**ELYSIUM**

**INDICE**

[**Descrizione 3**](#Descrizione)

[**Glossario dei termini 5**](#GlossarioTermini)

[**Requisiti e specifiche 7**](#Requisiti)

[**Schema E-R Logico 11**](#LOGICO)

[**Elenco entità 12**](#entità)

[**Elenco Relazioni 13**](#relazioni)

[**Schema E-R Fisico 16**](#FISICO)

[**Creazione tabelle 17**](#CREAZIONE)

[**Operazioni di insert 21**](#INSERIMENTI)

[**Query SQL 24**](#QUERY)

[**Query Algebra/Calcolo Relazionale 35**](#ALGEBRACALCOLO)

[**Parti ulteriori 36**](#ULTERIORI)

[**Creazione utenti** 36](#UTENTI)

[**Stored Procedures** 37](#PROCEDURES)

[**Trigger** 39](#TRIGGERS)

**[Procedures](#EVENTS)** [44](#EVENTS)

[**Events** 44](#EVENTS)

[**Transactions** 44](#TRANSACTIONS)

[**NoSQL**](#MONGODB) **46**

[**Definizione struttura** 46](#DefinizioneStrutturaMongo)

[**Operazioni su collections** 48](#OperazioniSuCollections)

[**Confronto MongoDB e MySQL**](#Confronto)53

**Descrizione**

L’intento è quello di progettare un database relazionale per la gestione delle informazioni in un videogioco MMORPG (*Massive(ly) Multiplayer Online Role-Playing Game*) ovvero un videogioco di ruolo che si svolge esclusivamente su Internet in cui migliaia di persone reali si ritrovano contemporaneamente nello stesso ambiente virtuale.

Il videogioco in questione è un fantasy basato sull’avventura, lo svolgimento di missioni e la ricerca di oggetti all’interno di una vasta mappa dalle caratteristiche fantastiche e medievali.

Ogni nuovo utente che vorrà entrare nel mondo di gioco dovrà registrarsi inserendo nome, cognome, username, e-mail, password e data di nascita. Una volta registrati, i videogiocatori possono accedere attraverso il loro username e la password scelta. Si vuole tenere traccia degli accessi dei giocatori, memorizzando l’indirizzo IP e l’orario, per evitare che più giocatori accedano contemporaneamente attraverso lo stesso account.

Completato l’accesso entreranno nel gioco vestendo i panni di diversi personaggi personalizzabili appartenenti ognuno ad una specifica razza (Elfo, Halfling, Nano, Umano, Dragonide, Dragonide, Mezzelfo, Mezzorco, Tiefling) e ad una delle classi messe a disposizione (Barbaro, Bardo, Chierico, Druido, Guerriero, Ladro, Mago, Monaco, Paladino, Ranger, Stregone", Warlock) che rappresentano la “professione” del personaggio. Queste categorie sono brevemente descritte per facilitare la scelta all’utente.

Ogni personaggio ha delle statistiche che lo caratterizzano; esse si basano su campi numerici predefiniti quali: livello, salute attuale, salute massima, mana attuale, mana massimo, difesa, forza, destrezza, costituzione, intelligenza, saggezza, carisma, tempra, furtività.

Gli avatar sono inoltre caratterizzati da un indicatore numerico detto “livello”, all’aumentare del quale si ottiene la possibilità di utilizzare abilità prima bloccate da un vincolo di livello minimo. I livelli sono 255 e per passare al livello successivo bisogno accumulare una determinata quantità di esperienza ottenibile completando incarichi e missioni.

Ad ogni combinazione classe/razza corrisponde un insieme dedicato di abilità e oggetti utilizzabili: un accoppiamento Umano-Mago avrà una vasta collezione di abilità magiche a disposizione mentre altre coppie, come ad esempio un Elfo-Guerriero, potranno avere un set di abilità ridotto ma un’ampia gamma di oggetti dedicati.

L’utilizzo di un oggetto o un’abilità può influire sulle statistiche del personaggio che lo utilizza o di altri giocatori nei dintorni incrementandone o decrementandone il valore per un periodo limitato di tempo.

Per poter utilizzare un’abilità, un’utente deve averla precedentemente equipaggiata ma per poterlo fare potrebbe essere necessario rispettare i vincoli di classe di appartenenza e livello attuale del personaggio. Ogni razza, invece, garantisce un’abilità, detta “innata”, sempre utilizzabile dal personaggio. Il numero massimo di abilità equipaggiabili, esclusa quella innata è pari a 20.

Ogni oggetto appartiene ad una categoria che ne determina l’utilità: armature, armi, pozioni, pergamene, generi alimentari, abiti, missione, varie. Ne esistono di utilizzabili e vendibili (armi, armature, pergamene, generi alimentari, abiti), solo vendibili (varie), non utilizzabili e non vendibili (missione, necessari per portare a termine, appunto, una missione). Gli strumenti posseduti da un giocatore sono visualizzabili attraverso un menù detto “inventario” nel quale viene memorizzato il nome dell’oggetto e la quantità disponibile per quel giocatore.

Gli Oggetti sono sempre seguiti da una breve descrizione ed è sempre specificato su chi possono essere utilizzati (sé, alleato singolo, alleato gruppo(5m), alleato gruppo(10m), nemico singolo, nemico gruppo(5m), nemico gruppo(10m)).

Essi possono essere guadagnati completando missioni o acquistandoli in apposite attività commerciali. Queste attività, visibili su una mappa bidimensionale disponibile per tutti i giocatori, sono di diverse specie (taverne, negozi, botteghe, venditori ambulanti), ognuno di essi ha quindi un ventaglio di prodotti che può vendere i cui prezzi possono variare in base all’attività.

Esistono diversi tipi di missioni con un diverso grado di difficoltà, le quali, una volta concluse, forniscono varie ricompense ai giocatori, come oggetti, monete o esperienza, di valore direttamente proporzionale alla difficoltà dell’incarico.

Un incarico, per essere avviato, può presentare alcuni vincoli, quali: numero minimo e numero massimo di partecipanti e missioni da completare necessariamente prima di avviare quella in questione (se esistono).

Per accettare una missione bisogna recarsi in uno degli specifici luoghi di interesse presenti nel mondo di gioco e segnati sulla stessa mappa precedentemente citata; le missioni hanno quindi posizioni prestabilite dove possono essere avviate. Possono essere prese in carico più missioni contemporaneamente.

Per favorire il gioco di squadra si è pensato di creare dei gruppi dove i giocatori possono riunirsi e progredire nell’avventura in compagnia di altri player. Ogni utente ne può aggiungere altri alla sua “lista amici”, così potersi rapportare più facilmente coi suoi compagni più stretti per completare missioni insieme ed esplorare la mappa. Durante il gioco si può comunicare con altri utenti attraverso un’apposita chat. Questa può avere uno o più partecipanti.

All’interno del gioco c’è la possibilità di creare delle compagnie dette “Gilde” gestite da un “Capo Gilda”, dove i giocatori ne entrano a far parte nei panni dei loro alter ego digitali e si riuniscono per completare specifiche missioni dedicate ai soli membri di una Gilda. Questo permette agli utenti di ottenere oggetti rari e grandi somme di denaro oltre che fare nuove conoscenze.

**Glossario dei termini**

|  |  |  |  |
| --- | --- | --- | --- |
| **Termine** | **Descrizione** | **Sinonimi** | **Collegamenti** |
| Utente | Colui che gioca al videogame | Giocatore, Videogiocatore, Player | * Accesso * Amici * Personaggio * Messaggio * Chat |
| Accesso | Ingresso di un utente nel gioco | // | * Utente |
| Messaggio | Messaggio inviato da un utente | // | * Utente * Chat |
| Chat | Conversazione tra utenti | // | * Utente * Messaggio |
| Personaggio | Avatar creato dall’utente | Avatar,  alter ego digitale | * Classe * Razza * Oggetti * Abilità * Missioni * Gilda * Livello |
| Statistica | Campo numerico che indica le capacità di un personaggio | // | * Personaggio * Abilità * Oggetto |
| Classe | Professione del personaggio | Categoria | * Personaggio * Abilità * Oggetto |
| Razza | Razza di appartenenza del personaggio | Categoria | * Personaggio * Abilità * Oggetto |
| Livello | Grado di avanzamento di un personaggio | Grado | * Personaggio * Abilità |
| Abilità | Capacità e potenziamenti di un personaggio | Capacità | * Personaggio * Livello * Classe * Razza * Statistiche |

|  |  |  |  |
| --- | --- | --- | --- |
| **Termine** | **Descrizione** | **Sinonimi** | **Collegamenti** |
| Missione | Incarico assegnato ad un giocatore che permette di accumulare esperienza, oggetti ed altre ricompense | Incarico | * Personaggio * Luogo di interesse * Oggetto |
| Oggetto | Insieme delle cose utilizzabili e collezionabili | strumento | * Personaggio * Classe * Razza * Missione * Negozio * Statistiche |
| Attività commerciale | Luogo dove acquistare e vendere oggetti | // | * Oggetto * Luogo di interesse |
| Luogo di Interesse | Elenco delle coordinate presso le quali sono presenti particolari attività o strutture | // | * Missione * Attività commerciale |
| Amico | Utente appartenente alla lista amici di un’altra persona | Compagno | * Utente |
| Gilda | Gruppi di avatar riunitisi in una compagnia | Compagnia | * Personaggio |

**Requisiti e specifiche**

La base di dati che si sta andando a sviluppare serve a supportare la struttura del videogioco online *“Elysium”* e il suo obiettivo principale è quello di memorizzare le informazioni essenziali per permettere a più utenti di giocare contemporaneamente in maniera equa e sicura. Per fare ciò si è pensato di memorizzare sia tutti i dati comuni a più utenti, sia i dati relativi ai progressi dei personaggi nella stessa base di dati.

La prima categoria di informazioni riguarda gilde, lista delle abilita, lista degli oggetti, attività commerciali, missioni e le loro posizioni; memorizzare queste informazioni aiuta a centralizzare tutti i dati che sono comuni a tutti gli utenti, così da evitare incongruenza tra ciò che utenti diversi vedono. Questi dati sono perlopiù statici, vengono aggiornati periodicamente dagli sviluppatori ma rimangono costanti per diversi giorni.

La seconda categoria invece è relativa ai dati di un utente e di un personaggio, come accessi, messaggi, chat, liste di amici, razza, classe, statistiche, inventario ed abilita equipaggiate. Occuparsi di queste informazioni in maniera centralizzata serve ad evitare eventuali problemi di consistenza e comportamenti scorretti da parte degli utenti (come, ad esempio, incremento delle monete disponibili o dei livelli) che potrebbero rendere il gioco meno equo. Questo insieme di dati è molto dinamico, ogni giocatore con una singola azione potrebbe modificarne diversi nello stesso momento.

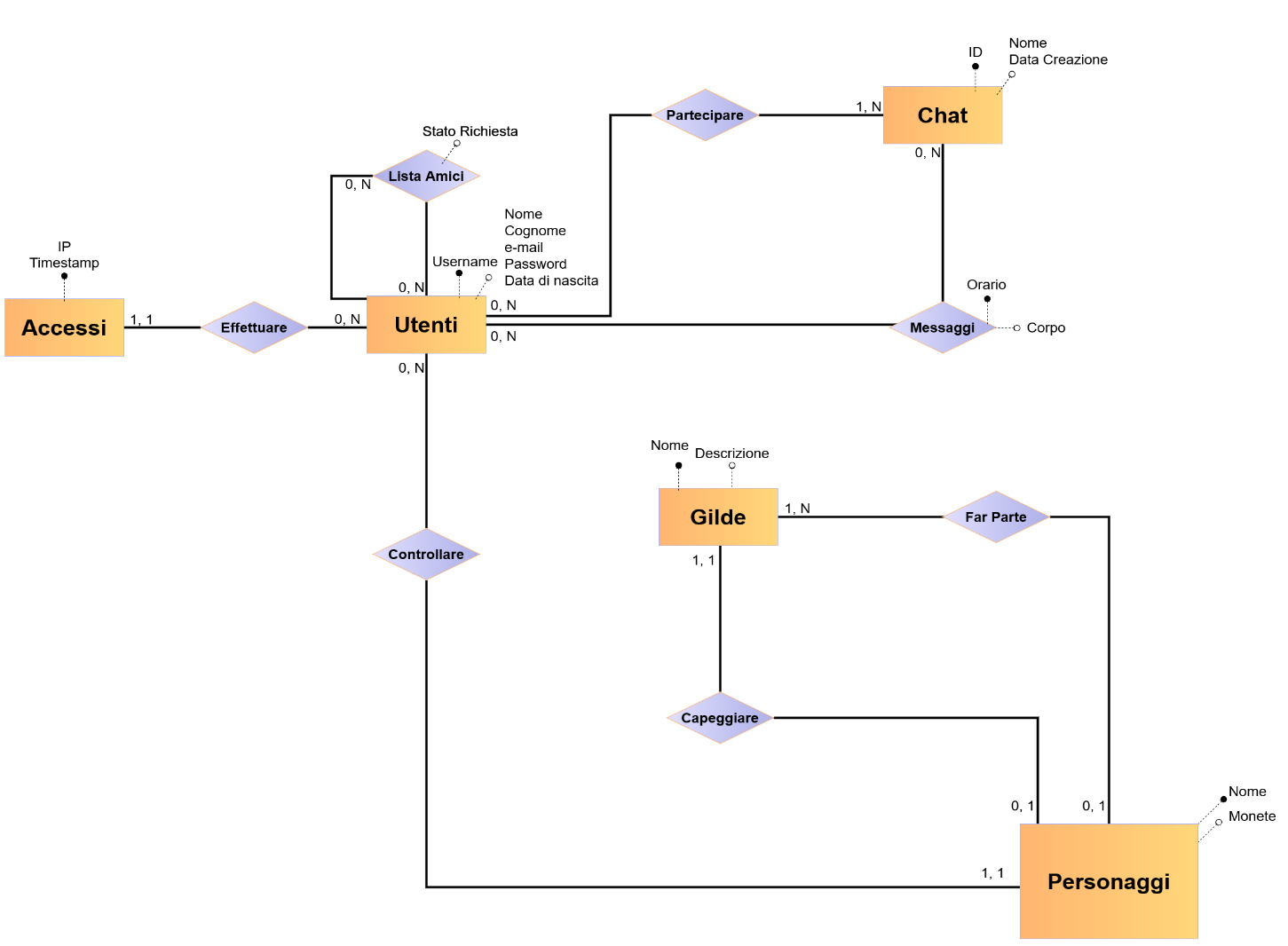
Fatte queste considerazioni e data la descrizione iniziale si è pensato costruire il database mettendo in relazione questi due gruppi di informazioni. Per fare ciò partiamo costruendo uno scheletro sulla base dei termini descritti nel *“glossario dei termini”* e costruiamo le relazioni tra essi.

Definiamo ora requisiti e casi particolari:

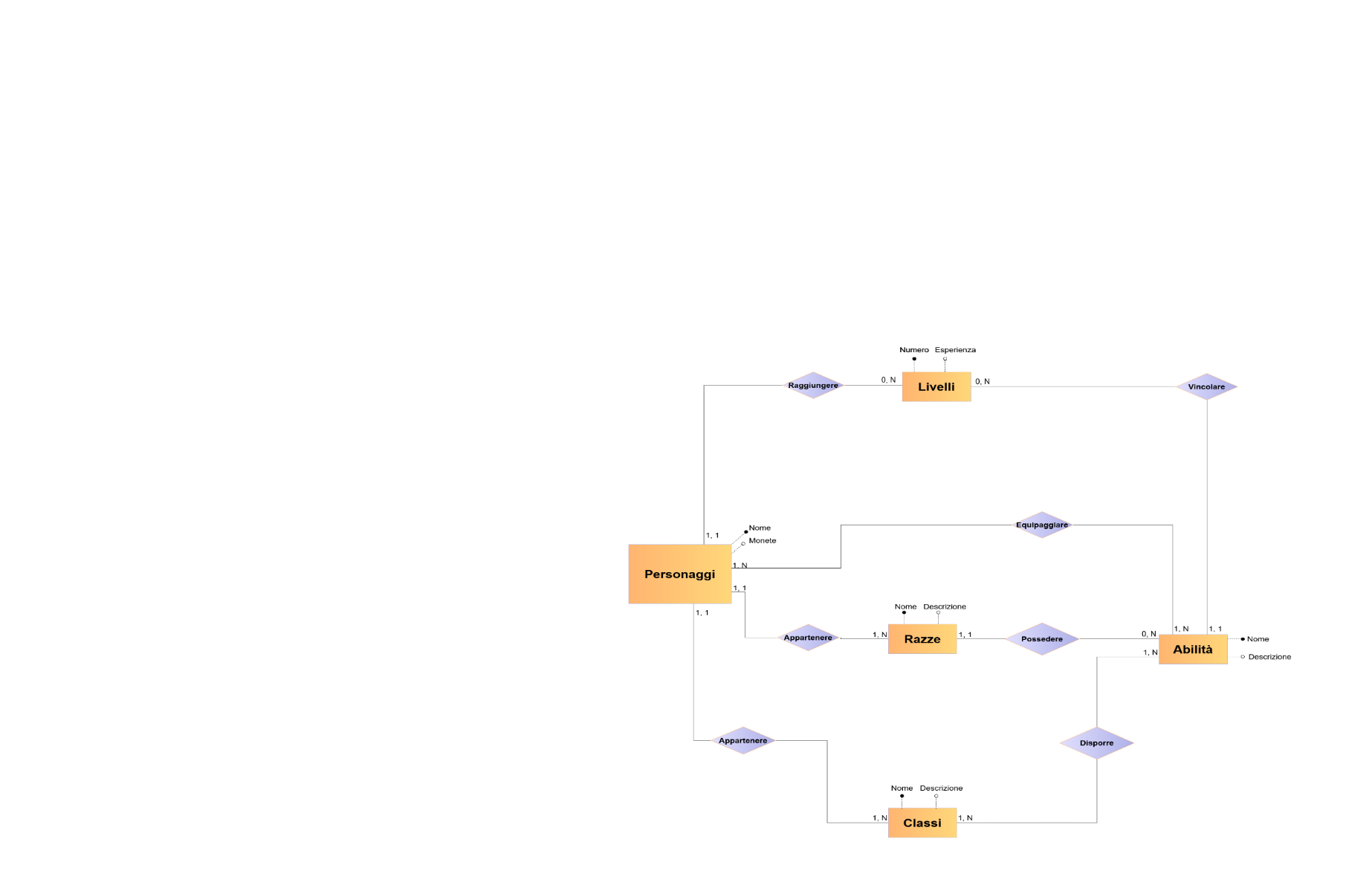
* La categoria “Amici” può essere descritta come una relazione tra due Utenti. Un utente può inviare una richiesta di amicizia, che prima di essere accettata viene mantenuta nel DB è caratterizzata da un valore booleano posto a 0, che verrà aggiornato a 1 nel momento in cui questa verrà confermata dall’utente destinatario della richiesta (questo per far sì che la richiesta non venga persa, ma rimanga visibile sia al mittente che al destinatario).

La relazione di amicizia si è deciso di rappresentarla come una relazione riflessiva: se l’utente A è amico dell’utente B anche l’utente B sarà amico dell’utente A.

* Un utente può partecipare ad una chat e può spedire dei messaggi destinati ad una di esse. Possiamo quindi immaginare l’entità messaggi come una relazione tra utenti e chat le cui n-uple sono rese univoche dall’orario di invio di un messaggio. Bisogna evitare che l’utente possa inviarne verso chat delle quali non fa parte.
* L’insieme delle gilde viene posto in relazione con quello dei personaggi, permettendo ai giocatori di appartenere a gilde diverse quando utilizzano personaggi diversi, così da differenziare il più possibile due avatar dello stesso utente. Un personaggio può far parte o meno di una gilda, ma quest’ultima deve obbligatoriamente avere un capo.



* Costruiamo l’entità livelli, caratterizzata da numero livello ed esperienza necessaria per raggiungerlo per mantenere consistenza e scalabilità. Il livello può caratterizzare un personaggio e allo stesso tempo vincolare un’abilità.
* Costruiamo anche le entità razze e classi, identificate dal loro nome e seguite da descrizione. Ciò rende il sistema più scalabile in caso di inserimenti o aggiornamenti futuri. Il limite di abilità equipaggiate da un personaggio deve essere di al massimo 20 e si deve impedire venga equipaggiata un’abilità già presente tra le abilità innate dello stesso.



* L’entità statistiche è caratterizzata solo dal nome della statistica ma è molto importante rappresentarla. Avere a disposizione questa entità e non trasformare i suoi record in attributi delle tabelle collegate aiuta in diversi casi:
  1. In caso di inserimenti di nuove statistiche basta soltanto aggiornare l’entità e non modificare tutte quelle collegate ad essa;
  2. Le informazioni riguardanti gli effetti di oggetti e abilità rimangono centralizzate, facili da reperire e consistenti;
  3. Gli inserimenti e gli aggiornamenti sono facili e veloci, soprattutto nel caso in cui ci sono abilità e oggetti che influiscono su un numero arbitrario di statistiche.
* L’inventario di un personaggio può essere visto come la relazione tra quest’ultimo e gli oggetti presenti nel gioco. Un personaggio potrebbe non possederne, oppure un oggetto potrebbe non essere posseduto da nessun giocatore.
* Mettiamo in relazione un personaggio con le missioni da lui attivate e completate. Un personaggio potrebbe avere più missioni attive e/o completate e la stessa missione potrebbe essere stata presa in carico da più giocatori. Inserendo un campo “stato” in questa relazione, che ci notifica quando una missione viene completata facilità il calcolo delle ricompense e livelli guadagnati da ogni giocatore completando una missione. Le missioni relative alle gilde sono riconoscibili grazie ad un campo “*rivolto\_a*”.

Si richiede anche di ricalcolare il livello di un giocatore in base all’esperienza acquisita nel caso in cui venga completata una nuova missione;

* Una missione e un’attività commerciale possono trovarsi nello stesso luogo;
* Quest’ultime sono in relazione con gli oggetti disponibili nel gioco, ogni attività può vendere più oggetti, caratterizzati da una disponibilità attuale e una disponibilità massima. Due attività diverse potrebbero vendere lo stesso oggetto ad un prezzo diverso, mentre un oggetto potrebbe non essere venduto in nessuna attività.

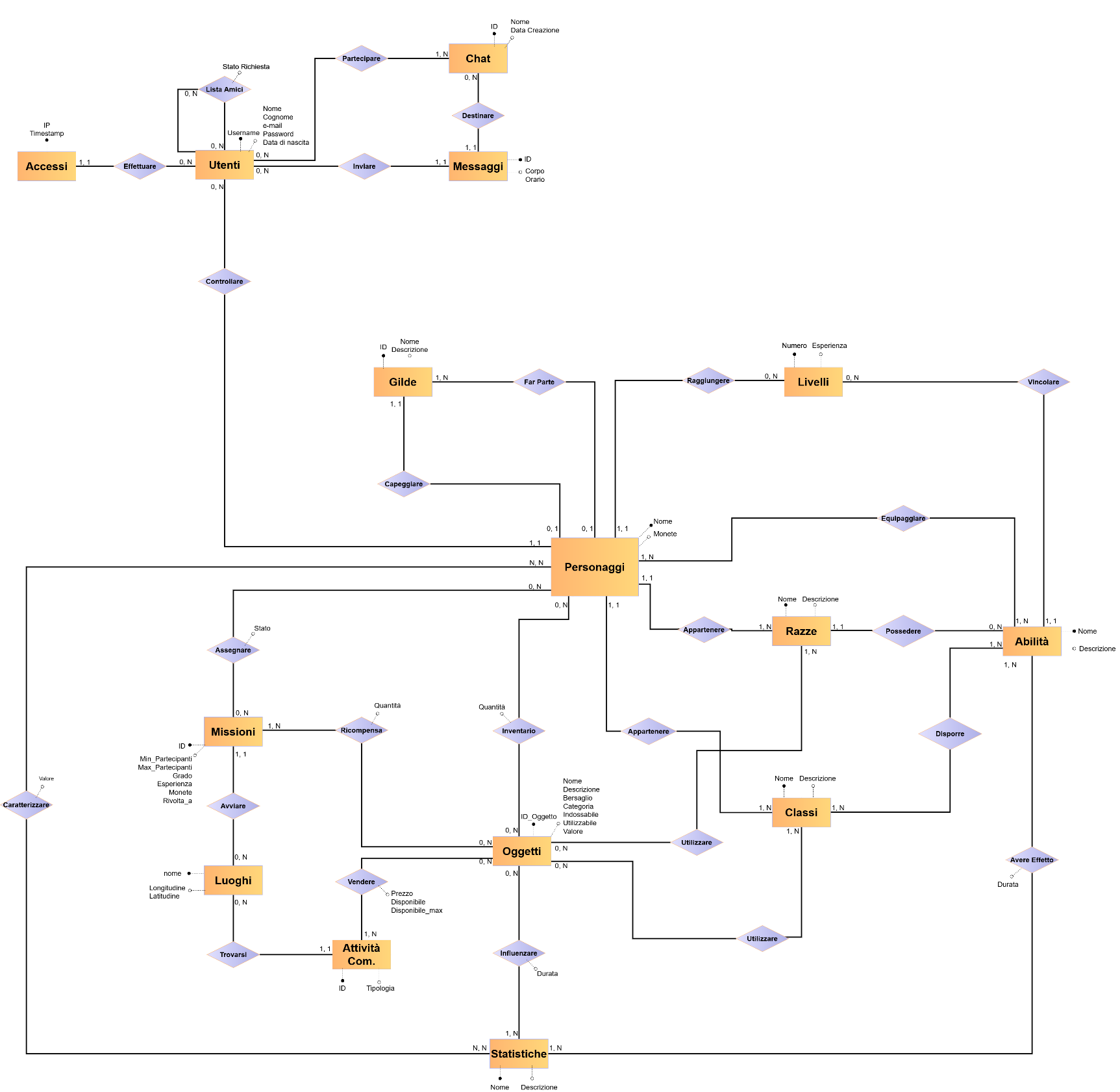
Si richiede di mantenere la consistenza tra le quantità di oggetti disponibili in negozi e inventari durante le transazioni che avvengono tra personaggi e attività commerciali. Anche a tale scopo scegliamo di usare ***InnoDB*** come engine per il DB (oltre che per le sue ottime prestazioni su database con molti inserimenti).

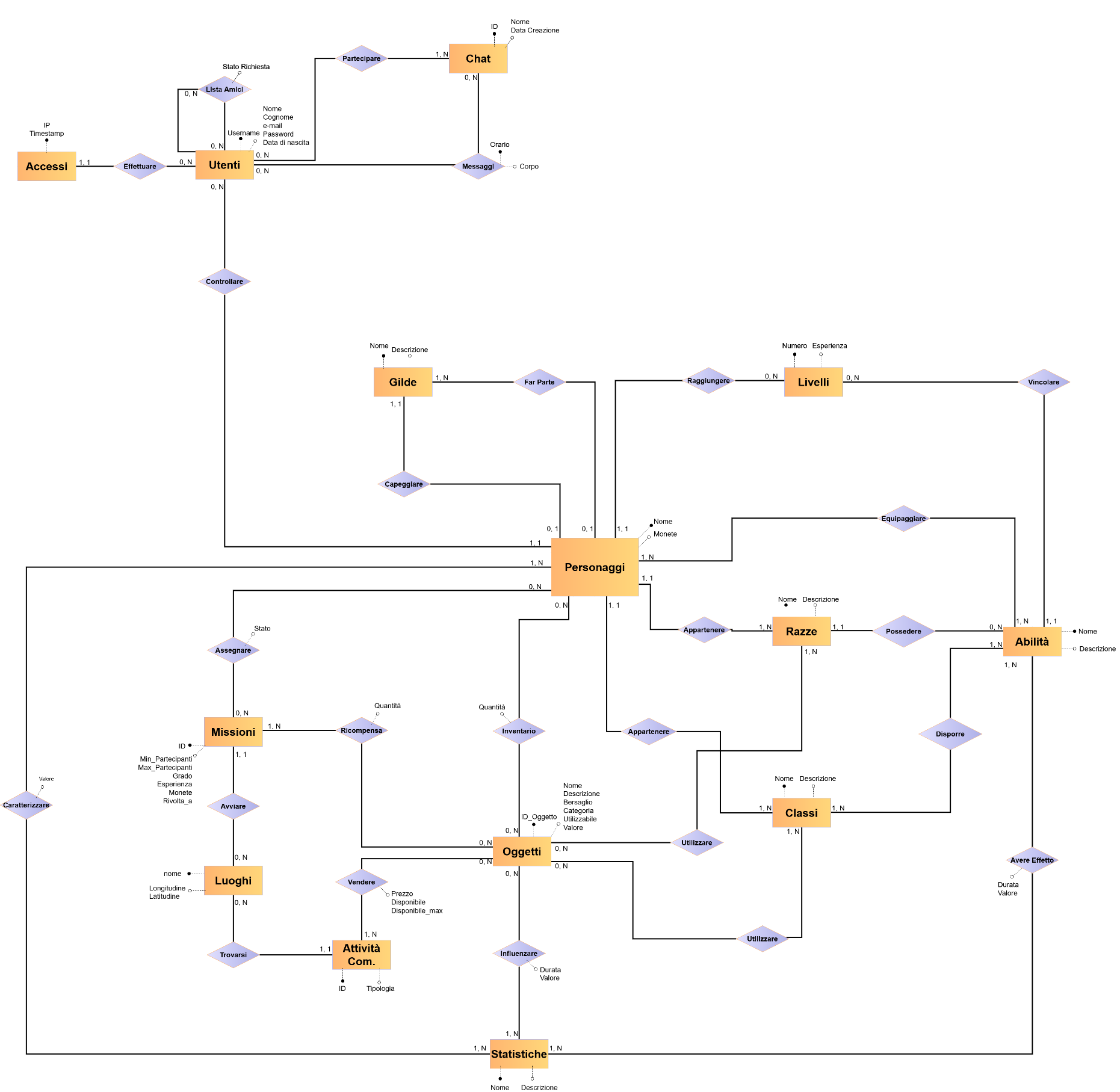
Ogni 24 ore la disponibilità di un oggetto in un’attività deve essere riportata al massimo. Ovviamente la disponibilità non può superare la disponibilità massima.

Non è di interesse memorizzare lo storico degli acquisti;

* Non interessa costruire entità che memorizzano categorie e bersagli di oggetti essendo statici e relativi solamente al singolo oggetto. Basta inserire un vincolo nella definizione della tabella.

C’è però la necessità di mettere in relazione gli oggetti con le razze e le classi per memorizzare tutte gli oggetti utilizzabili da una specifica classe/razza (possedere un oggetto non implica che esso possa essere utilizzato da quel personaggio);

**SCHEMA E-R** **LOGICO**

**[](https://uniroma2-my.sharepoint.com/personal/manuel_dilullo_students_uniroma2_eu/Documents/UNI/2_Anno/Corsi/Basi%20dati%20conoscenza/Progetto/Consegna/ERLogico.png)**

**Elenco** **entità:**

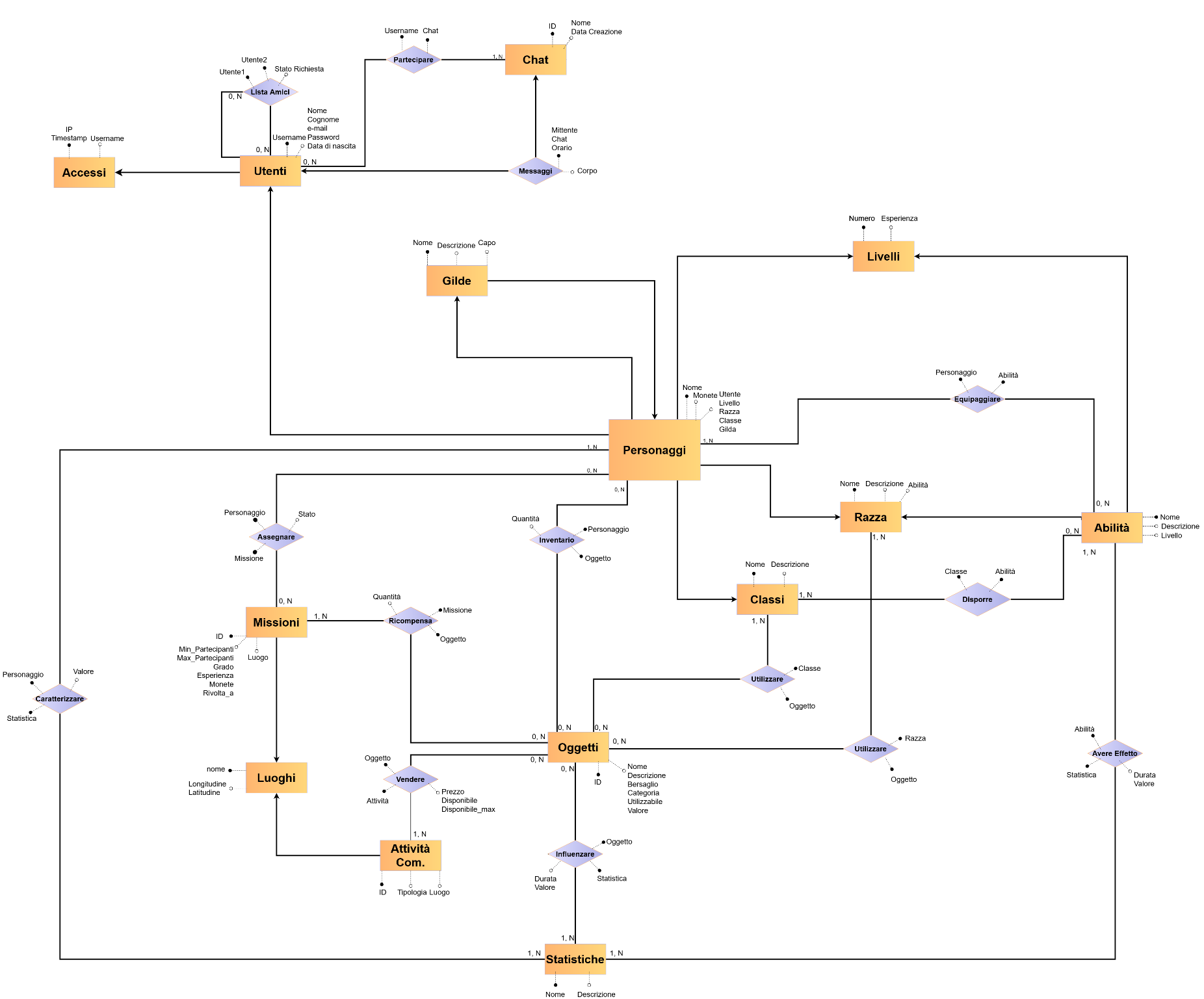
|  |  |  |
| --- | --- | --- |
| ENTITA’ | ATTRIBUTO CHIAVE | ATTRIBUTI |
| Utenti | Username | * Nome * Cognome * e-mail * password * data di nascita |
| Accessi | IP, timestamp |  |
| Chat | ID | * nome * data creazione |
| Personaggi | Nome | * monete |
| Gilde | Nome | * descrizione |
| Livelli | Livello | * esperienza |
| Abilità | Nome | * descrizione |
| Razze | Nome | * descrizione |
| Classi | Nome | * descrizione |
| Oggetti | Nome | * descrizione * bersaglio * categoria * valore |
| Statistiche | Nome |  |
| Missioni | ID | * min\_partecipanti * max\_partecipanti * grado * esperienza * monete * rivolata\_a |
| Luoghi | Nome | * latitudine * longitudine |
| Attività commerciali | ID | * tipologia |

**Elenco** **relazioni:**

|  |  |  |  |
| --- | --- | --- | --- |
| RELAZIONE | DESCRIZIONE | RELAZIONE | ATTRIBUTI |
| Effettuare | Un utente può effettuare N accessi ma un accesso può essere effettuato da un solo utente | **Accessi**  **1,1 < -- > 0,N**  **Utenti** |  |
| Lista Amici | Un utente può essere amico di uno o più utenti o potrebbe non avere amici | **Utenti**  **0,N < -- > 0,N**  **Utenti** | **stato** |
| Messaggi | Un utente può inviare un 0, 1 o più messsaggi verso una chat. Una chat può essere destinataria di molti messaggi inviati da diversi utenti | **Utenti**  **0,N < -- > 0,N**  **Chat** |  |
| Partecipare | Un utente può far parte di una o più chat o non far parte di nessuna, mentre una chat deve avere almeno un partecipante per esistere | **Utenti**  **0,N < -- > 1,N Chat** |  |
| Controllare | Un utente può non avere ancora creato un personaggio oppure potrebbe averne 1 o più, ma un personaggio appartiene ad un solo utente | **Utenti**  **0,N < -- > 1,1**  **Personaggi** |  |
| Capeggiare | Una gilda può avere un solo capo mentre un personaggio può essere a capo di una sola gilda | **Personaggi**  **0,1 < -- > 1,1**  **Gilde** |  |
| Far Parte | Un personaggio può non far parte o far parte di una sola gilda ma una gilda può avere più membri | **Personaggi**  **0,1 < -- > 1,N**  **Gilde** |  |
| Raggiungere | Un personaggio può raggiungere più livelli, ma viene memorizzato solo il più recente. Pertanto, un personaggio deve avere solo un livello, mentre uno stesso livello può essere stato raggiunto da più personaggi | **Personaggi**  **1,1 < -- > 0,N**  **Livelli** |  |
| Vincolare | Un’abilità deve essere disponibile da un determinato livello in poi, mentre uno stesso livello può vincolare più abilità | **Livelli**  **0,N < -- > 1,1**  **Abilità** |  |
| Equipaggiare | Un utente può equipaggiare da 1 a 20 abilità, mentre un’abilità potrebbe essere equipaggiata anche da tutti gli utenti. Poniamo 20 = N | **Personaggi**  **1,N < -- > 1,N**  **Abilità** |  |
| Appartenere\_  razze | Un personaggio deve appartenere ad una sola razza, ma ci sono più personaggi della stessa razza | **Personaggi**  **1,1 < -- > 1,N**  **Razze** |  |
| Possedere | Ogni razza possiede un’abilità innata, ma un’abilità può essere posseduta da più razze | **Razze**  **1,1 < -- > 0,N**  **Abilità** |  |
| Utilizzare\_ razze | Una razza può usare da 1 a N oggetti, mentre gli oggetti possono essere utilizzati anche da più razze. Esistono però gli oggetti non utilizzabili. | **Oggetti**  **0,N < -- > 1,N**  **Razze** |  |
| Appartenere\_  classi | Un personaggio deve appartenere ad una sola classe, ma ci sono più personaggi della stessa classe | **Personaggi**  **1,1 < -- > 1,N**  **Classi** |  |
| Disporre | Una classe può disporre di un insieme di abilità di grandezza arbitraria e ogni abilità potrebbe essere utilizzabile da 1 o più classi | **Classi**  **1,N < -- > 1,N**  **Abilità** |  |
| Utilizzare\_  Classi | Una classe può usare da 1 a N oggetti, mentre gli oggetti possono essere utilizzati anche da più classi. Esistono però gli oggetti non utilizzabili. | **Oggetti**  **0,N < -- > 1,N**  **Razze** |  |
| Inventario | Un personaggio può avere un numero indeterminato di oggetti, da 0 ad N ed un oggetto potrebbe essere nell’inventario di 0 o N personaggi. Se un’oggetto è presente nell’inventario di un personaggio è seguito da un valore numerico che indica la quantità | **Oggetti**  **0,N < -- >0,N**  **Personaggi** |  |
| Assegnare | Una missione potrebbe essere stata assegnata o meno ad un numero arbitrario di personaggi, mentre ogni personaggio potrebbe avere o meno preso in carico da 0 ad N missioni. Per ogni incarico viene anche specificato se è stato portato a termine o è ancora in corso | **Personaggi**  **0,N < -- > 0,N**  **Missioni** | **quantità** |
| Caratterizzare | Tutte le statistiche caratterizzano tutti i personaggi | **Statistiche**  **N,N < -- > N,N**  **Personaggi** | **stato** |
| Ricompensare | Una missione offre sicuramente 1 o più oggetti come ricompensa, ma un oggetto potrebbe essere ricompensa di 0, 1 o più missioni. Ogni ricompensa è seguita dalla sua quantità | **Missioni**  **1,N < -- > 0,N**  **Oggetti** |  |
| Avviare | Ogni missione ha un posto predefinito dove può essere avviata ma un posto potrebbe non essere un luogo dove poter avviare una missione | **Missioni**  **1,1 < -- > 0,N Luoghi** |  |
| Trovarsi | Un’attività commerciale si trova sicuramente in uno dei luoghi memorizzati nell’omonima entità, ma un luogo potrebbe non avere relazioni con un’attività | **Attività Commerciali**  **1,1 < -- > 0,N Luoghi** |  |

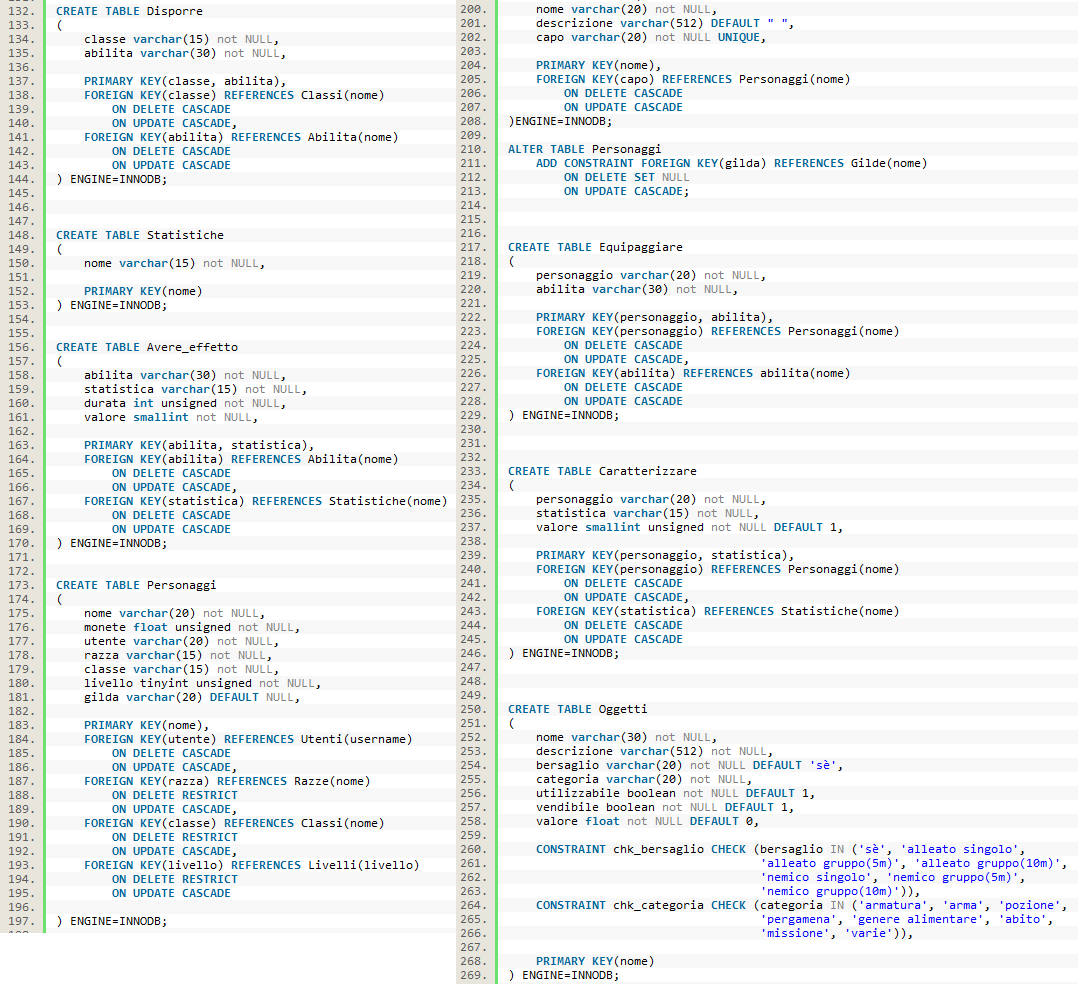
|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
| Vendere | Un’attività vende da 1 a più oggetti, ma un’oggetto potrebbe non essere acquistabile presso esse. Un’oggetto venduto presso un’attività commerciale presenta un prezzo, una disponibilità attuale e una disponibilità massima | **Attività Commerciali**  **1,N < -- > 0,N**  **Oggetti** |  |
| Influenzare | Un oggetto può influenzare più statistiche, mentre sicuramente esiste un’oggetto che influenza una determinata statistica. Esistono però anche oggetti non utilizzabili | **Statistiche**  **1,N < -- > 0,N Oggetti** | **prezzo, disponibile, disponibile\_max** |
| Avere Effetto | Un’abilità può avere effetto su più statistiche, mentre sicuramente esiste un’abilità che ha effetto su una determinata statistica. | **Statistiche**  **1,N < -- > 1,N**  **Abilità** |  |

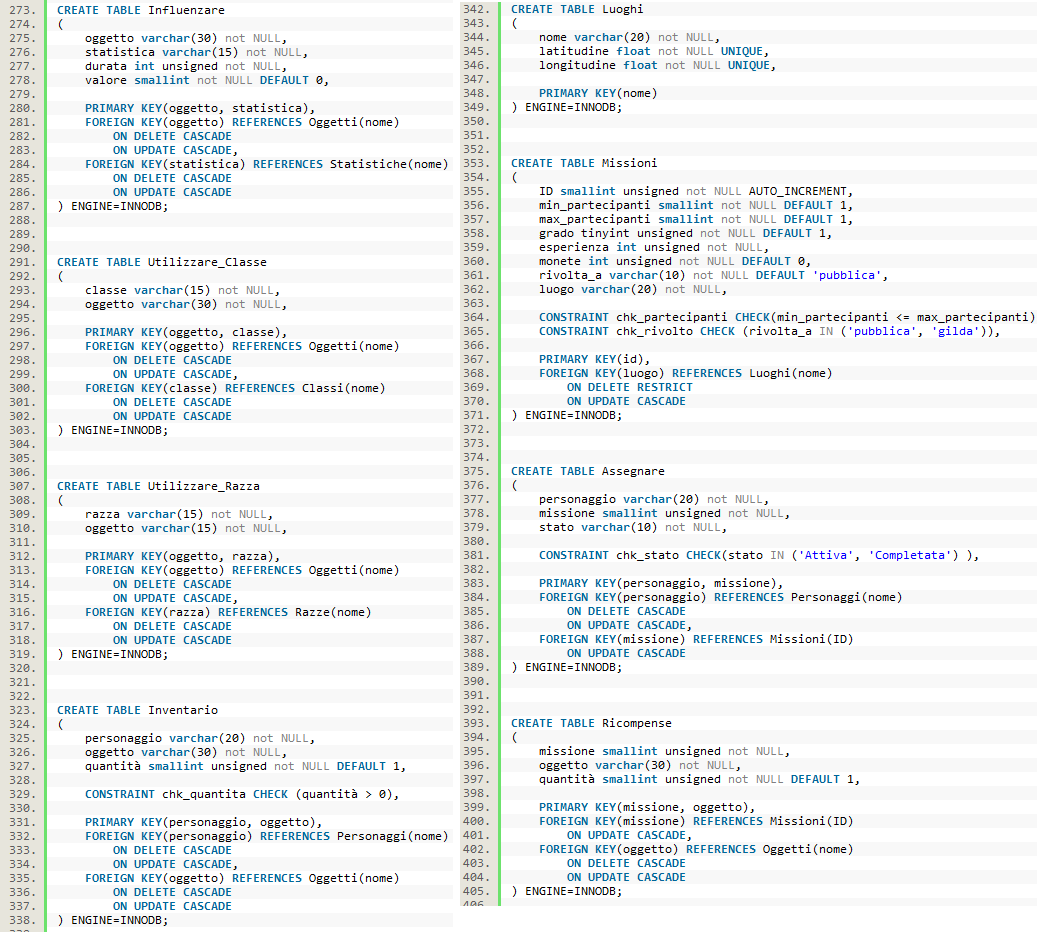
**SCHEMA E-R** **FISICO**

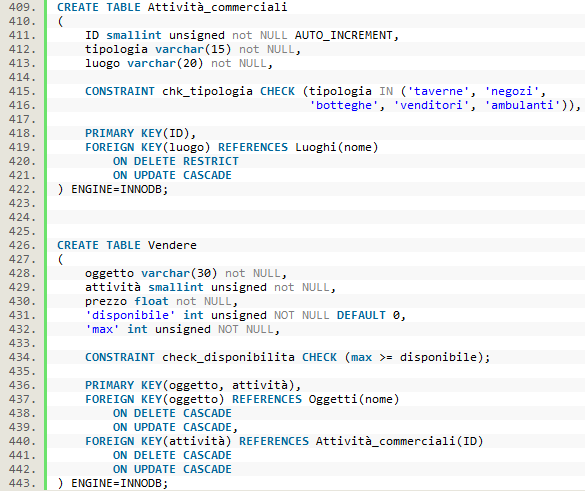
**[](https://uniroma2-my.sharepoint.com/personal/manuel_dilullo_students_uniroma2_eu/Documents/UNI/2_Anno/Corsi/Basi%20dati%20conoscenza/Progetto/Consegna/ERFisico.png)**

**CREAZIONE TABELLE**









FILE IN ALLEGATO.

(4\_CreateDB.sql

oppure

4\_CreateDB.pdf)

**INSERIMENTI TABELLE**

Per quanto riguarda gli inserimenti è stata sviluppata una piccola applicazione con interfaccia grafica in python per automatizzare le operazioni di inserimento.

Lo script utilizza il modulo mysql.connector per interfacciarsi direttamente col server mysql.

Una volta stabilita la connessione esegue una query per ottenere il nome di tutte le tabelle presenti nel database e li mostra a schermo sottoforma di etichette per checkbuttons.

Da quel momento in poi si può scegliere una o più tabelle e dare il via all’inserimento, oppure premere sul tasto “elimina tutto” per svuotare tutte le tabelle del DB, oppure premere direttamente sul tasto “inserisci tutto” per far partire l’inserimento su tutte le tabelle.

Le funzioni di inserimento generano una sequenza di dati casuali e permettono di inserire migliaia di dati in pochi istanti

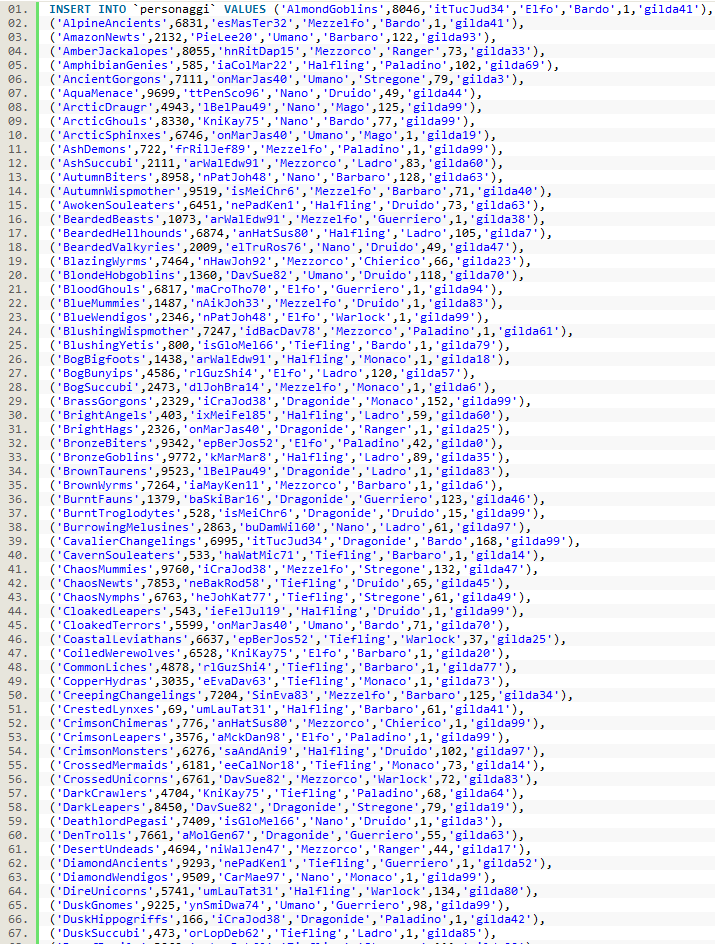
Qui di seguito la schermata iniziale e in allegato lo script.



Tramite il comando mysqldump si è poi potuto estrarre il contenuto delle tabelle ricavando tutti i comandi “Insert Into” applicati al database.

Esempi di inserimento:





FILE IN ALLEGATO.

(5\_Insert.sql

oppure

5\_Insert.pdf)

**QUERY\***

\*(Le query con il limit servono solo per facilitare la cattura delle schermate)

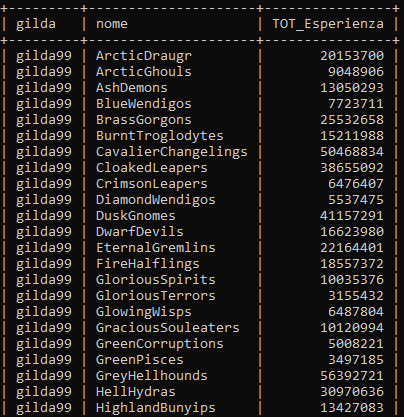
1. **STAMPA ESPERIENZA GUADAGNATA DA TUTTI I MEMBRI DELLA GILDA "gilda99"**

SELECT personaggi.gilda, personaggi.nome, SUM(esperienza) as TOT\_Esperienza

FROM (assegnare JOIN missioni ON assegnare.missione = missioni.ID) JOIN personaggi ON personaggi.nome = assegnare.personaggio

WHERE assegnare.stato = "Completata" AND personaggi.gilda = "gilda99"

GROUP BY personaggi.nome;



1. **STAMPA A SCHERMO L'ESPERIENZA GUADAGNATA DA I GIOCATORI MEMBRI DI UNA GILDA ATTRAVERSO MISSIONI DEDICATE A QUESTE ULTIME E ORDINARE IN BASE ALL'ESPERIENZA TOTALE GUADAGNATA DALLA GILDA**

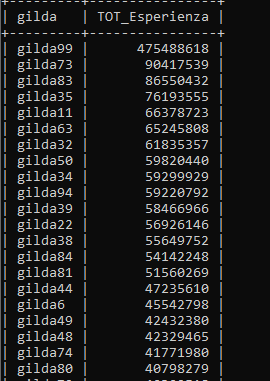
SELECT personaggi.gilda, SUM(esperienza) as TOT\_Esperienza

FROM (assegnare JOIN missioni ON assegnare.missione = missioni.ID) JOIN personaggi ON personaggi.nome = assegnare.personaggio

WHERE assegnare.stato = "Completata" AND missioni.rivolta\_a = "gilda" AND NOT ISNULL(personaggi.gilda)

GROUP BY personaggi.gilda

ORDER BY Tot\_Esperienza DESC;



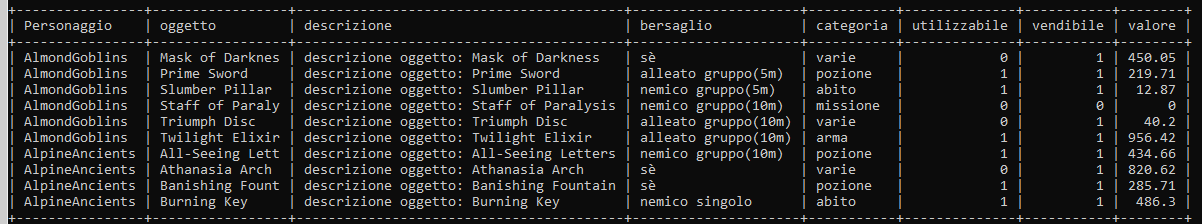
1. **STAMPA INFORMAZIONI DI TUTTI GLI OGGETTI POSSEDUTI DA UN UTENTE RAGGRUPPATI PER PERSONAGGIO**

SELECT personaggi.nome AS Personaggio, oggetti.\*

FROM (inventario JOIN oggetti ON oggetti.nome = inventario.oggetto) JOIN personaggi ON personaggi.nome = inventario.personaggio

ORDER BY personaggi.nome

LIMIT 10;



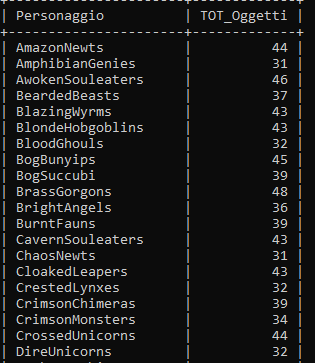
1. **STAMPA IL NUMERO DI OGGETTI POSSEDUTI DA OGNI PERSONAGGIO CHE NE POSSIEDE PIU DI 30**

SELECT personaggi.nome AS Personaggio, COUNT(oggetti.nome) as TOT\_Oggetti

FROM (inventario JOIN oggetti ON oggetti.nome = inventario.oggetto) JOIN personaggi ON personaggi.nome = inventario.personaggio

GROUP BY personaggi.nome

HAVING TOT\_Oggetti > 30;

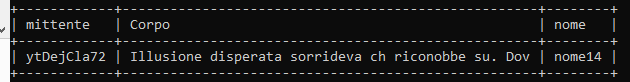


1. **STAMPA CORPO (primi 100 caratteri) DEI MESSAGGI INVIATI DA L'UTENTE "ytDejCla72" E LE CHAT VERSO LE QUALI LI HA INVIATI**

SELECT messaggi.mittente, SUBSTRING(messaggi.Corpo, 1, 50) AS Corpo, chat.nome

FROM chat JOIN messaggi ON chat.ID = messaggi.chat

WHERE messaggi.mittente = "ytDejCla72”;



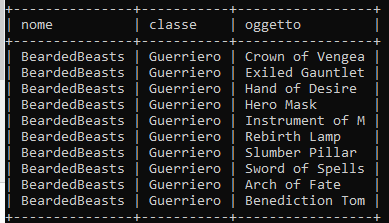
1. **STAMPA NOME DI TUTTI GLI OGGETTI GUADAGNATI COMPLETANDO MISSIONI DA PERSONAGGI DI CLASSE "Guerriero"**

SELECT personaggi.nome, personaggi.classe, ricompense.oggetto

FROM ( (assegnare JOIN personaggi ON personaggi.nome = assegnare.personaggio) JOIN missioni ON  assegnare.missione = missioni.ID) JOIN ricompense ON ricompense.missione = missioni.ID

WHERE assegnare.stato = "Completata" AND personaggi.classe = "Guerriero"

ORDER BY personaggi.nome LIMIT 10;



1. **STAMPA INFORMAZIONI PERSONAGGIO CHE POSSIEDE IL MAGGIOR NUMERO DI OGGETTI (numero oggetti diversi, non interessano le quantità)**

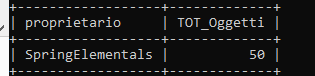
SELECT personaggi.nome as proprietario, COUNT(oggetti.nome) as TOT\_Oggetti

FROM (inventario JOIN oggetti ON oggetti.nome = inventario.oggetto) JOIN personaggi ON personaggi.nome = inventario.personaggio

GROUP BY personaggi.nome

ORDER BY TOT\_Oggetti DESC

LIMIT 1;



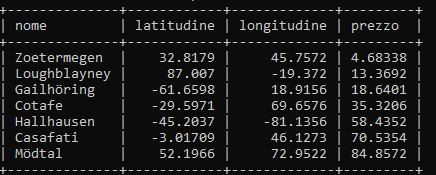
1. **STAMPA NOME, COORDINATE E PREZZO DEI LUOGHI CHE VENDONO "Wisdom Jar" ORDINATI IN ORDINE DI PREZZO CRESCENTE**

SELECT luoghi.nome, luoghi.latitudine, luoghi.longitudine, vendere.prezzo

FROM (luoghi JOIN attività\_commerciali ON luoghi.nome = attività\_commerciali.luogo) JOIN vendere ON attività\_commerciali.ID = vendere.attività

WHERE vendere.oggetto = "Wisdom Jar"

ORDER BY vendere.prezzo ASC;



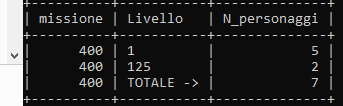
1. **STAMPA IL NUMERO DI PERSONAGGI DELLO STESSO LIVELLO CHE HANNO COMPLETATO LA MISSIONE "400" E POI STAMPARNE IL TOTALE**

SELECT assegnare.missione, COALESCE(personaggi.livello, "TOTALE ->") AS Livello, COUNT(personaggi.nome) AS N\_personaggi

FROM assegnare JOIN personaggi ON personaggi.nome = assegnare.personaggio

WHERE assegnare.missione = 400 AND assegnare.stato = "Completata"

GROUP BY personaggi.livello WITH ROLLUP;



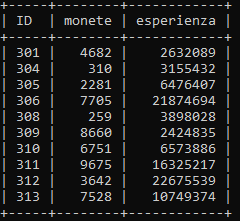
1. **STAMPA ID, MONETE, ESPERIENZA DELLE MISSIONI CHE FORNISCONO OGGETTI CHE HANNO EFFETTO SULLA SALUTE**

SELECT missioni.ID, missioni.monete, missioni.esperienza

FROM ( (ricompense JOIN missioni ON missioni.ID = ricompense.missione) JOIN oggetti ON ricompense.oggetto = oggetti.nome) JOIN influenzare ON influenzare.oggetto = oggetti.nome

WHERE influenzare.statistica IN ("salute\_attuale", "salute\_massima")

GROUP BY missioni.ID LIMIT 10;

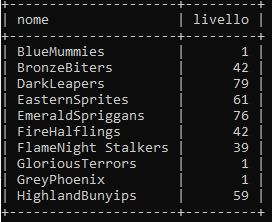


1. **STAMPA NOME E LIVELLO DI TUTTI I PERSONAGGI CHE HANNO COMPLETATO UNA MISSIONE DI GRADO >= 125**

SELECT personaggi.nome, personaggi.livello

FROM (assegnare JOIN personaggi ON personaggi.nome = assegnare.personaggio) JOIN missioni ON missioni.ID = assegnare.missione

WHERE missioni.grado >= 125 LIMIT 10;

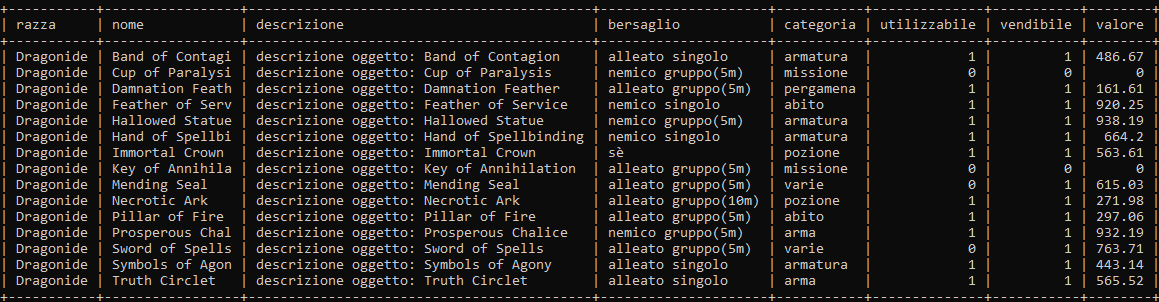


1. **STAMPA OGGE.TTI UTILIZZABILI DALLA RAZZA "Dragonide"**

SELECT utilizzare\_razza.razza, oggetti.\*

FROM utilizzare\_razza JOIN oggetti ON oggetti.nome = utilizzare\_razza.oggetto

WHERE utilizzare\_razza.razza = "Dragonide";



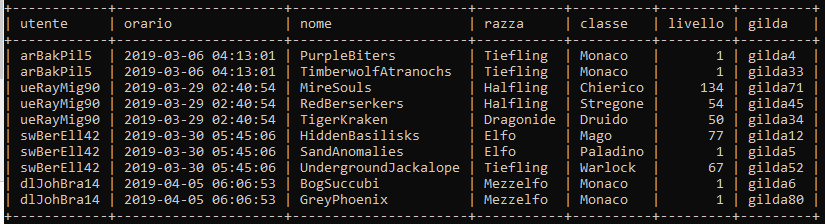
1. **STAMPA NOME, RAZZA, CLASSE, LIVELLO E GILDA DEI PERSONAGGI POSSEDUTI DA UTENTI CHE HANNO ESEGUITO L'ACCESSO TRA IL "2019-02-25 21:40:24" E IL "2019-10-31 17:50:21"**

SELECT personaggi.utente, orario, personaggi.nome, personaggi.razza, personaggi.classe, personaggi.livello, personaggi.gilda

FROM utenti JOIN accessi ON utenti.username = accessi.username JOIN personaggi ON personaggi.utente = utenti.username

WHERE orario BETWEEN "2019-02-25 21:40:24" AND "2019-10-31 17:50:21"

ORDER BY orario LIMIT 10;

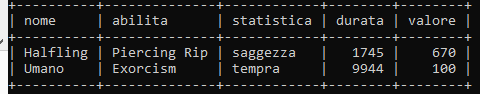


1. **STAMPA NOME RAZZA, NOME ABILITA, EFFETTI E DURATA DELLE ABILITA INNATE DELLE RAZZE CHE POSSONO USARE L'OGGETTO "Sanctifying Sta"**

SELECT razze.nome, avere\_effetto.\*

FROM ((avere\_effetto JOIN abilita ON abilita.nome = avere\_effetto.abilita) JOIN razze ON razze.abilita = abilita.nome) JOIN utilizzare\_razza ON utilizzare\_razza.razza = razze.nome

WHERE utilizzare\_razza.oggetto = "Sanctifying Sta";

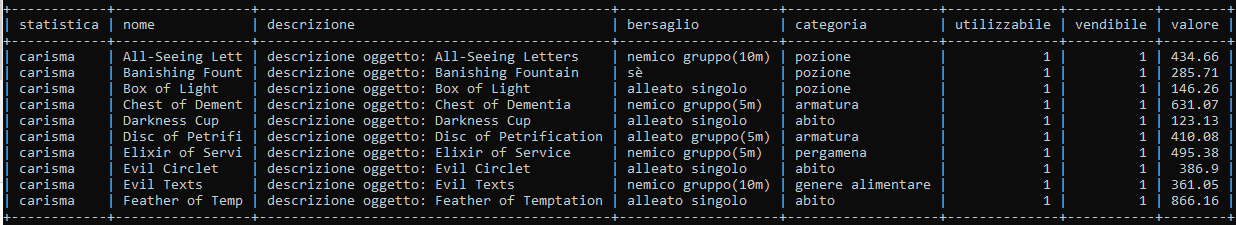


1. **STAMPA INFORMAZIONI SUGLI OGGETTI UTILIZZABILI SULLA STATISTICA COL VALORE MASSIMO TRA TUTTE LE STATISTICHE DI TUTTI I PERSONAGGI**

SELECT influenzare.statistica, oggetti.\*

FROM (SELECT MAX(caratterizzare.valore) as massimo, statistica from caratterizzare) max\_stat, oggetti JOIN influenzare ON influenzare.oggetto = oggetti.nome

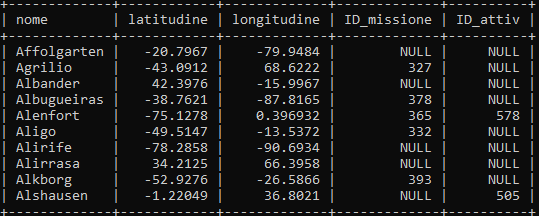
WHERE influenzare.statistica = max\_stat.statistica LIMIT 10;



1. **STAMPA INFORMAZIONI DI TUTTI I LUOGHI E, SE ESISTONO, MISSIONI E ATTIVITA SITUATE IN QUEL LUOGO**

SELECT luoghi.\*, missioni.ID AS ID\_missione, attività\_commerciali.ID AS ID\_attiv

FROM (luoghi LEFT JOIN missioni ON luoghi.nome = missioni.luogo) LEFT JOIN attività\_commerciali ON attività\_commerciali.luogo = luoghi.nome LIMIT 10;



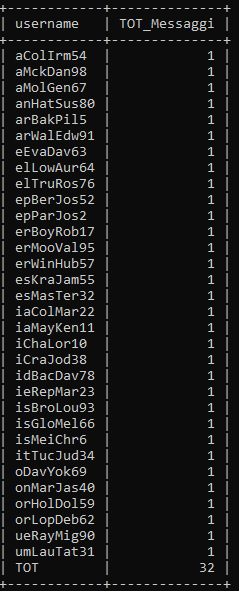
1. **STAMPA IL NUMERO DI MESSAGGI INVIATI DAGLI UTENTI IL CUI NOME INIZIA PER UNA VOCALE**

SELECT COALESCE(username, “TOT”) AS username, COUNT(mittente) as TOT\_Messaggi

FROM  messaggi JOIN utenti ON messaggi.mittente = utenti.username

WHERE utenti.username LIKE "a%" OR utenti.username LIKE "e%" OR utenti.username LIKE "i%" OR utenti.username LIKE "o%" OR utenti.username LIKE "u%"

GROUP BY utenti.username WITH ROLLUP;



1. **STAMPA LA MEDIA E LA SOMMA DELLE MISSIONI COMPLETATE DAI MEMBRI DI UNA GILDA E IL NUMERO DI MEBRI DI ESSA**

SELECT completate.gilde, COUNT(completate.pers),

SUM(completate.Totale) as Totale\_Gilde,

AVG(completate.Totale) AS Media\_Gilde

FROM (

           SELECT personaggi.nome as pers, personaggi.gilda as gilde,

COUNT(assegnare.missione) as Totale

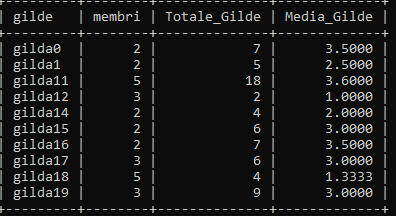
           FROM personaggi JOIN assegnare ON personaggi.nome = assegnare.personaggio

           WHERE assegnare.stato = "Completata"

           GROUP BY personaggi.nome

      ) completate

GROUP BY completate.gilde;



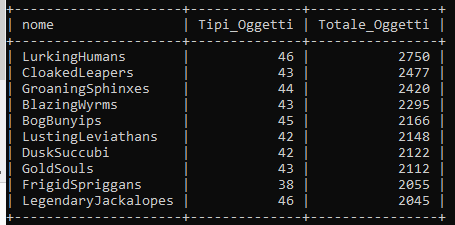
1. **STAMPA IL NUMERO DI OGGETTI POSSEDUTI DAI CAPI DELLE GILDE (dove il numero oggetti è inteso somma quantità per ogni oggetto) E ORDINALI IN MANIERA DESCRESCENTE**

SELECT personaggi.nome, COUNT(oggetto) as Tipi, SUM(quantità) as Totale

FROM (gilde JOIN personaggi ON personaggi.nome = gilde.capo) JOIN inventario ON inventario.personaggio = personaggi.nome

GROUP BY personaggi.nome

ORDER BY Totale DESC LIMIT 10;



1. **STAMPA, PER OGNI PERSONAGGIO, LA QUANTITA DI ESPERIENZA GUADAGNATA COMPLETANDO LE MISSIONI, QUELLA GUADAGNABILE TRAMITE MISSIONI ATTIVE E LA DIFFERENZA TRA LE DUE ORDINATI PER QUEST’ ULTIMO VALORE**

SELECT nome, Completate, Attive, (Completate - Attive) as Differenza

FROM (SELECT personaggio as nome,

SUM(case when stato = "Completata" then missioni.esperienza else 0

end) AS Completate,

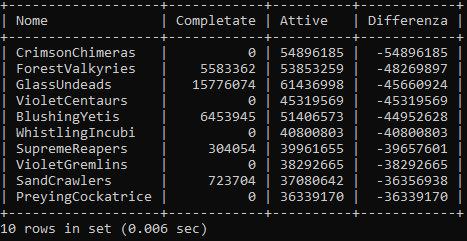
SUM(case when stato = "Attiva" then missioni.esperienza else 0 end)

AS Attive

FROM missioni JOIN assegnare ON missioni.ID = assegnare.missione

GROUP BY personaggio) AS Conteggio

ORDER BY Differenza LIMIT 10;



1. **STAMPA LE MISSIONI COMPLETATE E ATTIVE DEI MEMBRI DELLA GILDA DEL PERSONAGGIO "FrigidYetis" CHE SIANO STATE COMPLETATE O ATTIVATE DA QUEST'ULTIMO**

SELECT PM.gilda, PM.nome, AF.missione

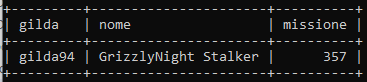
FROM assegnare AF JOIN personaggi PF ON AF.personaggio = PF.nome,

     assegnare AM JOIN personaggi PM ON AM.personaggio = PM.nome

WHERE AF.missione = AM.missione AND

      PF.gilda = PM.gilda AND

      PF.nome = "FrigidYetis" AND PM.nome <> "FrigidYetis";



**QUERY** **ALGEBRA/CALCOLO RELAZIONALE**

**STAMPA INFORMAZIONI DI TUTTI I LUOGHI E, SE ESISTONO, MISSIONI E ATTIVITA SITUATE IN QUEL LUOGO**

**SQL:**

SELECT luoghi.\*, missioni.ID AS ID\_missione, attività\_commerciali.ID AS ID\_attività

FROM (luoghi JOIN missioni ON luoghi.nome = missioni.luogo) JOIN attività\_commerciali ON attività\_commerciali.luogo = luoghi.nome;

**ALGEBRA RELAZIONALE:**

∏ luoghi.nome,luoghi.longitudine,luoghi.latitudine,ID\_missione,ID\_attività (ρ ID\_missione, ID\_attività <- missioni.ID, attività\_commerciali.ID ( (Luoghi |X|luoghi.nome = missioni.luogo Missioni) |X|attività\_commerciali.luogo = luoghi.nome attività\_commerciali ) )

**STAMPA NOME RAZZA, NOME ABILITA, EFFETTI E DURATA DELLE ABILITA INNATE DELLE RAZZE CHE POSSONO USARE L'OGGETTO "Sanctifying Sta"**

**SQL:**

SELECT razze.nome, avere\_effetto.\*

FROM ((avere\_effetto JOIN abilita ON abilita.nome = avere\_effetto.abilita) JOIN razze ON razze.abilita = abilita.nome) JOIN utilizzare\_razza ON utilizzare\_razza.razza = razze.nome

WHERE utilizzare\_razza.oggetto = "Sanctifying Sta";

**CALCOLO RELAZIONALE (calcolo su tuple con dichiarazione di range):**

**{**

**razze.nome, avere\_effetto.\* | avere\_effetto, abilita, razze, utilizzare\_razza |**

**avere\_effetto.abilita = abilita.nome ^**

**abilita.nome = razze.abilita ^**

**utilizzare\_razza.razza = razze.nome ^**

**utilizzare\_razza.oggetto = “Sanctifying Sta”**

**}**

**PARTI** **ULTERIORI**

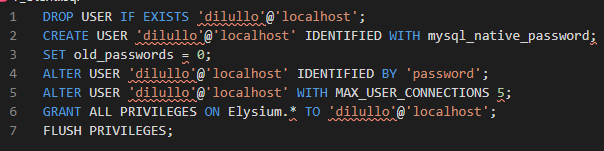
Durante la realizzazione della base di dati si è ritenuta necessaria la realizzazione di ulteriori funzionalità come:

* Creazione di nuovi utenti
* Stored Procedures
* Triggers
* Events
* Transactions

Tutti i codici sono disponibili come allegati al file di consegna.

**CREAZIONE** **UTENTI**

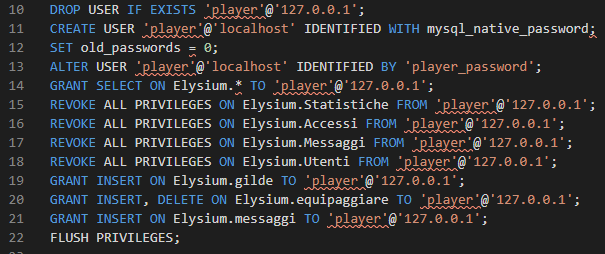
È stato necessario creare un nuovo utente, diverso dall’utente root, con i privilegi necessari ad operare nel DB Elysium



Con i comandi qui sopra elencati stiamo dichiarando un utente protetto da password con i privilegi massimi sul DB Elysium

Alla riga 2 stiamo definendo nome utente, IP e con quale plugin vogliamo memorizzare la password dell’utente. Per utilizzare il plugin mysql\_native\_password dobbiamo però disabilitare old\_passwords, settandolo a 0.

Fatto ciò, si può procedere scegliendo la nostra password, impostando il numero massimo di connessioni simultanee con quell’account e garantendo i privilegi massimi al nostro nuovo utente. Infine, eseguiamo un’operazione di flush per terminare e confermare le operazioni precedenti.



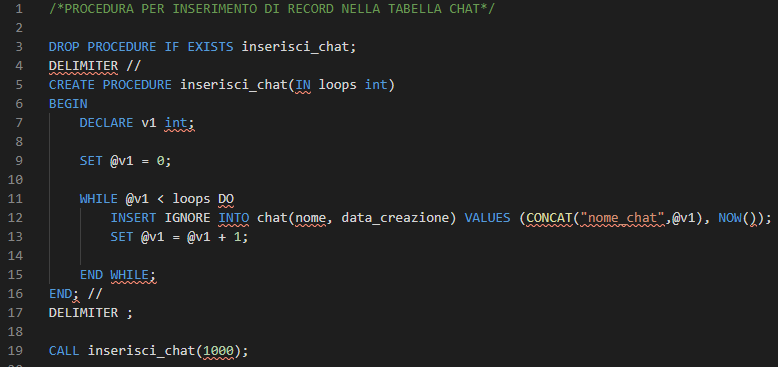
Creiamo ora un altro utente che ci servirà a gestire i privilegi dati ai giocatori. Questo utente permette di vedere i dati di tutte le tabelle escluse statistiche, utenti, accessi e messaggi. La tabella statistiche non necessita di essere visibile poiché è già visibile la tabella caratterizzare, le tabelle utenti ed accessi contengono informazioni private mentre la tabella messaggi non può essere completamente visibile a tutti i players.

Per il resto questo utente non può modificare le entità statiche, ma solo quelle direttamente utilizzabili in fase di gioco: può equipaggiare e “dis” -equipaggiare abilità, può creare gilde e può inviare messaggi.

**STORED** **PROCEDURES**

Sono state realizzate 2 stored procedure, il cui scopo è stato quello di velocizzare ed automatizzare gli inserimenti di record nel DB.

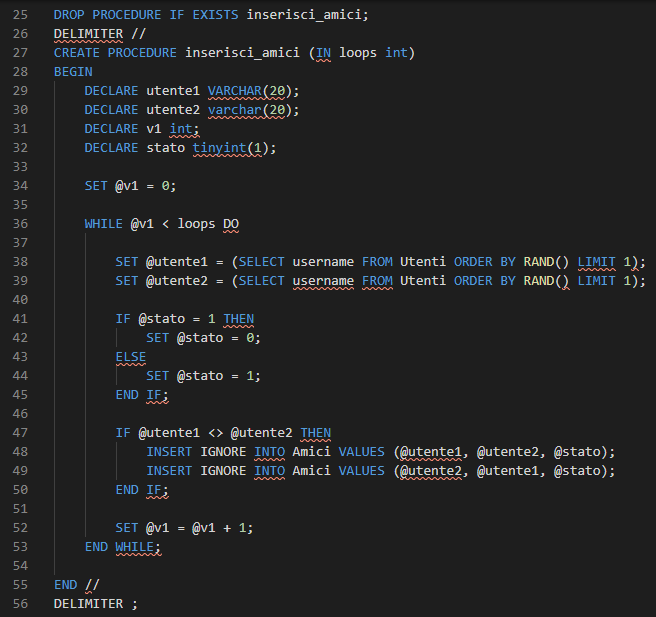
La prima prende in input un valore intero “loops” e lo usa come condizione per iterare una variabile v1. Ad ogni iterazione viene inserito nella tabella “chat” un nome, dato dalla concatenazione della stringa “nome\_chat” e l’attuale valore della variabile v1, e il timestamp attuale.



La seconda procedure invece agisce sulla tabella amici. Anche in questo caso viene preso in input un valore numerico “loops” utilizzato per l’iterazione della variabile v1.

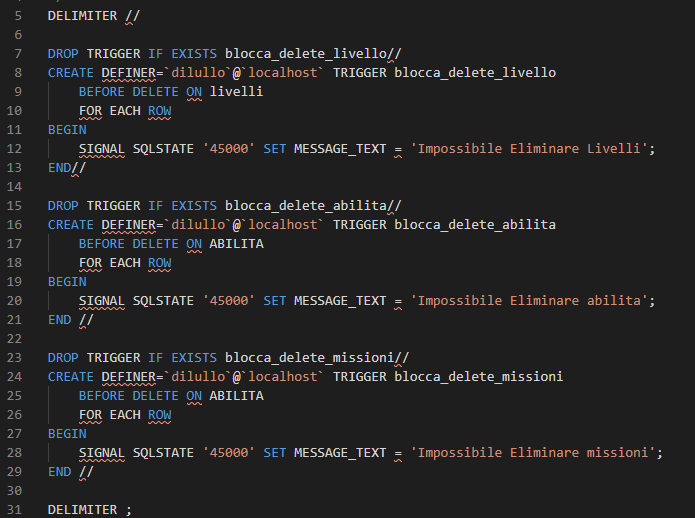
Ad ogni ciclo vengono estratti due utenti casuali dalla tabella utenti e vengono messi rispettivamente in due variabili “utente1” e “utente2”. Successivamente viene negata una variabile “stato” che serve ad identificare lo stato della richiesta di amicizia tra due utenti (la negazione serve a non inserire sempre lo stesso stato nella tabella).

Infine, si verifica che gli utenti estratti precedentemente siano diversi e si procede all’inserimento simmetrico dei due utenti nella tabella amici con lo stato dell’amicizia settato con la variabile “stato”.



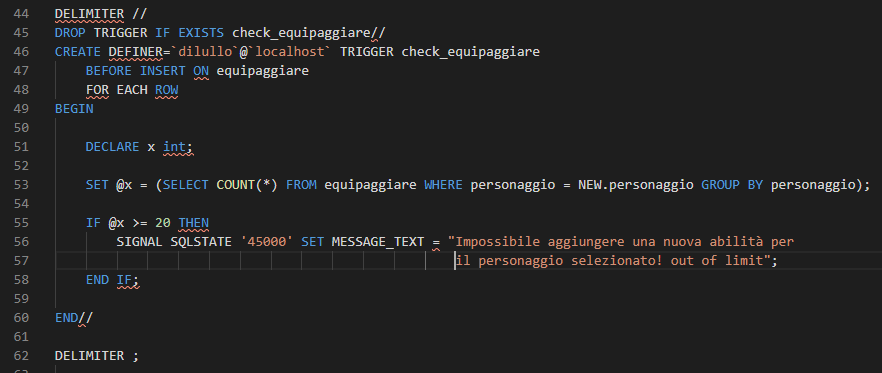
**TRIGGERS**

Essendo il videogioco un programma sempre in esecuzione e in continuo aggiornamento, la nostra base di dati deve essere pronta ad impedire operazioni che possano rendere dati inconsistenti o non disponibili. Per fare ciò si è deciso di costruire un insieme di triggers atti a difendere queste caratteristiche



Questo frammento di codice contiene 3 semplici triggers che si attivano nel momento in cui viene richiesta un’operazione di delete sulle tabelle “livelli”, “abilita” e “missioni”, impedendole lanciando un messaggio di errore

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_



Il trigger appena mostrato si attiva prima di un inserimento nella tabella equipaggiare, il suo scopo è quello di impedire vengano equipaggiate più di 20 abilità da un singolo personaggio.

Quando viene ricevuta una richiesta di inserimento nella tabella equipaggiare, questo trigger si attiva ed esegue una query atta a contare il numero di abilita già equipaggiate da un personaggio. Se questo numero è uguale a venti, lancia un’eccezione e stampa un messaggio di errore.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Un altro obiettivo che ci eravamo prefissati già al momento della descrizione del progetto, era quello di impedire che un personaggio già a capo di una gilda entri a far parte di un’altra.

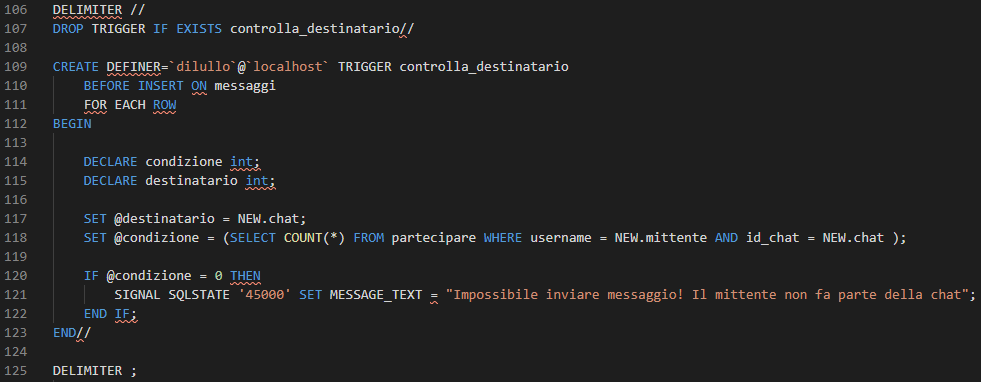
In questo caso entra in gioco il seguente trigger.



Il trigger appena definito entra in azione prima di un aggiornamento sulla tabella personaggi. Se si sta modificando la gilda di un personaggio, questa procedura lancia una query che ritorna il numero di gilde a cui è a capo l’avatar in questione. Se il risultato è maggiore di 0 vuol dire che è già a capo di una gilda diversa e non può cambiarla, pertanto, viene lanciato un messaggio di errore.

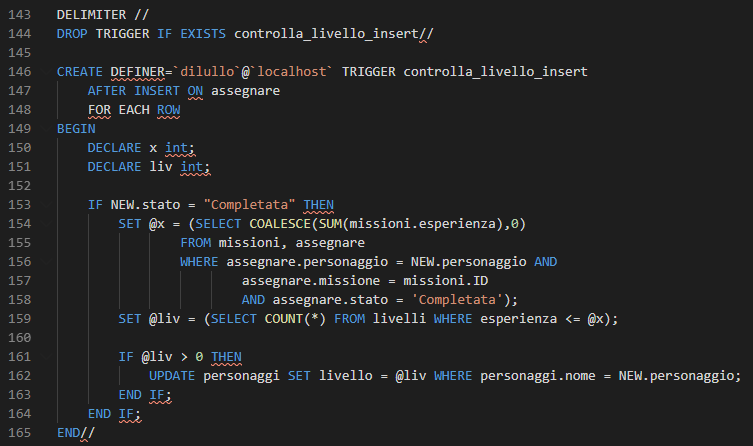
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Con il prossimo trigger si vuole risolvere il problema dei messaggi inviati a chat alle quali il mittente non appartiene



Con questo frammento di codice abbiamo creato un trigger che si attiva prima di un insert sulla tabella messaggi. Non appena attivo, il trigger controlla il destinatario di questo messaggio e lo cerca nella tabella partecipare, affiancato dall’username del mittente. Se la query risultante ritorna un valore > 0 vuol dire che l’utente fa parte di quella conversazione. Se non sarà così, verrà stampato un messaggio di errore.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_



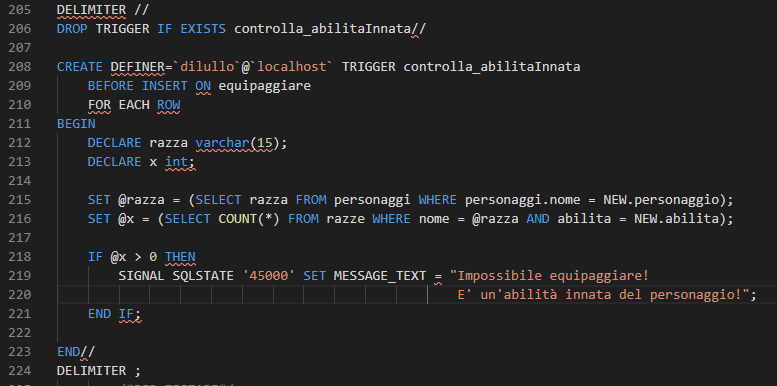
Questo trigger entra in gioco durante dopo un inserimento sulla tabella assegnare (viene omesso dalle schermate lo stesso trigger per gli update)

Questo trigger calcola rapidamente il livello raggiunto dall’utente quando completa una missione. Nel momento in cui viene inserita una nuova missione nella tabella assegnare si calcola la somma dell’esperienza guadagnata dalle missioni completate e si conta il numero di livelli con esperienza minore a quella precedentemente calcolata.

Se il numero di livelli superati è >0, questo viene aggiornato nella tabella personaggi, relativamente all’utente che ha completato la missione che ha fatto attivare il trigger.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

L’ultimo, ma non per importanza, trigger, verifica che non venga equipaggiata un’abilità che è già un’abilità innata per il personaggio in questione.

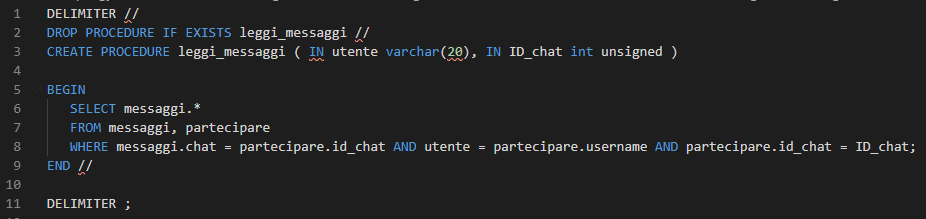


Si attiva prima di un inserimento nella tabella equipaggiare ed effettua una select per cercare le razza del personaggio che sta richiedendo di equipaggiare l’abilità.

Successivamente conta il numero di razze che hanno lo stesso nome della razza del personaggio e hanno come abilità quella che si sta cercando di equipaggiare. Se questo valore è maggiore di 0, allora l’abilità è innata per il personaggio in questione e viene impedito l’inserimento.

**PROCEDURES**

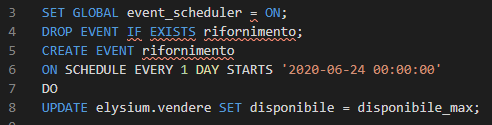
Tramite la creazione dell’utente “player” vista nel paragrafo “Creazione utenti”, è stata vietata l’operazione di select sulla tabella messaggi da parte dei giocatori. Ogni utente però deve poter leggere i messaggi destinati ad una chat della quale fa parte. Per questo è stata scritta una procedura apposita che mostra soltanto i messaggi della chat richiesta in input, se e solo se l’utente in input è un partecipate di essa.



**EVENTS**

La disponibilità di oggetti all’interno di un negozio potrebbe diventare un problema nel momento in cui raggiunge lo 0, soprattutto se mantenesse quel valore per troppo tempo.

Per automatizzare il rifornimento di oggetti nei confronti delle attività commerciali, è stato creato un “evento”, che si attiva ogni 24 ore e porta la disponibilità attuale degli oggetti al loro valore massimo.



**TRANSACTIONS**

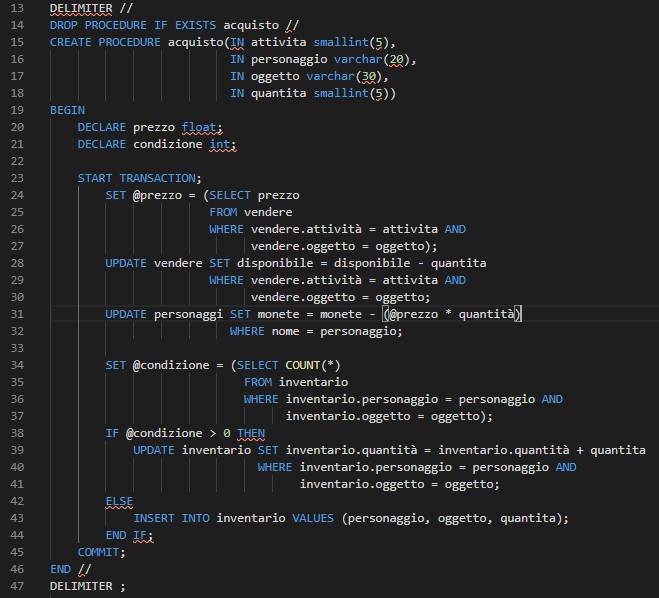
L’aggiornamento delle disponibilità e degli inventari dopo gli acquisti è un’operazione che, obbligatoriamente, deve preservare la consistenza delle informazioni all’interno della base di dati. Proprio per questo motivo si è deciso di costruire il database sfruttando le proprietà transazionali di INNODB Engine.

Nonostante ciò, si è preferito comunque costruire una procedura che si occupa delle transazioni che avvengono durante gli acquisti presso le attività commerciali.

La procedura prende in input il codice dell’attività, il nome del personaggio, il nome dell’oggetto e la quantità acquistata.

Successivamente fa partire una transazione vengono eseguite le seguenti operazioni:

* Ricerca del prezzo dell’oggetto presso l’attività in input
* Decremento della quantità disponibile presso l’attività in base alla quantità acquistata
* Decremento delle monete possedute dal personaggio nell’ordine del prezzo dell’oggetto – la quantità acquistata
* Aggiornamento dell’inventario tramite update della quantità posseduta di quell’oggetto, se il personaggio lo possiede. Se non lo ha già nell’inventario viene inserito in quest’ultimo



**MONGODB**

**DEFINIZIONE STRUTTURA**

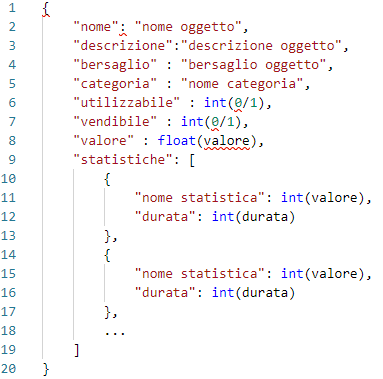
Per la realizzazione del database elysium su un DB NoSQL, si è deciso di prendere l’entità personaggio, l’entità oggetti e tutte le loro relazioni per costruire due collections in MongoDB.

La struttura immaginata è “**embedded array of document references**”: l’elemento centrale della struttura è il personaggio, che oltre ad avere le sue informazioni personali, memorizza al suo interno anche le chiavi dei documenti ad esso relazionati, come la chiave degli oggetti presenti nell’inventario (con relative quantità), quella delle missioni prese in carico (seguite dal loro stato) o quella delle abilità equipaggiate.

La struttura risultante della collection personaggi sarà di questo tipo:



Costruiamo, inoltre, la collection dedicata agli oggetti, contenente le informazioni riguardante direttamente essi e gli effetti di ognuno di loro. La struttura sarà la seguente



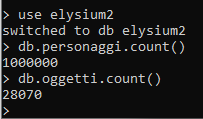
Date queste due collection, utilizzando il nome di un oggetto come identificativo, è possibile ricercare le informazioni relative agli oggetti presenti nell’inventario di un personaggio oppure effettuare la ricerca opposta, cercando tutti i personaggi che posseggono un oggetto con una determinata caratteristica. Per fare ciò basta prendere il nome di un oggetto ottenuto attraverso un’interrogazione su una delle due collection e ricercarlo all’interno della seconda collection.

**OPERAZIONI SU COLLECTIONS**

Per le operazioni di inserimento si è scelto di usare python, per automatizzare il procedimento (script in allegato). Lo script sviluppato si basa sull’eseguire centinaia di migliaia di operazioni dove in ognuna di essa viene definito un personaggio/oggetto, gli vengono assegnati dei valori e poi viene inserito nella collection apposita grazie al modulo pymongo, che permette la connessione diretta tra lo script e il server di MongoDB.

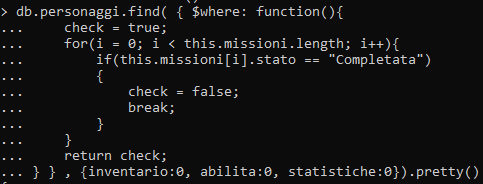
In queste slide verrà utilizzato un database contenente circa un milione record, però, per questioni di spazio occupato, il dump in formato “*.json”* consegnato ne conterrà soltanto 250.000.

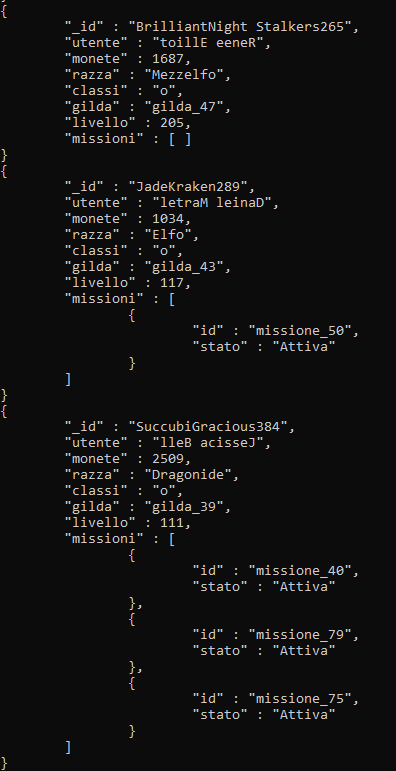
Eseguiamo per prima cosa un’operazione di count per verificare quanti record sono presenti:



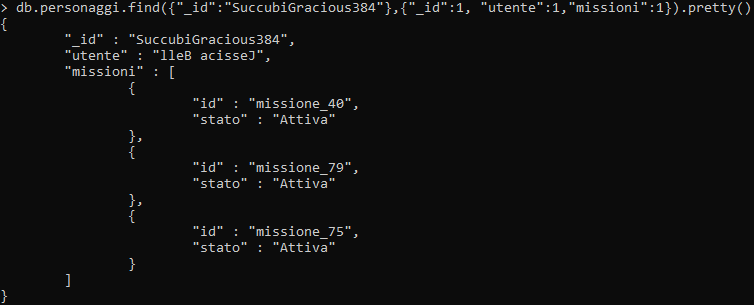
Adesso proviamo ad eseguire interrogazioni più elaborate:

/\*STAMPA L'INFORMAZIONI ED ELENCO MISSIONI DI TUTTI I PERSONAGGI CHE NON HANNO COMPLETATO NEMMENO UNA MISSIONE\*/

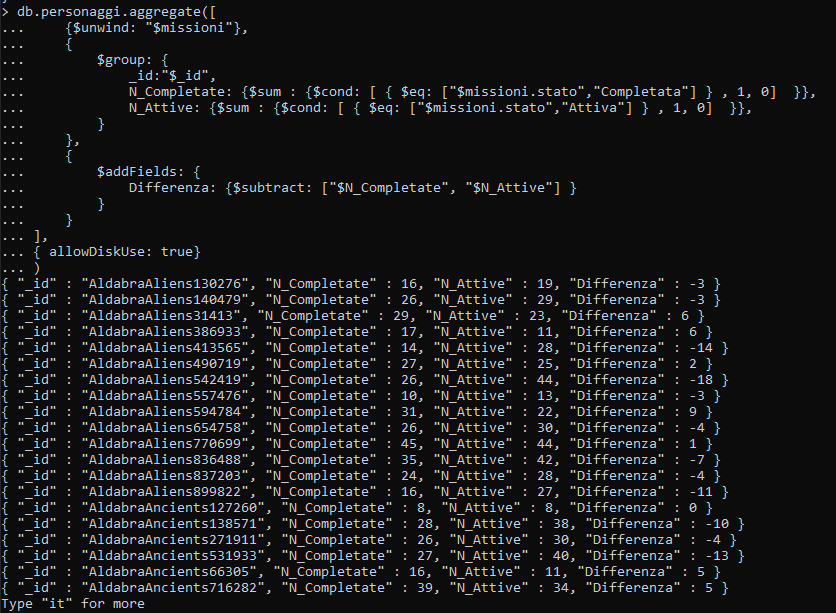




Verifichiamone la correttezza cercando, ad esempio, il personaggio “SuccubiGracious384”:

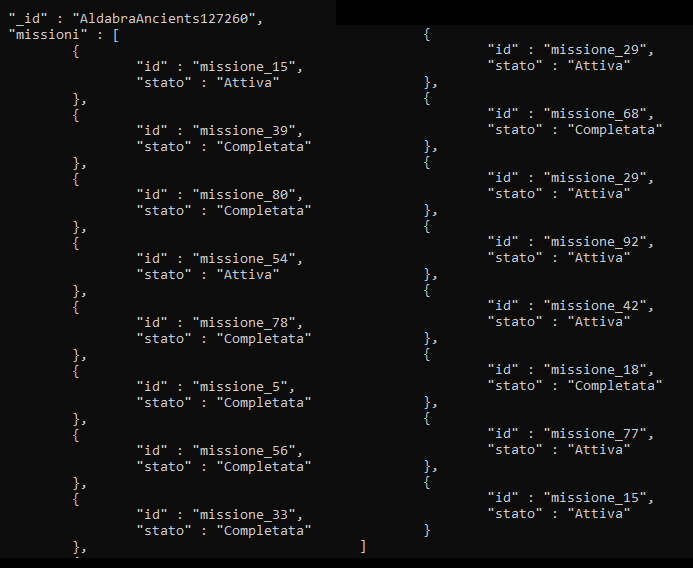


/\* MISSIONI COMPLETATE, ATTIVE E DIFFERENZA TRA LE DUE PER OGNI PERSONAGGIO \*/

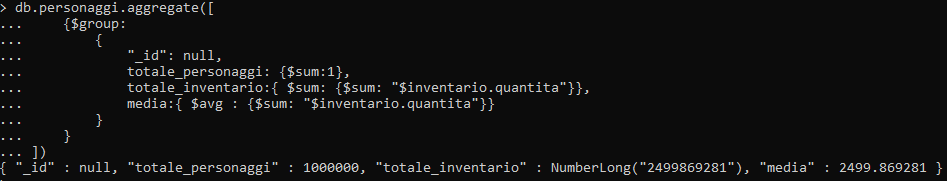


Verifichiamo la correttezza cercando le missioni di “AldabraAncients127260”:

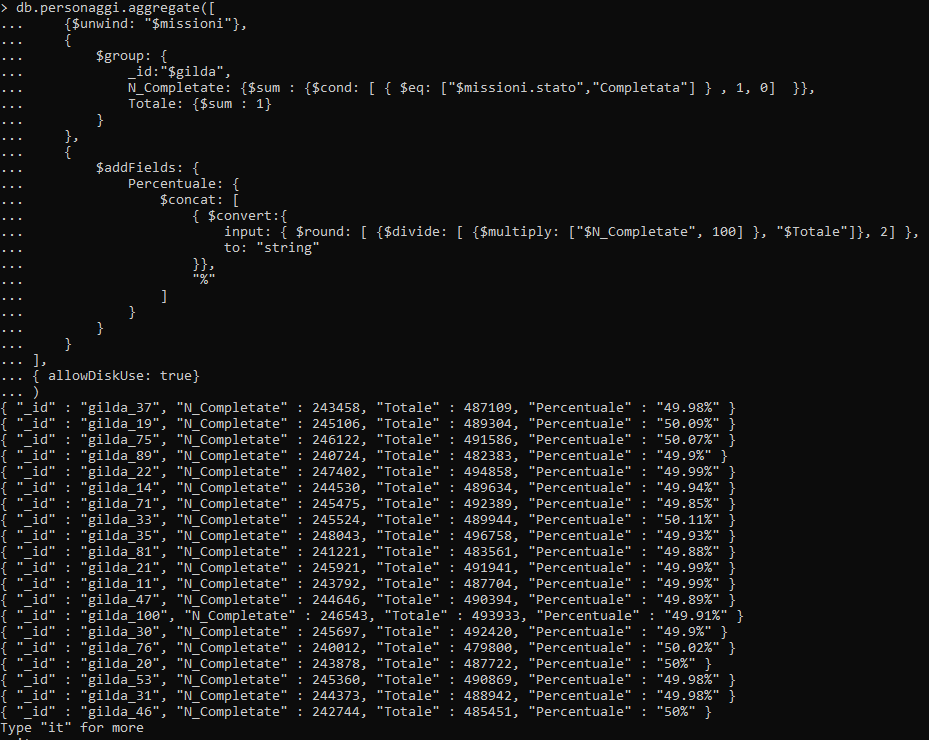
**db.personaggi.find({"\_id":"AldabraAncients127260"},{"\_id":1, "missioni":1}).pretty()**



/\* TOTALE OGGETTI IN INVENTARIO E MEDIA PER PERSONAGGIO \*/



/\*NUMERO E PERCENTUALE MISSIONI COMPLETATE DI TUTTI I PERSONAGGI RAGGRUPPATI PER GILDA\*/



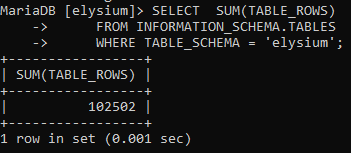
/\* ID PERSONAGGI CHE POSSEGGONO UN'OGGETTO CHE HA EFFETTO SU "Carisma" \*/



**CONFRONTO MongoDB e MySQL**

Proviamo ora a confrontare i tempi di esecuzione di query in MySQL e interrogazioni in MongoDB. Aumentiamo il numero di record in MySQL aumentare i tempi di esecuzione delle query e renderli più facilmente interpretabili.

Verifichiamo il numero di record attuali del nostro database in MySQL:



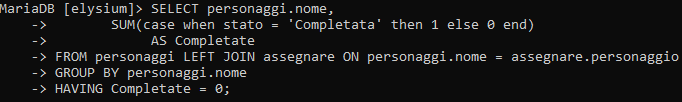
Ora possiamo proseguire con il confrontando dei tempi di esecuzione delle due tecnologie di fronte a 2 quesiti diversi (le cache di entrambi i DB sono state pulite prima dell’esecuzione delle query):

/\*STAMPA L'INFORMAZIONI ED ELENCO MISSIONI DI TUTTI I PERSONAGGI CHE NON HANNO COMPLETATO NEMMENO UNA MISSIONE\*/

**MongoDB (comando già visto nella sezione precedente)**



**MySQL**

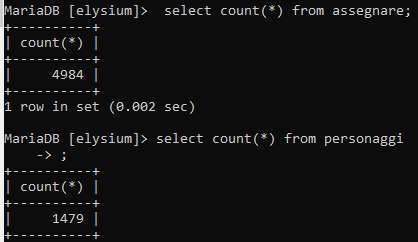




**Analisi:**

MongoDB ha impiegato 287 ms per cercare i personaggi dove nell’array delle missioni non compare lo stato “Completata”. Il lower bound di questa ricerca è il numero di personaggi (1.000.000) mentre nel caso peggiore potrebbe dover scorrere tutte le missioni di tutti i personaggi (se consideriamo una media di 50 missioni per personaggio, circa 50.000.000).

MySQL effettua un left join tra la tabella personaggi e la tabella assegnare e conta, per ogni personaggio, il numero di missioni completate. Infine, ritorna tutti i personaggi per i quali quest’ultimo valore è uguale a 0. Tutte queste operazioni vengono eseguite da MySQL in 4ms su una tabella “assegnare” che contiene circa 5000 records e una tabella personaggi che ne contiene circa 1400.



Nonostante le operazioni eseguite su MySQL siano più pesanti rispetto al semplice scorrimento di MongoDB, il rapporto tra i tempi di esecuzione è di circa 1:72, mentre il rapporto tra i record memorizzati è dell’ordine delle centinaia.

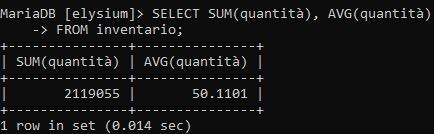
Possiamo quindi dire che questa struttura di DB NoSQL velocizza di molto l’esecuzione di questo tipo di interrogazioni rispetto ad un DB MySQL, poiché i dati sono contenuti in un’unica collection in MongoDB, al contrario di MySQL che deve effettuare una join tra 2 tabelle.

/\* TOTALE OGGETTI IN INVENTARIO E MEDIA PER PERSONAGGIO \*/

**MongoDB (comando già visto nella sezione precedente):**

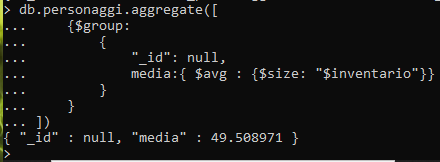


**MySQL**



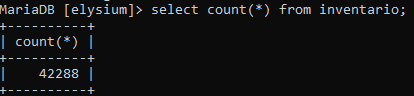
**Analisi:**

MongoDB ha impiegato 23938 ms per cercare per scorrere l’inventario di tutti i personaggi. Un lower bound per questa operazione è pari al numero di personaggi (1.000.000). Considerando una media di circa 49 oggetti per ogni personaggio, possiamo affermare che sono state eseguite circa 49.000.000 di somme.



(Media di oggetti per personaggio)

MySQL scorre l’intera tabella *inventario* e somma i valori presenti sulla colonna *quantità*. La struttura non è nidificata quindi il numero di controlli e pari al numero di righe della tabella *inventario*, che nel nostro caso è pari a: 42.288



Stiamo quindi confrontando i tempi di esecuzione di due query che si occupano semplicemente di scorrere tutti i record presenti nella tabella/collection.

Considerando che il rapporto tra il numero dei record analizzati da MySQL e MongoDB è di circa 1:1170 e il rapporto tra i tempi di esecuzione è di circa 1:1710, possiamo dire che su semplici operazioni di scorrimento su una tabella, MySQL supera MongoDB in velocità