printf("\n");
	fflush(stdout);
}

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;
		fgets(&c, 2, stdin);

		// Controlla quale risposta e' stata data
		if (c == '\0') {
			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(yes)) {
			if (!predef || insensitive) return false;
		}
	}
}
```

```
char multiChoice(char* domanda, char choices[], int num)
{
	char inputStr[3];
	// Genera la stringa delle possibilita'
	char* possib = malloc(2 * num * sizeof(char));
	int i, j = 0;
	for (i = 0; i < num; i++) {
		possib[j++] = choices[i];
		possib[j++] = '/';
	}
	possib[j - 1] = '\0'; // Per eliminare l'ultima '/'

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

		fgets(inputStr, 3, stdin);
		char c = inputStr[0];

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

main.c:

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

#include "defines.h"


typedef enum {
	GIOCATORE = 1,
	MODERATORE,
	FAILED_LOGIN
} role_t;

#define DIM 46
```

```
struct configuration conf;

static MYSQL* conn;

/*Funzione che mi permettera' tramite prepared statement di invocare la store procedure login*/
static role_t attempt_login(MYSQL* conn, char* username, char* password) {
	MYSQL_STMT* login_procedure; /*Struttura del C connector*/

	/*3 sono i parametri che dobbiamo passare alla store procedure login, quindi dovro' fare 3 binding*/
	MYSQL_BIND param[3]; // Used both for input and output
	int role = 0;

	/*setup_prepared_stmt e' definita in utils.c*/
	if (!setup_prepared_stmt(&login_procedure, "call login(?, ?, ?)", conn)) {
		print_stmt_error(login_procedure, "Unable to initialize login statement\n");
		goto err2;
	}

	// Prepare parameters
	/*Binding di tutti i parametri da passare, sia gli in che gli out*/
	memset(param, 0, sizeof(param));

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

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

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

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

	// Run procedure
	/*Invio i parametri al DBMS*/
	if (mysql_stmt_execute(login_procedure) != 0) {
		print_stmt_error(login_procedure, "Could not execute login procedure");
		goto err;
	}

	// Prepare output parameters
	/*Binding al contrario per ottenere dal DBMS il parametro out della procedura login*/
```

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

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

	// Retrieve output parameter
	/*Estraggo il parametro out dal result set, che in questo caso e' una tabella 1x1*/
	if (mysql_stmt_fetch(login_procedure)) {
		print_stmt_error(login_procedure, "Could not buffer results");
		goto err;
	}

	/*Chiudo lo statement e restituisco il ruolo (che sara' un intero che viene castato a role_t che
e' una enum)*/
	mysql_stmt_close(login_procedure);
	return role;

err:
	mysql_stmt_close(login_procedure);
err2:
	return FAILED_LOGIN;
}

void login(MYSQL *conn) {
	role_t role;


	printf("Username: ");
	fflush(stdout);
	fgets(conf.username, DIM, stdin);
	conf.username[strlen(conf.username) - 1] = '\0';
	printf("\nPassword: ");
	fflush(stdout);
	insertPassword(conf.password);

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

	/*A seconda del ruolo dell'utente che ha fatto il login, voglio far girare il mio applicativo
	come se fossi un moderatore o come se fossi un giocatore o in caso di login fallito uscire*/
	switch (role) {
	case MODERATORE:
		run_as_moderatore(conn);
		break;

	case GIOCATORE:
```

```
                run_as_giocatore(conn);
                break;

        case FAILED_LOGIN:
                fprintf(stderr, "Invalid credentials\n");
                exit(EXIT_FAILURE);
                break;

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

}

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

        char username[46];
        char password[46];

        printf("Inserire username: ");
        fflush(stdout);
        fgets(username, DIM, stdin);
        username[strlen(username) - 1] = '\0';
        printf("\nInserire password: ");
        fflush(stdout);
        insertPassword(password);

        // Prepare stored procedure call
        if (!setup_prepared_stmt(&prepared_stmt, "call aggiungi_giocatore(?, ?)", conn)) {
                finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize aggiungi_giocatore
statement\n", false);
        }

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

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

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



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

```
		finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters for
aggiungi_giocatore\n", true);
	}

	// Run procedure
	if (mysql_stmt_execute(prepared_stmt) != 0) {
		print_stmt_error(prepared_stmt, "An error occurred while retrieving
aggiungi_giocatore");
	}else
		printf("Giocatore registrato correttamente\n");

	mysql_stmt_close(prepared_stmt);
}

int main(void) {
	char options[3] = {'1','2','3'};
	char op;

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

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

	if (mysql_real_connect(conn, conf.host, conf.db_username, conf.db_password, conf.database,
conf.port, NULL, CLIENT_MULTI_STATEMENTS | CLIENT_MULTI_RESULTS) == NULL) {
		fprintf(stderr, "mysql_real_connect() failed\n");
		mysql_close(conn);
		exit(EXIT_FAILURE);
	}

	while (1) {
		system("cls");
		//printf("\033[2J\033[H");
		printf("*** Cosa desideri fare? ***\n\n");
		printf("1) Login\n");
		printf("2) Registrazione\n");
		printf("3) Uscire\n");

		op = multiChoice("Select an option", options, 3);

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

```
                case '2':
                        registra_giocatore(conn);
                        break;

                case '3':
                        return;

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

        printf("Bye!\n");

        mysql_close(conn);
        return 0;
}
```

giocatore.c:

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

#include "defines.h"

#define DIM 16


static void stato_di_gioco(MYSQL* conn) {
        MYSQL_STMT* prepared_stmt;
        MYSQL_BIND param[1];

        // Prepare stored procedure call
        if (!setup_prepared_stmt(&prepared_stmt, "call stato_di_gioco(?)", conn)) {
                finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize stato_di_gioco
statement\n", false);
                goto err2;
        }

        // Prepare parameters
```

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

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

	if (mysql_stmt_bind_param(prepared_stmt, param) != 0) { // Note _param
		print_stmt_error(prepared_stmt, "Could not bind parameters for stato_di_gioco");
		goto err;
	}

	// Run procedure
	/*Inviamo i parametri al DBMS*/
	if (mysql_stmt_execute(prepared_stmt) != 0) {
		print_stmt_error(prepared_stmt, "Could not execute stato_di_gioco procedure");
		goto err;
	}

	// Dump the result set
	dump_result_set(conn, prepared_stmt, "\nStato di gioco della partita");
	mysql_stmt_next_result(prepared_stmt);
	mysql_stmt_close(prepared_stmt);

	return;

err:
	mysql_stmt_close(prepared_stmt);
err2:
	return;
}


static void posizionamento(MYSQL* conn) {
	MYSQL_STMT* prepared_stmt;
	MYSQL_BIND param[3];
	char nome_stato[26];
	char num_armate[16];
	int num_armate_int;

	// Prepare stored procedure call
	if (!setup_prepared_stmt(&prepared_stmt, "call posiziona_armate(?,?,?)", conn)) {
		finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize posiziona_armate
statement\n", false);
		goto err2;
	}

	printf("Nome stato in cui posizionare armate: ");
	fflush(stdout);
	fgets(nome_stato, 26, stdin);
	nome_stato[strlen(nome_stato) - 1] = '\0';
```

```
	printf("Numero armate da posizionare: ");
	fgets(num_armate, DIM, stdin);
	num_armate[strlen(num_armate) - 1] = '\0';
	// Convert values
	num_armate_int = atoi(num_armate);
	if (num_armate_int == 0) {
		printf("Inserisci un numero di armate valido\n");
		goto err2;
	}

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

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

	param[1].buffer_type = MYSQL_TYPE_LONG; // IN
	param[1].buffer = &num_armate_int;
	param[1].buffer_length = sizeof(num_armate_int);

	param[2].buffer_type = MYSQL_TYPE_VAR_STRING; // IN
	param[2].buffer = nome_stato;
	param[2].buffer_length = strlen(nome_stato);

	if (mysql_stmt_bind_param(prepared_stmt, param) != 0) { // Note _param
		print_stmt_error(prepared_stmt, "Could not bind parameters for posiziona_armate");
		goto err;
	}

	// Run procedure
	/*Inviamo i parametri al DBMS*/
	if (mysql_stmt_execute(prepared_stmt) != 0) {
		print_stmt_error(prepared_stmt, "Could not execute posiziona_armate procedure");
		goto err;
	}

	/*Chiudo lo statement*/
	mysql_stmt_close(prepared_stmt);
	printf("L'azione di posizionamento e' stata effettuata correttamente\n");
	return;

err:
	mysql_stmt_close(prepared_stmt);
err2:
	return;
}

static void spostamento(MYSQL* conn) {
```

```
	MYSQL_STMT* prepared_stmt;
	MYSQL_BIND param[4];
	char nome_stato_part[26];
	char nome_stato_dest[26];
	char num_armate[16];
	int num_armate_int;

	// Prepare stored procedure call
	if (!setup_prepared_stmt(&prepared_stmt, "call sposta_armate(?,?,?,?)", conn)) {
		finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize sposta_armate statement\n", false);
		goto err2;
	}

	printf("Nome stato da cui prendere le armate da spostare: ");
	fflush(stdout);
	fgets(nome_stato_part, 26, stdin);
	nome_stato_part[strlen(nome_stato_part) - 1] = '\0';

	printf("Nome stato in cui spostare le armate: ");
	fflush(stdout);
	fgets(nome_stato_dest, 26, stdin);
	nome_stato_dest[strlen(nome_stato_dest) - 1] = '\0';

	printf("Numero armate da spostare: ");
	fgets(num_armate, DIM, stdin);
	num_armate[strlen(num_armate) - 1] = '\0';
	// Convert values
	num_armate_int = atoi(num_armate);
	if (num_armate_int == 0) {
		printf("Inserisci un numero di armate valido\n");
		goto err2;
	}


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

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

	param[1].buffer_type = MYSQL_TYPE_LONG; // IN
	param[1].buffer = &num_armate_int;
	param[1].buffer_length = sizeof(num_armate_int);

	param[2].buffer_type = MYSQL_TYPE_VAR_STRING; // IN
	param[2].buffer = nome_stato_part;
	param[2].buffer_length = strlen(nome_stato_part);
```

```
	param[3].buffer_type = MYSQL_TYPE_VAR_STRING; // IN
	param[3].buffer = nome_stato_dest;
	param[3].buffer_length = strlen(nome_stato_dest);


	if (mysql_stmt_bind_param(prepared_stmt, param) != 0) { // Note _param
		print_stmt_error(prepared_stmt, "Could not bind parameters for sposta_armate");
		goto err;
	}

	// Run procedure
	/*Inviamo i parametri al DBMS*/
	if (mysql_stmt_execute(prepared_stmt) != 0) {
		print_stmt_error(prepared_stmt, "Could not execute sposta_armate procedure");
		goto err;
	}

	/*Chiudo lo statement*/
	mysql_stmt_close(prepared_stmt);
	printf("L'azione di spostamento e' stata effettuata correttamente\n");
	return;

err:
	mysql_stmt_close(prepared_stmt);
err2:
	return;
}


static void attacco(MYSQL* conn) {
	MYSQL_STMT* prepared_stmt;
	MYSQL_BIND param[4];
	char nome_stato_att[26];
	char nome_stato_dif[26];
	char num_armate[16];
	int num_armate_int;

	// Prepare stored procedure call
	if (!setup_prepared_stmt(&prepared_stmt, "call effettua_attacco(?,?,?,?)", conn)) {
		finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize effettua_attacco
statement\n", false);
		goto err2;
	}

	printf("Nome stato attaccante: ");
	fflush(stdout);
	fgets(nome_stato_att, 26, stdin);
	nome_stato_att[strlen(nome_stato_att) - 1] = '\0';

	printf("Nome stato attaccato: ");
```

```
fflush(stdout);
fgets(nome_stato_dif, 26, stdin);
nome_stato_dif[strlen(nome_stato_dif) - 1] = '\0';

printf("Numero armate da schierare in attacco: ");
fgets(num_armate, DIM, stdin);
num_armate[strlen(num_armate) - 1] = '\0';
// Convert values
num_armate_int = atoi(num_armate);
if (num_armate_int == 0) {
        printf("Inserisci un numero di armate valido\n");
        goto err2;
}


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

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

param[1].buffer_type = MYSQL_TYPE_LONG; // IN
param[1].buffer = &num_armate_int;
param[1].buffer_length = sizeof(num_armate_int);

param[2].buffer_type = MYSQL_TYPE_VAR_STRING; // IN
param[2].buffer = nome_stato_att;
param[2].buffer_length = strlen(nome_stato_att);

param[3].buffer_type = MYSQL_TYPE_VAR_STRING; // IN
param[3].buffer = nome_stato_dif;
param[3].buffer_length = strlen(nome_stato_dif);


if (mysql_stmt_bind_param(prepared_stmt, param) != 0) { // Note _param
        print_stmt_error(prepared_stmt, "Could not bind parameters for effettua_attacco");
        goto err;
}

// Run procedure
/*Inviamo i parametri al DBMS*/
if (mysql_stmt_execute(prepared_stmt) != 0) {
        print_stmt_error(prepared_stmt, "Could not execute effettua_attacco procedure");
        goto err;
}

/*Chiudo lo statement*/
mysql_stmt_close(prepared_stmt);
printf("L'azione di attacco e' stata effettuata correttamente\n");
```

```
	return;

err:
	mysql_stmt_close(prepared_stmt);
err2:
	return;
}

static void gioca_turno(MYSQL* conn) {
	char options[5] = { '1','2','3','4','5' };
	char op;

	/*Ciclo infinito*/
	while (1) {
		/*Cancello quello che c'e' sullo schermo*/
		printf("\033[2J\033[H");
		printf("*** Quale azione vuoi effettuare? ***\n\n");
		printf("1) Visualizzare lo stato di gioco\n");
		printf("2) Posizionare armate\n");
		printf("3) Spostare armate\n");
		printf("4) Attaccare uno stato\n");
		printf("5) Uscire\n");

		/*recupero l'operazione che voglio andare a realizzare chiamando multiChoice definita in inout.c */
		op = multiChoice("Select an option", options, 5);

		/*Tramite uno switch case attivo la funzione corrispondente all'operazione
		che voglio andare a realizzare*/
		switch (op) {
		case '1':
			stato_di_gioco(conn);
			break;

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

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

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

		case '5':
			return;

		default:
```

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


static void crea_partita(MYSQL* conn) {
        MYSQL_STMT* prepared_stmt;
        MYSQL_BIND param[3];
        char nome_stanza[16];
        int idpartita;

        // Prepare stored procedure call
        if (!setup_prepared_stmt(&prepared_stmt, "call visualizza_stanze_libere()", conn)) {
                finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize stanze_libere
statement\n", false);
        }

        // Run procedure
        if (mysql_stmt_execute(prepared_stmt) != 0) {
                print_stmt_error(prepared_stmt, "An error occurred while retrieving stanze_libere.");
        }

        // Dump the result set
        dump_result_set(conn, prepared_stmt, "\nLista delle stanze di gioco libere");
        mysql_stmt_next_result(prepared_stmt);
        mysql_stmt_close(prepared_stmt);

        printf("Nome stanza: ");
        fflush(stdout);
        fgets(nome_stanza, DIM, stdin);
        nome_stanza[strlen(nome_stanza) - 1] = '\0';

        // Prepare stored procedure call
        if (!setup_prepared_stmt(&prepared_stmt, "call crea_partita(?, ?, ?)", conn)) {
                finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize crea_partita
statement\n", false);
                goto err2;
        }

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

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

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

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

	param[2].buffer_type = MYSQL_TYPE_LONG; // OUT
	param[2].buffer = &idpartita;
	param[2].buffer_length = sizeof(idpartita);


	if (mysql_stmt_bind_param(prepared_stmt, param) != 0) { // Note _param
		print_stmt_error(prepared_stmt, "Could not bind parameters for crea_partita");
		goto err;
	}

	// Run procedure
	/*Inviamo i parametri al DBMS*/
	if (mysql_stmt_execute(prepared_stmt) != 0) {
		print_stmt_error(prepared_stmt, "Could not execute crea_partita procedure");
		goto err;
	}

	// Prepare output parameters
	/*Binding al contrario per ottenere dal DBMS il parametro out della procedura login*/
	memset(param, 0, sizeof(param));
	param[0].buffer_type = MYSQL_TYPE_LONG; // OUT
	param[0].buffer = &idpartita;
	param[0].buffer_length = sizeof(idpartita);

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

	// Retrieve output parameter
	/*Estraggo il parametro out dal result set, che in questo caso e' una tabella 1x1*/
	if (mysql_stmt_fetch(prepared_stmt)) {
		print_stmt_error(prepared_stmt, "Could not buffer results");
		goto err;
	}

	/*Chiudo lo statement*/
	mysql_stmt_close(prepared_stmt);

	printf ("E' stata creata la partita con id: %i\n", idpartita);
	gioca_turno(conn);

err:
	mysql_stmt_close(prepared_stmt);
err2:
	return;
}
```

```
static void visualizza_storico_partite(MYSQL* conn) {
	MYSQL_STMT* prepared_stmt;
	MYSQL_BIND param[1];

	// Prepare stored procedure call
	if (!setup_prepared_stmt(&prepared_stmt, "call storico_partite(?)", conn)) {
		finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize storico_partite statement\n", false);
		goto err2;
	}

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

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

	if (mysql_stmt_bind_param(prepared_stmt, param) != 0) { // Note _param
		print_stmt_error(prepared_stmt, "Could not bind parameters for storico_partite\n");
		goto err;
	}

	// Run procedure
	/*Inviamo i parametri al DBMS*/
	if (mysql_stmt_execute(prepared_stmt) != 0) {
		print_stmt_error(prepared_stmt, "Could not execute storico_partite procedure\n");
		goto err;
	}

	// Dump the result set
	dump_result_set(conn, prepared_stmt, "Storico partite giocate\n");
	mysql_stmt_next_result(prepared_stmt);
	mysql_stmt_close(prepared_stmt);

	return;

	err:
		mysql_stmt_close(prepared_stmt);
	err2:
		return;
}
```

```
static void partecipa_partita(MYSQL* conn) {
```

```
	MYSQL_STMT* prepared_stmt;
	MYSQL_BIND param[2];
	char partita[16];
	int partita_int;

	// Prepare stored procedure call
	if (!setup_prepared_stmt(&prepared_stmt, "call visualizza_partite_in_attesa()", conn)) {
		finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize partite_in_attesa statement\n", false);
		goto err2;
	}

	// Run procedure
	if (mysql_stmt_execute(prepared_stmt) != 0) {
		print_stmt_error(prepared_stmt, "An error occurred while retrieving partite_in_attesa");
		goto err;
	}

	// Dump the result set
	dump_result_set(conn, prepared_stmt, "\nLista partite in attesa");
	mysql_stmt_next_result(prepared_stmt);
	mysql_stmt_close(prepared_stmt);

	printf("Partita: ");
	fflush(stdout);
	fgets(partita, DIM, stdin);
	partita[strlen(partita) - 1] = '\0';
	// Convert values
	partita_int = atoi(partita);

	// Prepare stored procedure call
	if (!setup_prepared_stmt(&prepared_stmt, "call partecipa_partita(?, ?)", conn)) {
		finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize partecipa_partita statement\n", false);
		goto err2;
	}

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

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

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

```
	if (mysql_stmt_bind_param(prepared_stmt, param) != 0) { // Note _param
		print_stmt_error(prepared_stmt, "Could not bind parameters for partecipa_partita");
		goto err;
	}

	// Run procedure
	/*Inviamo i parametri al DBMS*/
	if (mysql_stmt_execute(prepared_stmt) != 0) {
		print_stmt_error(prepared_stmt, "Could not execute partecipa_partita procedure");
		goto err;
	}

	mysql_stmt_close(prepared_stmt);

	printf("Giocatore entrato nella partita con id: %i\n", partita_int);
	gioca_turno(conn);

err:
	mysql_stmt_close(prepared_stmt);
err2:
	return;

}



void run_as_giocatore(MYSQL* conn)
{
	char options[4] = {'1','2','3','4' };
	char op;

	/*Faccio privilege escalation, cambiando il tipo di connessione che ho con il mio DBMS
	per girare non piu' come utente di tipo login ma come utente di tipo giocatore*/
	printf("Switching to giocatore role...\n");

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

	/*Sulla stessa connessione conn che e' gia instaurata voglio modificare qualcosa, in particolare
	voglio modificare lo username, la password ed eventualmente anche lo schema che sto utilizzando*/
	if (mysql_change_user(conn, conf.db_username, conf.db_password, conf.database)) {
		fprintf(stderr, "mysql_change_user() failed\n");
		exit(EXIT_FAILURE);
	}
```

```
	/*Ciclo infinito*/
	while (1) {
		/*Cancello quello che c'e' sullo schermo*/
		printf("\033[2J\033[H");
		printf("*** Cosa desideri fare? ***\n\n");
		printf("1) Creare una nuova partita\n");
		printf("2) Partecipare a una partita\n");
		printf("3) Visualizzare storico partite\n");
		printf("4) Uscire\n");

		/*recupero l'operazione che voglio andare a realizzare chiamando multiChoice definita in inout.c */
		op = multiChoice("Select an option", options, 4);

		/*Tramite uno switch case attivo la funzione corrispondente all'operazione
		che voglio andare a realizzare*/
		switch (op) {
		case '1':
			crea_partita(conn);
			break;

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

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

		case '4':
			return;

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

}
```

moderatore.c:

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

#include "defines.h"
```

```
#define DIM 46

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

	char username[46];
	char password[46];

	printf("Inserire username: ");
	fflush(stdout);
	fgets(username, DIM, stdin);
	username[strlen(username) - 1] = '\0';
	printf("\nInserire password: ");
	fflush(stdout);
	insertPassword(password);

	// Prepare stored procedure call
	if (!setup_prepared_stmt(&prepared_stmt, "call aggiungi_moderatore(?, ?)", conn)) {
		finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize
aggiungi_moderatore statement\n", false);
	}

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

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

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



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

	// Run procedure
	if (mysql_stmt_execute(prepared_stmt) != 0) {
		print_stmt_error(prepared_stmt, "An error occurred while retrieving
aggiungi_moderatore\n");
	}
	else
		printf("Nuovo moderatore registrato correttamente\n");
```

```
	mysql_stmt_close(prepared_stmt);
}


static void crea_stanzadigioco(MYSQL* conn) {
	MYSQL_STMT* prepared_stmt;
	MYSQL_BIND param[1];

	char nomestanza[16];

	printf("Inserire nome stanza di gioco: ");
	fflush(stdout);
	fgets(nomestanza, 16, stdin);
	nomestanza[strlen(nomestanza) - 1] = '\0';

	// Prepare stored procedure call
	if (!setup_prepared_stmt(&prepared_stmt, "call crea_stanzadigioco(?)", conn)) {
		finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize crea_stanzadigioco
statement\n", false);
	}

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

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



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

	// Run procedure
	if (mysql_stmt_execute(prepared_stmt) != 0) {
		print_stmt_error(prepared_stmt, "An error occurred while retrieving
crea_stanzadigioco\n");
	}
	else
		printf("Nuova stanza di gioco creata correttamente\n");

	mysql_stmt_close(prepared_stmt);
}



static void report(MYSQL* conn) {
	MYSQL_STMT* prepared_stmt;
```

```
	int status;
	char* title = "Numero di stanze di gioco con una partita attualmente in corso:";
	bool second_tb = false;
	bool third_tb = false;

	// Prepare stored procedure call
	if (!setup_prepared_stmt(&prepared_stmt, "call report_moderatore()", conn)) {
		finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize career report
statement\n", false);
	}

	// Run procedure
	if (mysql_stmt_execute(prepared_stmt) != 0) {
		print_stmt_error(prepared_stmt, "An error occurred while retrieving the career
report.");
		goto out;
	}

	// We have multiple result sets here!
	do {
		if (third_tb == true) {
			title = "\n\nNumero di giocatori che hanno effettuato almeno una un'azione
degli ultimi 15 minuti, che non sono in nessuna stanza di gioco:";
			second_tb = false;
			third_tb = false;
		}

		if (second_tb == true) {
		title = "\n\nStanze di gioco con una partita attualmente in corso e numero di giocatori
in ciascuna di queste ultime:";
		second_tb = false;
		third_tb = true;
		}

		dump_result_set(conn, prepared_stmt, title);
		// more results? -1 = no, >0 = error, 0 = yes (keep looking)
		status = mysql_stmt_next_result(prepared_stmt);
		if (status > 0)
			finish_with_stmt_error(conn, prepared_stmt, "Unexpected condition", true);

		second_tb = true;

	} while (status == 0);

out:
	mysql_stmt_close(prepared_stmt);
}

void run_as_moderatore(MYSQL* conn)
```

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

	/*Faccio privilege escalation, cambiando il tipo di connessione che ho con il mio DBMS
	per girare non piu' come utente di tipo login ma come utente di tipo moderatore*/
	printf("Switching to moderatore role...\n");

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

	/*Sulla stessa connessione conn che e' gia instaurata voglio modificare qualcosa, in
particolare
	voglio modificare lo username, la password ed eventualmente anche lo schema che sto
utilizzando*/
	if (mysql_change_user(conn, conf.db_username, conf.db_password, conf.database)) {
		fprintf(stderr, "mysql_change_user() failed\n");
		exit(EXIT_FAILURE);
	}

	/*Ciclo infinito*/
	while (1) {
		/*Cancello quello che c'e' sullo schermo*/
		printf("\033[2J\033[H");
		printf("*** Cosa desideri fare? ***\n\n");
		printf("1) Aggiungere un nuovo moderatore\n");
		printf("2) Creare una nuova stanza di gioco\n");
		printf("3) Visualizzare report\n");
		printf("4) Uscire\n");

		/*recupero l'operazione che voglio andare a realizzare chiamando multiChoice
		definita in inout.c */
		op = multiChoice("Select an option", options, 4);

		/*Tramite uno switch case attivo la funzione corrispondente all'operazione
		che voglio andare a realizzare*/
		switch (op) {
		case '1':
			aggiungi_moderatore(conn);
			break;

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

		case '3':
			report(conn);
			break;
```

```
                case '4':
                        return;

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

                _getch();
        }
}
```