Cocco Alberto 1167891
Mazzucchelli Alessandro 1171163

Fantasia: Second World

1. Abstract

Fantasia: Second World è un gioco di ruolo online (MMORPG o Massive Multiplayer Online RolePlaying Game) sviluppato e pubblicato indipendentemente da due studenti di informatica dell'Università di Padova all'inizio del 2018. L'idea è quella di testare le conoscenze acquisite nel proprio corso di studi, e di mettersi alla prova nel campo videoludico con un esperimento pratico. Pur essendo un primo tentativo, il gioco promette bene, grazie ad alcune scelte progettuali che garantiscono oltre alle tipiche caratteristiche di un mmorpg, alcune peculiarità che stimolano nuove e vecchie generazioni di giocatori a provare il titolo.
Come è tipico del genere, l'utente crea e controlla personaggi che faranno parte dell'avventura insieme ad altri come lui. Questo personaggio, o avatar, dovrà esplorare, completare missioni ed interagire con il mondo circostante.
Non avendo previsto che il bacino di utenti potesse crescere più di tanto, gli sviluppatori avevano inizialmente creato una base di dati essenziale, senza eccessive aspettative. Per far fronte all'aumento del numero di giocatori, e per rendere più facilmente gestibile le meccaniche di gioco, è richiesta l'estensione dell'attuale struttura della base di dati.

2. Analisi dei requisiti

2.1 Descrizione

Si vuole realizzare una base di dati che gestisce ed organizza le informazioni relative al gioco Fantasia: Second World. A tale scopo andremo ad analizzare le principali entità del progetto.

Server, caratterizzato da:

- Un indirizzo IP che lo identifica univocamente;
- La regione di appartenenza;
- Il nome del server;
- La capienza del server, ovvero quanti utenti può ospitare contemporaneamente;
- Lo stato attuale del server, che può essere online o offline.

Ogni server ospita grandi quantità di giocatori e può essere di due tipi:

- Server pvp: I giocatori agiscono in un mondo virtuale in cui si possono sfidare altri giocatori;
- Server pve: i giocatori possono incontrare altri giocatori ma posso sfidare solo personaggi controllati dall'intelligenza artificiale.

I server vengono suddivisi in reami, le cui caratteristiche sono:

- Un indirizzo id che lo identifica univocamente;
- Lo stato attuale del reame, che può essere online o offline;
- La popolazione, ovvero il numero di utenti totali del reame.

Ogni reame contiene un certo numero di giocatori caratterizzati da:

- Il proprio username, che lo identifica univocamente;
- Lo stato attuale del giocatore, che può essere online o offline;
- Un valore di ping.

Un gruppo di giocatori può decidere di formare un clan. I clan vengono così memorizzati:

- Un NomeClan, che lo identifica univocamente;
- N.membri, che indica quanti giocatori fanno parte del clan;

- Un Tag, ovvero una sigla che apparirà di fianco al nome del giocatore, per indicare l'appartenenza al clan.

I giocatori possono richiedere l'amicizia ad altri utenti, di cui verrà memorizzata la data d'inizio. Inoltre, ogni giocatore può mandare e ricevere messaggi da altri giocatori, anche se non amici. Le caratteristiche dei messaggi sono:

- Un indirizzo id che identifica ogni messaggio univocamente;
- Il testo del messaggio;
- La data in cui il messaggio è stato inviato.

I giocatori possono affrontare missioni in singolo oppure insieme ad altri giocatori riunendosi in lobby. Ogni lobby è caratterizzata da:

- ID Lobby: identificatore univoco della lobby;
- Proprietario: è il proprietario della lobby, ovvero colui che l'ha inizialmente creata;
- Data creazione: indica la data di creazione della lobby di gioco.

Ogni missione attribuisce ai giocatori delle ricompense ed è composta da:

- Codice: identifica univocamente la missione;
- Tipologia: indica il tipo di missione;
- Ricompensa: indica la ricompensa attribuita al termine della missione;
- A tempo: indica se la missione è a tempo oppure no.

Le missioni possono essere:

- Missione singolo: affrontata da un solo giocatore;
- Missione multigiocatore: affrontata da più giocatori appartenenti alla stessa lobby.

Se la missione è multigiocatore si indica anche il numero di membri.

Nel mondo di gioco l'utente utilizza dei personaggi virtuali (definiti anche avatar), di cui sono noti:

- ID personaggio: identifica univocamente il personaggio;
- NomePersonaggio : il nome del personaggio;
- Monete: soldi posseduti dal personaggio;
- Esperienza: valore numerico che quantifica quanta esperienza si ha con il personaggio;
- Potenza: valore numerico che indica la forza del personaggio;
- Punti vita: salute del personaggio.

Ogni personaggio può avere un impiego o essere specializzato in un'attività da cui può trarre un guadagno. Ogni professione è caratterizzata da:

- Nome: identifica univocamente la professione;
- Guadagno: ricavo della professione esercitata dal personaggio.

Ogni personaggio possiede inoltre un inventario in cui possono essere contenuti oggetti di qualsiasi tipo o armi per il combattimento con le rispettive quantità per ciascuno. Ogni inventario è caratterizzato da:

- ID Personaggio: identificativo del personaggio (chiave esterna);
- NomeOggetto: nome dell'oggetto;
- N.oggetti: quantità dell'oggetto.

ID Personaggio e NomeOggetto costituiscono un identificatore univoco primario.

Per ogni arma si vuole conoscere anche :

- ID arma: identificatore univoco dell'arma;
- Danno: danno che infligge l'arma con un colpo.

Le armi possono essere di due tipi:

- Arma bianca: es: spade, coltelli, bastoni, daghe.
- Arma a distanza(es: balestre, archi) con attributi gittata, MaxMunizioni e munizioni.

Ogni personaggio appartiene ad una razza che è caratterizzata da:

- Nome: identificatore univoco della razza;
- Fazione: indica l'alleanza della razza;
- Magia: indica il tipo di magia peculiare di quella razza;
- Mezzo di trasporto: indica il mezzo di trasporto prediletto da quella razza;
- Area provenienza: luogo da cui proviene la razza.

Ogni personaggio appartiene anche ad una classe che include avente come attributi:

- Nome: identificativo univoco della classe;
- Risorsa: risorsa utilizzata dalla classe per utilizzare magie; (es: mana, energia);
- Ruolo: tipo di ruolo all'interno del gioco.

Ogni classe ha delle specializzazioni peculiari in cui si memorizza:

- Nome: identificatore della specializzazione;
- Tipo: tipo della specializzazione (es: danno, cura, scudo).

2.2 Glossario dei termini

Termine	Descrizione	Sinonimi	Collegamenti
server	componente hardware/software che gestisce il mondo di gioco e conserva i dati dei giocatori		reame
reame	sottocomponenti in cui si divide ogni server. Ogni server è costituito da più reami		server, giocatore
giocatore	persona reale che gioca	utente	reame, clan, messaggio, lobby, personaggio, missione
clan	Insieme di più giocatori	gilda	giocatore
messaggio	Messaggio che utilizzano gli utenti per comunicare tra di loro		giocatore
lobby	Gruppo di persone che si riuniscono in una "stanza virtuale" per giocare insieme	gruppo	giocatore, missione
personaggio	Personaggio che permette all'utente di agire nel mondo virtuale di gioco	avatar	giocatore, razza, professione, inventario, classe
professione	Impiego del personaggio	attività, impiego	personaggio
missione	incarichi assegnati dal gioco che danno ricompense		giocatore, lobby
inventario	Insieme degli oggetti posseduti dal personaggio		personaggio, arma
arma	Comprende le armi bianche e le armi a distanza del personaggio		inventario
razza	razza del personaggio		personaggio
classe	specialità del personaggio (es: mago, monaco)		personaggio, specializzazione

specializzazione	Talento particolare della classe in questione		classe

2.3 Operazioni previste sulla base di dati

OPERAZIONI	TIPO	FREQUENZA
1.Registrazione nuovo giocatore	I	200 volte alla settimana
2.Login giocatore	I	1000 volte al giorno
3.Richiesta di amicizia tra due giocatori	I	250 volte al giorno
4.Spostamento di clan	I	45 volte al giorno
5.Stampa numero di membri appartenenti a un clan	I	80 volte al giorno
6.Creazione di una lobby	I	300 volte al giorno
7.Ingresso in una lobby	B	1500 volte al giorno
8.Creazione di un personaggio	I	400 volte alla settimana
9.Ottenimento oggetto	B	2500 volte al giorno
10.Ottenimento stipendio per professione	B	1 volta alla settimana

3. Progettazione concettuale

3.1 Analisi delle entità

Si riporta una lista delle entità della base di dati con i relativi identificatori primari sottolineati.

- **Server :** rappresenta un server di gioco
  - IPserver : int , univoco, not null
  - Nome: varchar, not null
  - Regione: varchar, not null
  - Capienza: int, not null
  - Stato: enum{online, offline}, not null, default ‘offline’

  Ogni server può essere di due tipi:
  - ➢ Server pvp: un giocatore si confronta e agisce in un server costituito da altri giocatori
  - ➢ Server pve: il giocatore si confronta con altri personaggi controllati dall’intelligenza artificiale

- **Reame:** rappresenta un reame di gioco
  - IDreame : int , univoco, not null
  - Status: enum{online,offline}, not null, default ‘offline’
  - Popolazione: int, not null

- **Giocatore:** rappresenta un giocatore
  - Username : varchar , univoco, not null
  - Stato: enum{online, offline}, not null, default ‘offline’
  - Ping: smallint, not null

- **Clan:** rappresenta un clan
  - NomeClan: varchar ,univoco, not null
  - N_membri: int, not null

- Tag: varchar, not null

- **Messaggio:** rappresenta un messaggio
  - ID: int ,univoco, not null
  - Testo: varchar, not null
  - Data: date, not null

- **Lobby:** rappresenta una lobby
  - IDlobby: int ,univoco, not null
  - Proprietario: varchar, not null
  - Data_creazione: date, not null

- **Personaggio:** rappresenta un personaggio
  - IDpersonaggio: int ,univoco, not null
  - IDpersonaggio: varchar, not null
  - Potenza: smallint, not null
  - Monete: int, default 0
  - Esperienza: int, default 0
  - Punti_vita: int, default 750

- **Professione:** rappresenta una professione
  - Nome: varchar ,univoco, not null
  - Guadagno: int, default 0

- **Missione:** rappresenta una missione
  - Codice: int ,univoco, not null
  - Tipologia: varchar, not null
  - Ricompensa: varchar, default “500 oro”, not null
  - A_tempo: bool, default “no”, not null

Le missioni possono essere svolte in due modi:

- Missioni singleplayer(giocatore singolo)
- Missioni multiplayer(tra più giocatori):
  - N.membri: smallint, not null

- **Inventario:** rappresenta un inventario
  - IDpersonaggio: int, univoco, not null
  - Nome_oggetto: varchar, univoco, not null
  - N_oggetti: smallint, default 1

- **Arma:** rappresenta un’arma
  - IDarma: int, univoco, not null
  - Nome: int, varchar, not null
  - Danno: int, not null

Le armi possono essere di due tipi:

- Armi bianche
- Armi a distanza:
  - Gittata: int, not null
  - Max Munizioni int, default 30
  - Munizioni int, not null

- **Razza:** rappresenta una razza
  - Nome: varchar, univoco, not null
  - Fazione:enum{orda, alleanza}, not null
  - Mezzo_di_trasporto: varchar, not null
  - Magia: varchar, not null
  - Area_provenienza: varchar, not null

- **Classe:** rappresenta una classe
  - Nome: varchar, univoco, not null
  - Risorsa: varchar, default “mana”
  - Ruolo: varchar, not null

- **Specializzazione:** rappresenta una specializzazione
  - Nome: varchar, univoco, not null
  - Tipo: varchar, not null

Approfondimenti

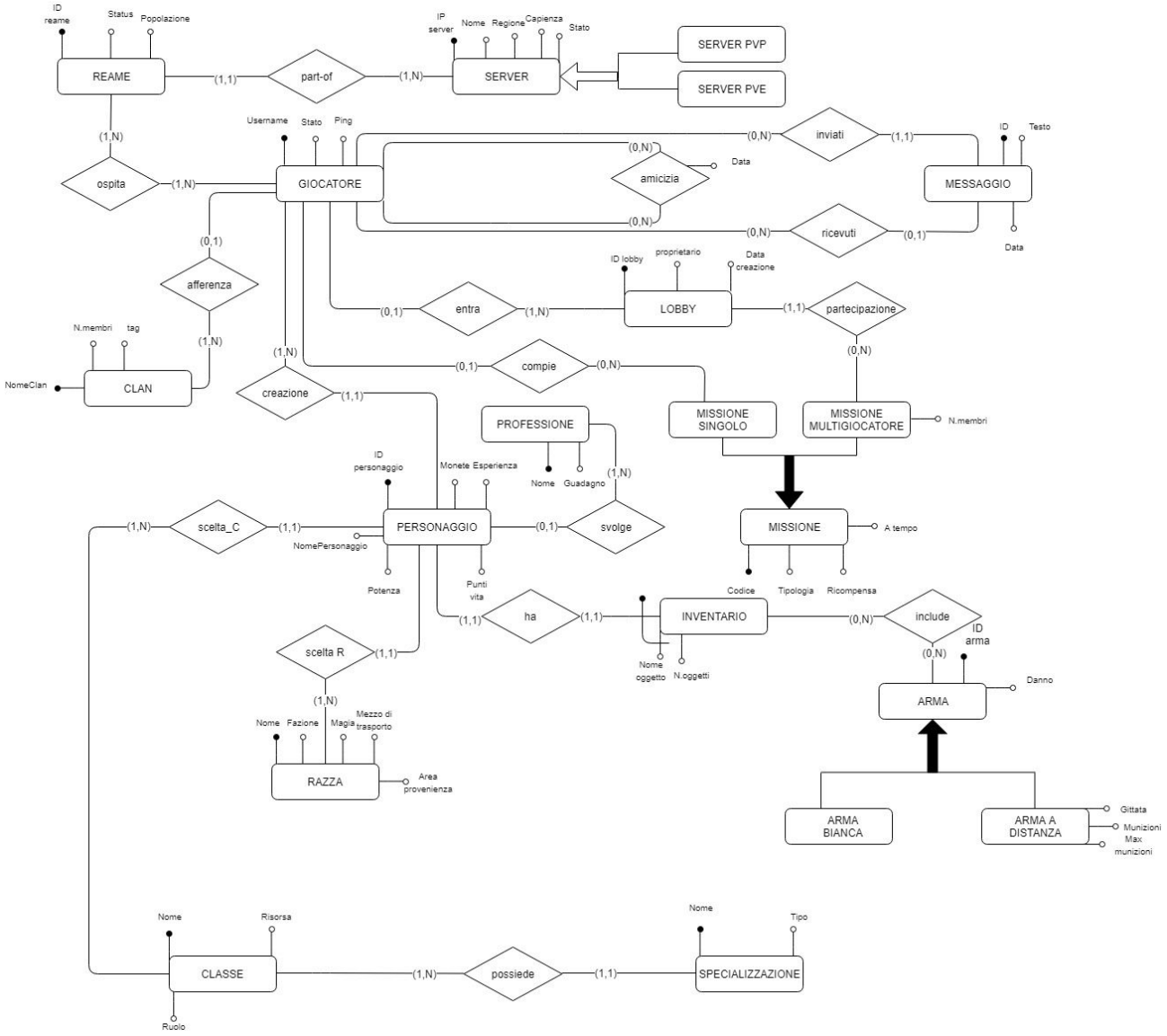
Generalizzazioni

- ➔ Server è generalizzazione parziale ed esclusiva di: server pve, server pvp. (Altri tipi di server potrebbero essere aggiunti in futuro)
- ➔ Missione è generalizzazione completa ed esclusiva di: missione singolo, missione multigiocatore.
- ➔ Arma è generalizzazione completa ed esclusiva di: arma bianca, arma a distanza.

3.2 Analisi delle relazioni

- Server - Reame: part-of
  - Un server è costituito da uno o più reami (1,N);
  - Un reame deve far parte di un solo server (1,1).
- Reame - Giocatore: ospita
  - Un reame ospita uno o più giocatori (1,N);
  - Un giocatore può far parte di uno o più reami (1,N).
- Giocatore - Giocatore: amicizia
  - Un giocatore può avere da zero a N amici (0,N);
  - Un giocatore può essere amico di zero o N altri giocatori (0,N).
- Giocatore - Messaggio: inviati
  - Un giocatore può non inviare messaggi, o inviarne N (0,N);
  - Un messaggio deve essere inviato da un solo giocatore (1,1).
- Giocatore - Messaggio: ricevuti
  - Un giocatore può non ricevere messaggi, o riceverne N (0,N);
  - Un messaggio può essere ricevuto da un solo giocatore (0,1).
- Giocatore - Clan: afferenza
  - Un giocatore può scegliere se far parte o meno di un clan (0,1);
  - Un clan può essere composto da uno o più giocatori (1,N).
- Giocatore - Personaggio: creazione
  - Un giocatore deve creare uno o più personaggi (1,N);
  - Un personaggio deve essere creato da un solo giocatore (1,1).
- Giocatore - Lobby: entra
  - Un giocatore può decidere di entrare in una sola lobby alla volta (0,1);
  - In una lobby devono essere entrati almeno uno o più giocatori (1,N).
- Lobby - Missione Multiplayer: partecipazione
  - Una lobby deve partecipare ad una missione multiplayer (1,1);
  - Una missione multiplayer può avere da zero a N lobby partecipanti (0,N).
- Giocatore - Missione Singolo: compie
  - Un giocatore può decidere se compiere o meno ad una missione in modalità Singolo (0,1);
  - Una missione in modalità Singolo può essere compiuta da zero a N giocatori (0,N).
- Personaggio - Professione: svolge
  - Un personaggio può svolgere o meno una sola professione (0,1);
  - Una professione può essere svolta da uno o più personaggi (1,N).
- Personaggio - Classe: scelta_C
  - Ogni personaggio deve scegliere una sola classe (1,1);
  - Una classe può essere scelta da uno o più personaggi, (1,N).
- Classe - Specializzazione: possiede
  - Ogni classe possiede una o più specializzazioni (1,N);
  - Ogni specializzazione è relativa a una singola classe (1,1);
- Personaggio - Razza: scelta_R
  - Ogni personaggio deve scegliere una sola razza (1,1);
  - Una razza può essere scelta da uno o più personaggi, (1,N).
- Personaggio - Inventario: ha
  - Ogni personaggio possiede un solo inventario (1,1);
  - Ogni inventario appartiene ad un solo personaggio (1,1).
- Inventario - Arma: include
  - In un inventario può essere presente un numero da zero a N di armi (0,N);
  - Un arma può essere presente in un numero da zero a N di inventari (0,N).

3.3 Modello Concettuale



Lista dei vincoli
1. Un giocatore può appartenere a più reami ma dello stesso server. 2. Un reame deve appartenere ad un unico server. 3. Un giocatore deve essere online per appartenere ad una lobby o per partecipare ad una missione in singolo. 4. In ogni lobby deve essere sempre presente almeno un giocatore. 5. In ogni lobby ci deve essere una missione associata. 6. Ad ogni personaggio è adibito unicamente l'uso del proprio inventario. 7. Un personaggio può svolgere al massimo una professione. 8. Se il giocatore A è amico del giocatore B allora il giocatore B è amico del giocatore A. 9. Un clan deve contenere almeno un giocatore.

4. Progettazione logica

4.1 Analisi di ridondanza

All'interno dello schema concettuale, è presente un attributo ridondante, ovvero N.membri che si trova all'interno dell'entità Clan. Per valutare se sia necessario o no mantenere questo attributo esamineremo le operazioni 4 e 5 della tabella in 2.3.

- Spostamento di clan (45 volte al giorno).
- Stampa del numero di membri appartenenti ad un clan (80 volte al giorno).

Tabella dei Volumi

Concetto	**Tipo**	**Volume**
Giocatore	E	1100
Afferenza	R	30
Clan	E	200

OPERAZIONE 1:

CON RIDONDANZA		
Concetto	**Accesso**	**Tipo**
Giocatore	2	L/S
Afferenza	1	L
Clan	2	S

CON RIDONDANZA: In presenza di ridondanza si ha un accesso in lettura all'entità giocatore per cercare la persona che desidera effettuare il cambio clan e il nome del vecchio clan (il nome del nuovo clan lo si conosce già perchè deriva da una richiesta dell'utente) e un accesso in scrittura per aggiornare l'attributo Clan su giocatore. Successivamente si effettua una lettura dell' associazione afferenza (giocatore-clan) per cercare l'occorrenza corrispondente in Clan. Si effettua quindi un accesso in scrittura per aggiornare Nmembri del nuovo clan a Nmembri+1 e un altro accesso in scrittura per aggiornare Nmembri del vecchio clan a Nmembri-1. Considerando che un' operazione in scrittura costa il doppio di una in lettura si ha che:

- (2 accessi lettura) * 45 volte al giorno + (3*2 accessi in scrittura) * 45 volte al giorno = 360 accessi in lettura e scrittura al giorno.

OPERAZIONE 2:

CON RIDONDANZA		
Concetto	**Accesso**	**Tipo**
Clan	1	L

CON RIDONDANZA: In presenza di ridondanza si ha un solo accesso in lettura per leggere il numero di

membri di un clan e stamparlo. Quindi:

- (1 accesso in lettura)*80= 80 accessi in lettura al giorno

Cond ridondanza si hanno quindi un totale di accessi in lettura/scrittura pari a

- 360+80= 440 accessi in lettura/scrittura.

OPERAZIONE 1:

SENZA RIDONDANZA		
Concetto	**Accesso**	**Tipo**
Giocatore	1	L
Afferenza	1	L

SENZA RIDONDANZA: in assenza di ridondanza si ha un accesso in lettura all'entità giocatore per cercare la persona che desidera effettuare il cambio clan e il nome del vecchio clan (il nome del nuovo clan lo si conosce già perchè deriva da una richiesta dell'utente) e un accesso in scrittura per aggiornare l'attributo Clan su giocatore. Non è necessario effettuare altre operazioni in clan in quanto si assume che l'attributo Nmembri non sia presente. Si ottiene che:

- (1 accesso in lettura *45 volte al giorno) + (2 accessi in scrittura*45 volte al giorno)= 135 accessi in lettura/scrittura al giorno.

OPERAZIONE 2:

SENZA RIDONDANZA		
Concetto	**Accesso**	**Tipo**
Giocatore	1100	L

SENZA RIDONDANZA: In assenza di ridondanza si deve accedere all'attributo clan di ogni giocatore una volta in lettura per verificare se il giocatore faccia o no parte del clan, eventualmente aggiungendolo al conteggio. Avendo un volume di 1100 giocatori si hanno un totale di 1100 accessi in lettura. Quindi:

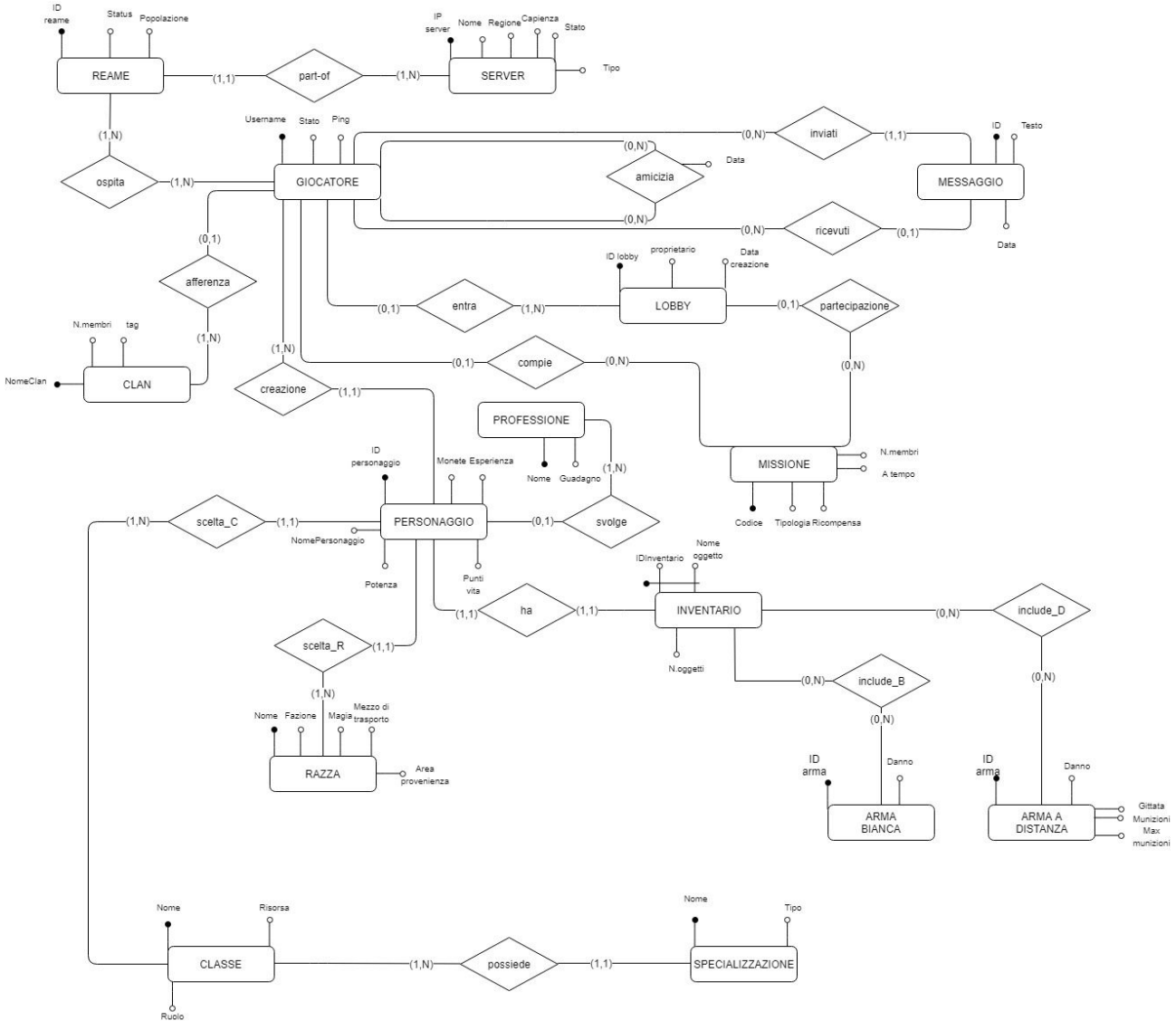
- (1100 accessi in lettura * 80 volte al giorno) = 88.000 accessi in lettura

Senza ridondanza si hanno quindi un totale di :

- 88.000+ 135= 88.135 accessi in lettura/scrittura.

Esaminando le due operazioni si hanno un totale 440 accessi in lettura/scrittura con ridondanza, mentre 88.135 accessi in lettura/scrittura. Data l'evidente differenza tra i due risultati si è optato per mantenere la ridondanza.

4.2 Schema E-R ristrutturato



Per effettuare la traduzione al modello logico è necessario prima realizzare il cosiddetto schema E-R ristrutturato al fine di facilitarne la traduzione. Lo schema E-R ristrutturato nella figura soprastante è stato ottenuto dallo schema E-R eseguendo le seguenti operazioni:

- Eliminazione delle generalizzazioni le quali non sono traducibili in alcun modo nel modello relazionale, che sono state quindi sostituite da altri costrutti (le scelte progettuali saranno commentate in seguito).
- Sostituzione di identificatori esterni con identificatori interni per semplificare lo schema.

ELIMINAZIONE DELLE GENERALIZZAZIONI

1. La generalizzazione parziale ed esclusiva di server su server pvp-server pve è stata eliminata.

È stato deciso di accorpare le due entità figlie nell'entità genitore e di aggiungere un campo tipo all' entità server. Il motivo della scelta progettuale è dato dalla mancata definizione di attributi distinti non solo tra le due entità figlie, ma anche fra quest'ultime e l'entità genitore. Ovviamente, essendo la generalizzazione non totale, un eventuale aggiornamento della base di dati potrebbe produrre valori nulli, ma allo stesso tempo ci assicura un minor numero di accessi rispetto alla struttura con generalizzazioni dove gli attributi sono sparsi tra le varie entità.

2. La generalizzazione totale ed esclusiva di missione su missione singolo/ missione multigiocatore è stata eliminata. È stato deciso di accorpare le due entità figlie nell'entità genitore per lo stesso motivo esposto in precedenza: missione singolo e missione multigiocatore avevano gli stessi attributi dell'entità missione, mentre si differenziavano per un attributo tra di loro. È stato deciso, quindi, di aggiungere l'attributo N. membri(che per le missioni singolo sarà nullo e quindi causerà spreco di memoria) all'entità missione. A differenza del caso precedente anche le relazioni cambiano: mentre prima giocatore era in relazione con missione singolo, e lobby con missione multigiocatore ora entrambe sono in relazione con la medesima entità missione.

3. La generalizzazione totale ed esclusiva di arma su arma bianca/ arma a distanza è stata eliminata. È stato deciso di accorpare il genitore nelle entità figlie. Questa scelta progettuale è stata dovuta alla presenza di più attributi distinti nelle entità figlie. Tale decisione diminuisce il numero di accessi totali in quanto non è più necessario accedere all'entità genitore ogni volta ed elimina la presenza di valori nulli. Ciò ha prodotto la creazione delle due nuove relazioni include_b ed include_d che mettono in relazione rispettivamente inventario con arma bianca e inventario con arma a distanza.

SCELTA DEGLI IDENTIFICATORI

Come di consueto si è deciso di adottare, per la maggioranza delle entità, degli identificatori univoci che possono essere ID o codici. È preferibile, infatti, aggiungere degli attributi che possano fungere da soli da chiavi, invece che selezionare insieme di attributi che potrebbero appesantire inutilmente la traduzione al modello relazionale. Tuttavia per alcune entità (es : clan, giocatore, razza, classe, specializzazione), si sono trovati degli identificatori già univoci nelle fase di progettazione e quindi si è deciso di utilizzarli direttamente come chiavi. L'identificatore esterno NomeOggetto di inventario che coinvolge l'entità personaggio è stato trasformato in un identificatore interno aggiungendo IDInventario all'entità e costituiscono insieme la chiave.

4.3 Progettazione logica

Server(IPserver, Nome, Regione, Capienza, Stato, Tipo)
Reame(IDreame, Status, Popolazione, Server*)
Vincolo di integrità referenziale tra Server in Reame e IPserver in Server
Ospita(IDreame*, Username*)
Vincolo di integrità referenziale tra IDreame in Ospita e IDreame in Reame
Vincolo di integrità referenziale tra Username in Ospita e Username in Giocatore
Giocatore(Username, Stato, Ping, Clan*, Lobby*, MissioneInCorso*)
Vincolo di integrità referenziale tra Clan in Giocatore e NomeClan in Clan
Vincolo di integrità referenziale tra MissioneInCorso in Giocatore e Codice in Missione
Amicizia(Username1*, Username2*, Data)
Vincolo di integrità referenziale tra Username1 in Amicizia e Username in Giocatore
Vincolo di integrità referenziale tra Username2 in Amicizia e Username in Giocatore
Clan(NomeClan, N_membri, Tag)
Messaggio(ID, Testo, Data, Mittente*, Destinatario*)
Vincolo di integrità referenziale tra Mittente in Messaggio e Username in Giocatore
Vincolo di integrità referenziale tra Destinatario in Messaggio e Username in Giocatore
Lobby(IDlobby, Proprietario, Data_creazione, Missione*)
Vincolo di integrità referenziale tra Missione in Lobby e Codice in Missione
Missione(Codice, Tipologia, Ricompensa, N_membri, A_tempo)

Personaggio(IDpersonaggio, NomePersonaggio, Monete, Esperienza, Potenza, Punti_vita, Giocatore*, Professione*, Razza*, Classe*, Inventario*)
Vincolo di integrità referenziale tra Giocatore in Personaggio e Username in Giocatore
Vincolo di integrità referenziale tra Professione in Personaggio e Nome in Professione
Vincolo di integrità referenziale tra Razza in Personaggio e Nome in Razza
Vincolo di integrità referenziale tra Classe in Personaggio e Nome in Classe
Vincolo di integrità referenziale tra Inventario in Personaggio e IDInventario in Inventario
Professione(Nome, Guadagno)
Razza(Nome, Fazione, Magia, Mezzo_di_ trasporto, Area_ provenienza)
Classe(Nome, Risorsa, Ruolo)
Specializzazione(Nome, Tipo, Classe*)
Vincolo di integrità referenziale tra Classe in Specializzazione e Nome in Classe
Inventario(IDInventario, NomeOggetto, N_oggetti)
Arma_bianca(IDarma, danno)
Include_b(IDInventario*, NomeArma*,IDarma*)
Vincolo di integrità referenziale tra IDInventario in Include_b e IDInventario in Inventario
Vincolo di integrità referenziale tra NomeArma in Include_b e NomeOggetto in Inventario
Vincolo di integrità referenziale tra IDarma in Include_b e IDarma in Arma bianca
Arma_a_distanza(IDarma, Danno, Gittata, Munizioni, MaxMunizioni)
Include_d(IDInventario*, NomeArma*, IDarma*)
Vincolo di integrità referenziale tra IDInventario in Include_d e IDInventario in Inventario
Vincolo di integrità referenziale tra NomeArma in Include_d e NomeOggetto in Inventario
Vincolo di integrità referenziale tra IDarma in Include_d e IDarma in Arma a distanza

NB: * indica le chiavi esterne

5. QUERY, PROCEDURE, TRIGGER
5.1 Query

1. Contare il numero di oggetti totali contenuti nell'inventario di ciascun personaggio considerando le relative quantità:

```
SELECT p.IDpersonaggio, p.NomePersonaggio, sum(N_oggetti) AS oggetti_totali
FROM Personaggio p, Inventario i
WHERE p.Inventario= i.IDinventario
GROUP BY p.IDpersonaggio, p.NomePersonaggio
ORDER BY oggetti_totali
```

OUTPUT:

IDpersonaggio	NomePersonaggio	oggetti_totali
4567	Kailu	11
3541	Garrik	12
7648	Karfu	12
1346	Alais	14
7542	Meriel	16
3542	Ayen	17
6479	Zarod	19
4613	Morbash	25

2. Stampare la lista di giocatori ed i relativi personaggi inclusa la razza appartenenti al reame 0 che hanno creato almeno un personaggio guaritore:

```
SELECT  g.Username, p.NomePersonaggio, p.razza
FROM reame r JOIN ospita o ON r.IDreame=o.IDreame JOIN giocatore g ON g.Username=o.Username
JOIN personaggio p ON p.Giocatore=g.Username JOIN classe c ON p.Classe=c.nome
WHERE r.IDreame=0 AND C.Ruolo='guaritore'
```

OUTPUT:

Username	NomePersonaggio	razza
Pippo	Garrik	Worg
Pluto	Kailu	Argoniano

3. Stampare I nomi dei personaggi e dei giocatori che li hanno creati che possiedono almeno un personaggio avente due armi a distanza o due armi bianche:

Per questa query è necessario crearsi una vista che isoli gli IDInventario su Inventario in modo che le operazioni di conteggio avvengano una sola volta:

```
CREATE VIEW inv AS
SELECT DISTINCT IDinventario
FROM inventario
```

```
SELECT g.Username, p.NomePersonaggio
FROM giocatore g JOIN personaggio p on p.Giocatore=g.Username JOIN inv i on i.
IDinventario=p.Inventario JOIN include_b ib on ib.IDInventario=i.IDinventario
GROUP BY g.Username, p.NomePersonaggio
HAVING count(ib.IDinventario) > 1
UNION
SELECT g.Username, p.NomePersonaggio
FROM giocatore g JOIN personaggio p on p.Giocatore=g.Username JOIN inv i on
i.IDinventario=p.Inventario JOIN include_d id on id.IDInventario=i.IDinventario
GROUP BY g.Username, p.NomePersonaggio
HAVING count(id.IDinventario)> 1
```

OUTPUT:

Username	NomePersonaggio
TeamLeader	Zarod
Arthas007	Alais

4. Visualizzare il ping massimo e il ping medio dei giocatori che sono online e giocano nei reami 0 o 1, mostrando anche il nome e l'IP del server a cui questi reami appartengono.

```
SELECT s.IPserver, s.Nome, MAX(g.Ping) AS Maxping, AVG(g.Ping) AS pingMedio
FROM server s, giocatore g, ospita o, reame r
WHERE (r.Server= s.IPserver) AND (g.Username=o.Username AND r.IDreame=o.IDreame) AND
```

```
(o.IDreame=0 OR o.IDreame=1) AND (g.Stato='online')
GROUP BY s.IPserver, s.Nome
```

OUTPUT:

IPserver	Nome	Maxping	pingMedio
0	Azeroth	21	10.6000

5. Mostrare il nome, i punti vita e la potenza dei personaggi della fazione='orda' che svolgono una professione con guadagno>=200 visualizzandoli in ordine decrescente rispetto al guadagno.

```
SELECT p.NomePersonaggio, p.Potenza, p.Punti_vita, po.Nome, po.Guadagno
FROM personaggio p, razza r, professione po
WHERE p.Professione=po.Nome AND p.Razza= r.Nome AND Fazione='orda' AND po.Guadagno>= 200
ORDER BY po.Guadagno DESC
```

OUTPUT:

NomePersonaggio	Potenza	Punti_vita	Nome	Guadagno
Ayen	17	660	Mercante	380
Garrik	12	750	Mercenario	200
Zarod	17	468	Mercenario	200

6. Operazione 2 :Procedura che permette il login di un giocatore che deve essere offline (setta a online il suo campo stato del giocatore e setta a un valore di default (30) il ping del giocatore):

```
DELIMITER |
CREATE PROCEDURE login ( User varchar(255))
BEGIN
   DECLARE p varchar(50);
   SELECT g.Stato INTO p
   FROM giocatore g
   WHERE g.Username=User;
   IF ( p= 'offline') THEN
   update giocatore
            set Stato= 'online'
            where Username=User;
  update giocatore
            set Ping=30
            where Username=User;
   END IF;
END|
DELIMITER ;
```

7. Procedura che permette il logout di un giocatore (setta a offline il campo stato del giocatore user, setta a un valore di default(0) il ping del giocatore e mette a NULL a i campi Lobby e MissioneInCorso):

```
DELIMITER | CREATE PROCEDURE logout ( User varchar(255))
BEGIN
  DECLARE p varchar(50);
  SELECT g.Stato INTO p
  FROM giocatore g
  WHERE g.Username=User;
  IF ( p= ‘online’’) THEN
        update giocatore
          set Stato= 'offline'
          where Username=User;
          update giocatore
          set Ping= 0, Lobby=NULL, MissioneInCorso=NULL
          where Username=User;
 END IF;
END|
DELIMITER ;
```

8. Operazione 4 : Procedura che permette lo spostamento di clan. Se il clan non esiste ancora crea un nuovo clan di cui setta un tag predefinito. Se si inserisce un oldClan e un clanName che non coincidono viene prodotto un messaggio d’errore:

```
DELIMITER |
CREATE PROCEDURE Newclan ( clanName varchar(255), giocatore varchar(255), oldClan varchar(255))
BEGIN
 DECLARE m varchar(50);
 DECLARE n varchar(50);
 DECLARE CodErr CONDITION FOR SQLSTATE ’45000’;
 set @err_mess = ’il giocatore non appartiene a quel clan’;
 SELECT g.Clan into m
 FROM giocatore g
 where g.Username= giocatore;
 IF ( m <> oldClan) THEN
          SIGNAL CodErr SET MESSAGE_TEXT= @err_mess;
ELSE
          SELECT NomeClan into m
          FROM Clan
          WHERE NomeClan=ClanName;
          Update clan
          set Nmembri=Nmembri-1
          where nomeClan=oldClan;
          IF m is NOT NULL THEN
                    update Clan
                   set Nmembri=Nmembri +1
                   where nomeClan=ClanName;
                   update giocatore
                   set clan=clanName
                    where Username=giocatore;
          ELSE
                   insert into Clan values ( ClanName, 1, "tag predefinito");
                    Update giocatore
                    set clan=clanName
                   where Username=giocatore;
          END IF;
          END IF;
 END|
 DELIMITER ;
```

9. Funzione che dato l'ID di un personaggio ne calcola il livello utilizzando un algoritmo:

```
DELIMITER |
 CREATE FUNCTION LivelloPersonaggio (IDPersonaggio int)
 RETURNS int
 BEGIN
   DECLARE n int;
    DECLARE m int;
 DECLARE r int;
 set r= 20;
 set m=0;
 SELECT Esperienza into n
 FROM personaggio p
 WHERE p.IDPersonaggio = IDPersonaggio;
 WHILE (n-r >=0) DO
          set n=n-r;
          set m=m+1;
          set r=r+5;
 END WHILE;
          RETURN m;
 END|
 DELIMITER ;
```

10. Funzione che dato l'username di un giocatore ne restituisce il personaggio con monete+ guadagno maggiore:

```
CREATE VIEW Ng AS
SELECT Giocatore, Monete+ Guadagno AS TOT
FROM personaggio p JOIN professione  on Professione= Nome
```

```
DELIMITER |
CREATE FUNCTION MaxSoldi ( Username varchar(255))
RETURNS int
BEGIN
          DECLARE N int;
          SELECT MAX(TOT) into N
          FROM Ng n, giocatore g
          WHERE n.Giocatore=g.Username AND  g.Username= Username;
          RETURN N;
END |
DELIMITER ;
```

11. Trigger che mette a 'offline' i campi di status dei reami appartenenti ad un server che per qualsiasi motivo va 'offline'. Attivare il trigger a. attiva anche il trigger b. il quale mette a 'offline' i campi Stato di tutti i giocatori nei reami che sono andati offline settando il campo Ping a 0 e i campi Lobby e MissioneInCorso a NULL:
Si è dovuto utilizzare un if in quanto mariadb/phpmyadmin non supporta statement del tipo:
before/after update of *Attributo_tabella* on *nome_tabella*
Altrimenti il trigger si sarebbe attivato a qualsiasi update.

a.
```
DELIMITER |
CREATE TRIGGER SetReami
AFTER UPDATE ON server
```

```
for each row
BEGIN
IF( old.Stato <> new.Stato) THEN
   update reame set reame.Status= ‘offline’
    where reame.Server= new.IPServer;
END IF;
END|
DELIMITER ;
```

b.
```
DELIMITER |
CREATE TRIGGER SetPlayersdown
AFTER UPDATE ON reame
for each ROW
BEGIN
IF(new.Status <> old.Status) THEN
    Update giocatore
    set Stato='offline', Ping=0, Lobby=NULL, MissioneInCorso=NULL
     where Username in  (SELECT O.Username
                         FROM ospita o
                         where o.IDReame= new.IDReame);
END IF;
END |
DELIMITER ;
```