ELYSIUM

INDICE

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.

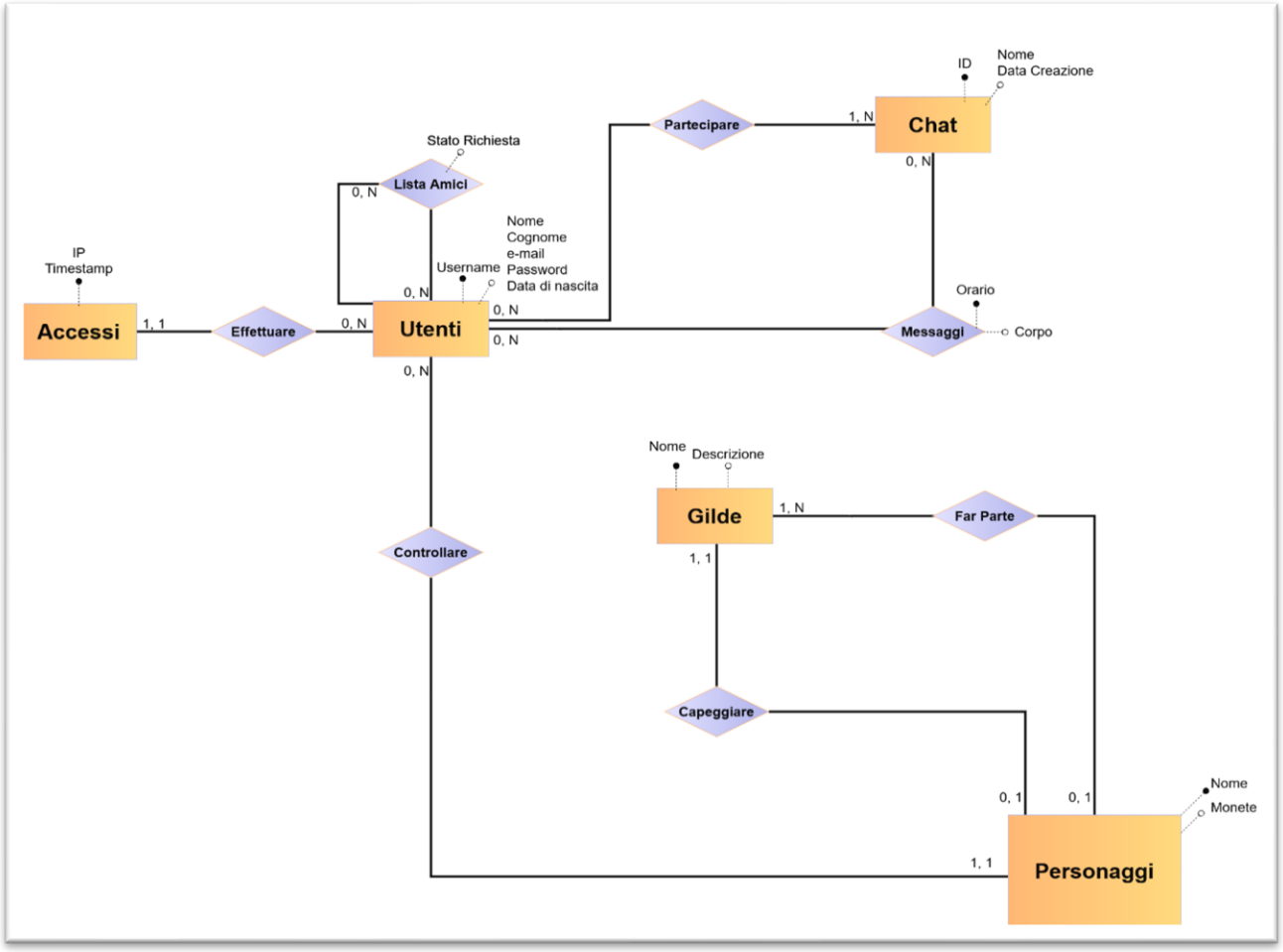
La seconda categoria invece è relativa ai dati di un utente e di un personaggio, come accessi, messaggi, chat, liste di amici, razza, classe, statistiche, inventario ed abilita equipaggiate. Occuparsi di queste informazioni in maniera centralizzata serve ad evitare eventuali problemi di consistenza e 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.

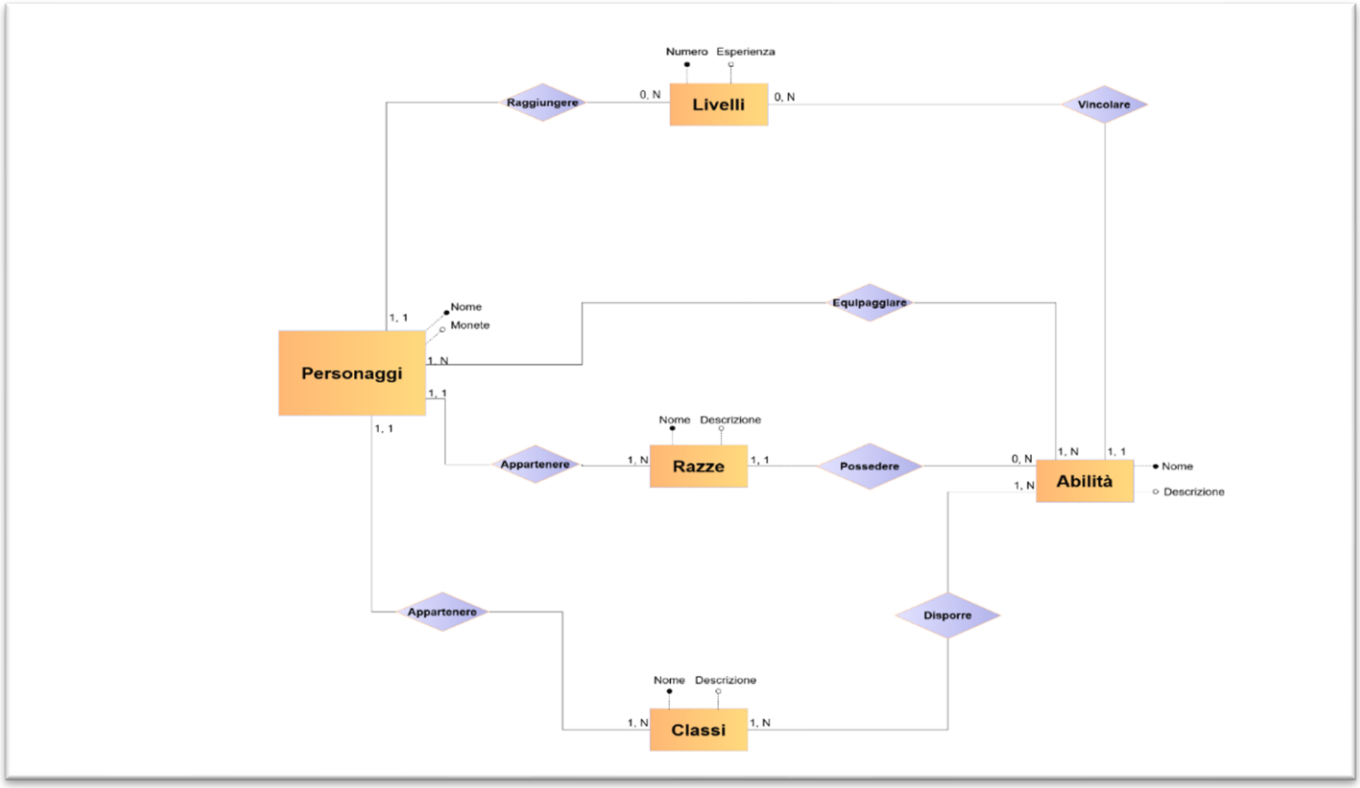
Definiamo ora requisiti e casi particolari:

- 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.

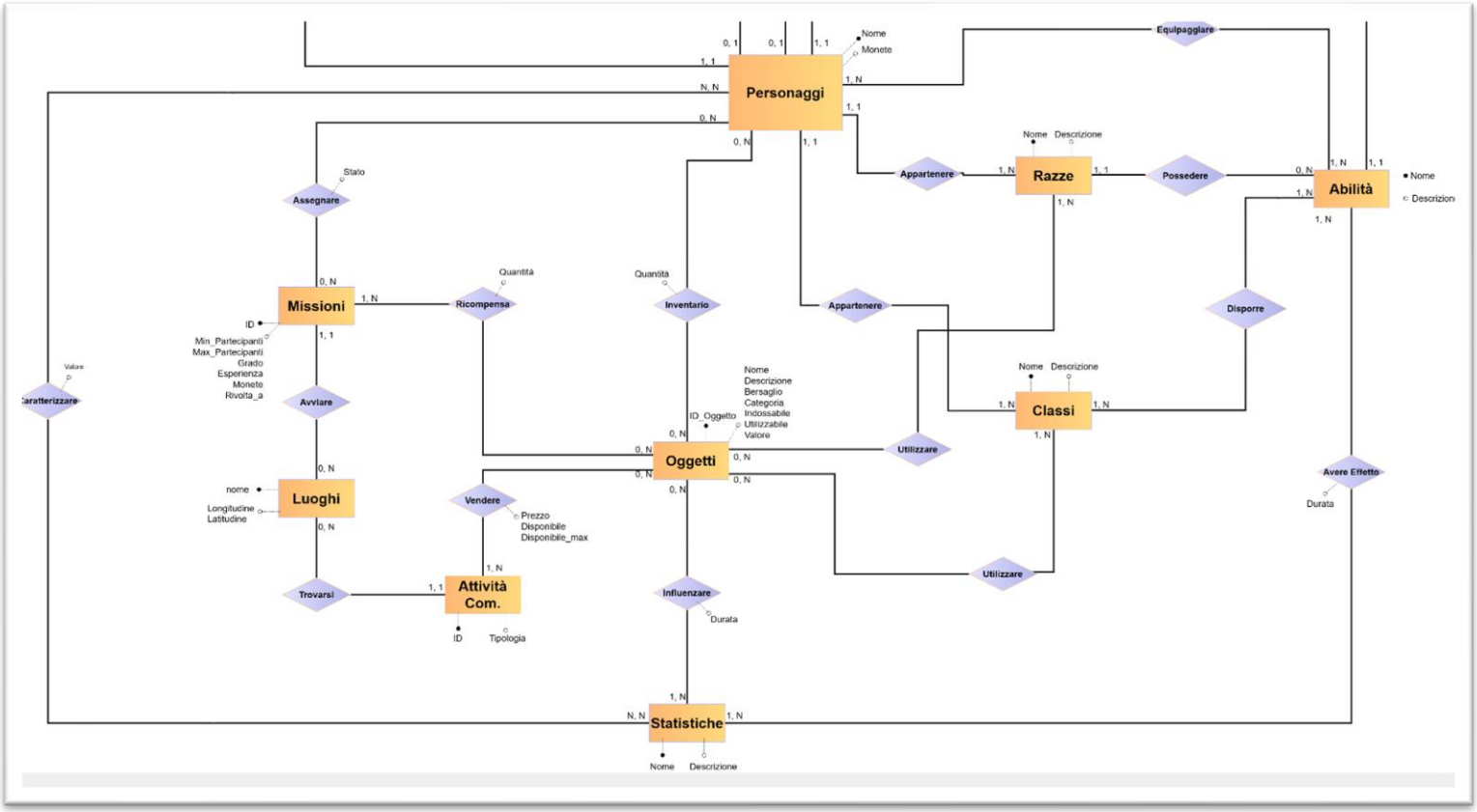


- Costruiamo l'entità livelli, caratterizzata da numero livello ed esperienza necessaria per raggiungerlo per mantenere consistenza e scalabilità. Il livello può caratterizzare un personaggio e allo stesso tempo vincolare un'abilità.
- Costruiamo anche le entità razze e classi, identificate dal loro 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.

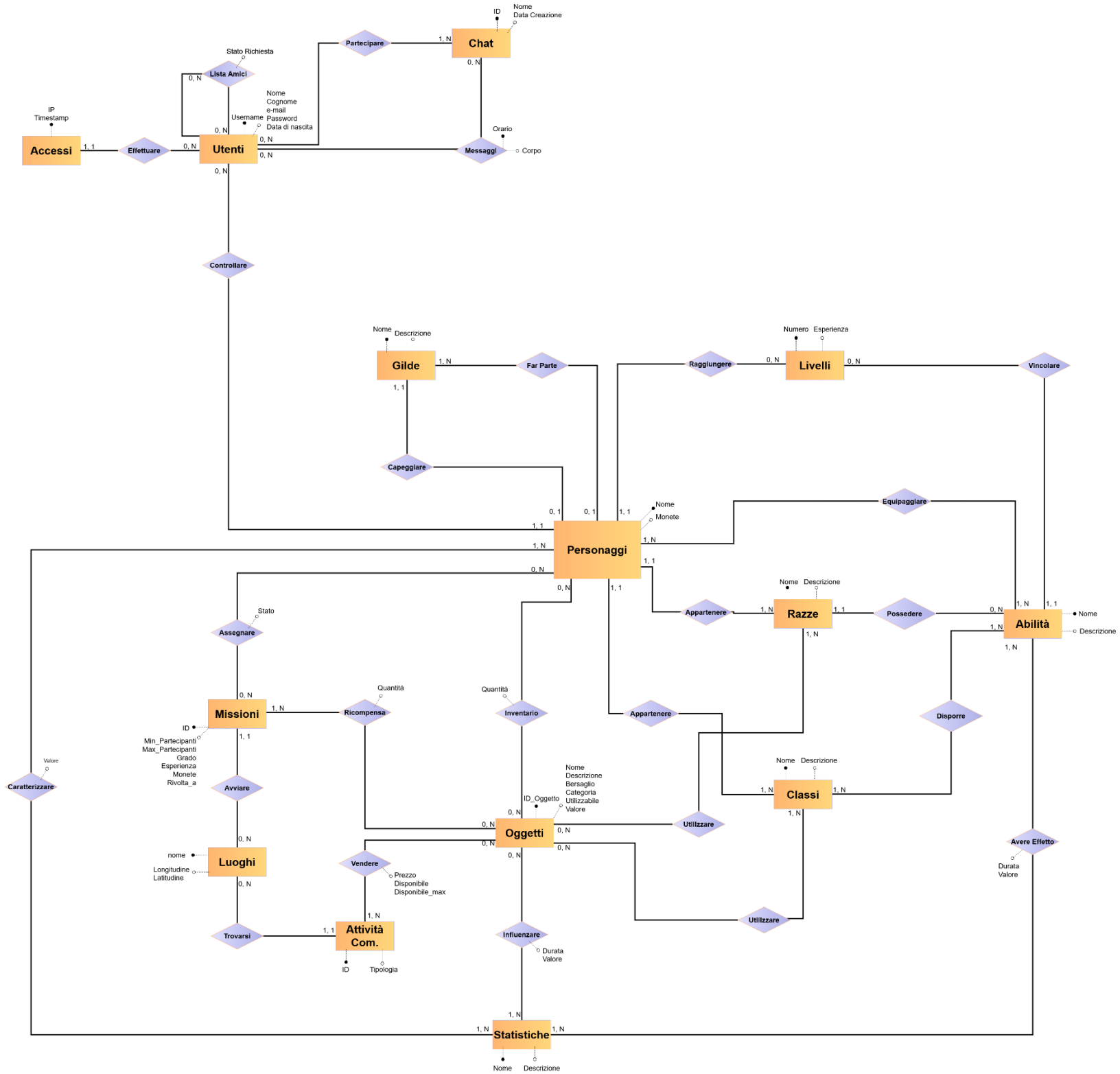


- L'entità statistiche è caratterizzata solo dal nome della statistica ma è molto importante rappresentarla. Avere a disposizione questa entità e non trasformare i suoi record in attributi delle tabelle collegate aiuta in diversi casi:
    1. In caso di inserimenti di nuove statistiche basta soltanto 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



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

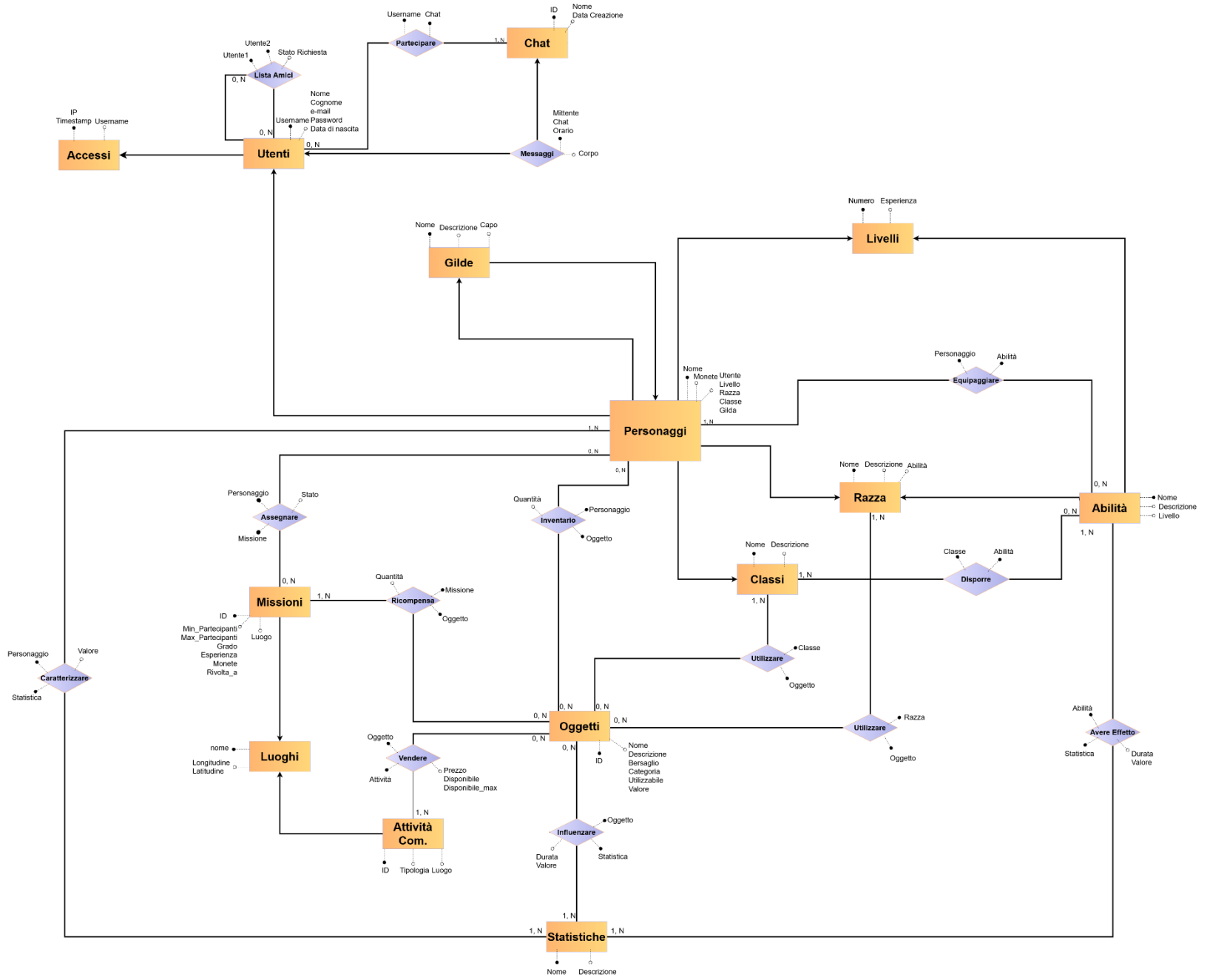
Elenco relazioni:

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



CREAZIONE TABELLE

```sql
DROP DATABASE IF EXISTS Elysium;
CREATE DATABASE Elysium;
USE Elysium;

CREATE TABLE Utenti(
    username varchar(20) not NULL,
    nome  varchar(20) not NULL,
    cognome varchar(25) not NULL,
    email varchar(255) not NULL unique,
    pass char(40) not NULL,
    data_di_nascita DATE not NULL,

    PRIMARY KEY(username)
) ENGINE=INNODB;


CREATE TABLE Accessi
(
    ip char(15) not NULL,
    orario TIMESTAMP not NULL DEFAULT CURRENT_TIMESTAMP,
    username varchar(20) not NULL,

    PRIMARY KEY (ip, orario),
    FOREIGN KEY(username) REFERENCES Utenti(username)
      ON DELETE CASCADE
      ON UPDATE CASCADE
);


CREATE TABLE Amici
(
    utente1 varchar(20) not NULL,
    utente2 varchar(20) not NULL,
    stato_richiesta boolean not NULL,

    CONSTRAINT chk_utenti CHECK(utente1 <> utente2),

    PRIMARY KEY(utente1, utente2),
    FOREIGN KEY(utente1) REFERENCES Utenti(username)
        ON DELETE CASCADE
        ON UPDATE CASCADE,
    FOREIGN KEY(utente2) REFERENCES Utenti(username)
        ON DELETE CASCADE
        ON UPDATE CASCADE
) ENGINE=INNODB;


CREATE TABLE Messaggi
(
    mittente varchar(20) not NULL,
    chat int unsigned not NULL,
    corpo varchar(1024 ) not NULL,
    orario TIMESTAMP not NULL DEFAULT NOW(),

    PRIMARY KEY(mittente, chat,orario)
    FOREIGN KEY(mittente) REFERENCES Utenti(username)
        ON DELETE CASCADE
        ON UPDATE CASCADE,
    FOREIGN KEY(chat) REFERENCES Chat(ID)
        ON DELETE CASCADE
        ON UPDATE CASCADE
) ENGINE=INNODB;


CREATE TABLE Chat
(
    ID int unsigned AUTO_INCREMENT not NULL,
    nome varchar(20) not NULL,
    data_creazione TIMESTAMP not NULL,

    PRIMARY KEY(ID)
) ENGINE=INNODB;

CREATE TABLE Partecipare
(
    username varchar(20) not NULL,
    id_chat int unsigned not NULL,

    PRIMARY KEY(username, id_chat),
    FOREIGN KEY(username) REFERENCES Utenti(username)
        ON DELETE CASCADE
        ON UPDATE CASCADE,
    FOREIGN KEY(id_chat) REFERENCES Chat(ID)
        ON DELETE CASCADE
        ON UPDATE CASCADE
) ENGINE=INNODB;


CREATE TABLE Livelli
(
    livello tinyint unsigned not NULL,
    esperienza int unsigned not NULL,

    PRIMARY KEY(livello)
) ENGINE=INNODB;


CREATE TABLE Abilita
(
    nome varchar(30) not NULL,
    descrizione varchar(512) not NULL,
    livello tinyint unsigned not NULL DEFAULT 1,

    PRIMARY KEY(nome),
    FOREIGN KEY(livello) REFERENCES Livelli(livello)
        ON UPDATE CASCADE
) ENGINE=INNODB;


CREATE TABLE Razze
(
    nome varchar(15) not NULL,
    descrizione varchar(512) not NULL,
    abilita varchar(30) not NULL UNIQUE,

    PRIMARY KEY(nome),
    FOREIGN KEY(abilita) REFERENCES Abilita(nome)
        ON DELETE RESTRICT
        ON UPDATE CASCADE
) ENGINE=INNODB;


CREATE TABLE Classi
(
    nome varchar(15) not NULL,
    descrizione varchar(512) not NULL,

    PRIMARY KEY(nome)
) ENGINE=INNODB;
```

```sql
CREATE TABLE Disporre
(
    classe varchar(15) not NULL,
    abilita varchar(30) not NULL,

    PRIMARY KEY(classe, abilita),
    FOREIGN KEY(classe) REFERENCES Classi(nome)
        ON DELETE CASCADE
        ON UPDATE CASCADE,
    FOREIGN KEY(abilita) REFERENCES Abilita(nome)
        ON DELETE CASCADE
        ON UPDATE CASCADE
) ENGINE=INNODB;



CREATE TABLE Statistiche
(
    nome varchar(15) not NULL,

    PRIMARY KEY(nome)
) ENGINE=INNODB;


CREATE TABLE Avere_effetto
(
    abilita varchar(30) not NULL,
    statistica varchar(15) not NULL,
    durata int unsigned not NULL,
    valore smallint not NULL,

    PRIMARY KEY(abilita, statistica),
    FOREIGN KEY(abilita) REFERENCES Abilita(nome)
        ON DELETE CASCADE
        ON UPDATE CASCADE,
    FOREIGN KEY(statistica) REFERENCES Statistiche(nome)
        ON DELETE CASCADE
        ON UPDATE CASCADE
) ENGINE=INNODB;


CREATE TABLE Personaggi
(
    nome varchar(20) not NULL,
    monete float unsigned not NULL,
    utente varchar(20) not NULL,
    razza varchar(15) not NULL,
    classe varchar(15) not NULL,
    livello tinyint unsigned not NULL,
    gilda varchar(20) DEFAULT NULL,

    PRIMARY KEY(nome),
    FOREIGN KEY(utente) REFERENCES Utenti(username)
        ON DELETE CASCADE
        ON UPDATE CASCADE,
    FOREIGN KEY(razza) REFERENCES Razze(nome)
        ON DELETE RESTRICT
        ON UPDATE CASCADE,
    FOREIGN KEY(classe) REFERENCES Classi(nome)
        ON DELETE RESTRICT
        ON UPDATE CASCADE,
    FOREIGN KEY(livello) REFERENCES Livelli(livello)
        ON DELETE RESTRICT
        ON UPDATE CASCADE

) ENGINE=INNODB;
```

```sql
    nome varchar(20) not NULL,
    descrizione varchar(512) DEFAULT " ",
    capo varchar(20) not NULL UNIQUE,

    PRIMARY KEY(nome),
    FOREIGN KEY(capo) REFERENCES Personaggi(nome)
        ON DELETE CASCADE
        ON UPDATE CASCADE
)ENGINE=INNODB;

ALTER TABLE Personaggi
    ADD CONSTRAINT FOREIGN KEY(gilda) REFERENCES Gilde(nome)
        ON DELETE SET NULL
        ON UPDATE CASCADE;



CREATE TABLE Equipaggiare
(
    personaggio varchar(20) not NULL,
    abilita varchar(30) not NULL,

    PRIMARY KEY(personaggio, abilita),
    FOREIGN KEY(personaggio) REFERENCES Personaggi(nome)
        ON DELETE CASCADE
        ON UPDATE CASCADE,
    FOREIGN KEY(abilita) REFERENCES abilita(nome)
        ON DELETE CASCADE
        ON UPDATE CASCADE
) ENGINE=INNODB;



CREATE TABLE Caratterizzare
(
    personaggio varchar(20) not NULL,
    statistica varchar(15) not NULL,
    valore smallint unsigned not NULL DEFAULT 1,

    PRIMARY KEY(personaggio, statistica),
    FOREIGN KEY(personaggio) REFERENCES Personaggi(nome)
        ON DELETE CASCADE
        ON UPDATE CASCADE,
    FOREIGN KEY(statistica) REFERENCES Statistiche(nome)
        ON DELETE CASCADE
        ON UPDATE CASCADE
) ENGINE=INNODB;



CREATE TABLE Oggetti
(
    nome varchar(30) not NULL,
    descrizione varchar(512) not NULL,
    bersaglio varchar(20) not NULL DEFAULT 'sè',
    categoria varchar(20) not NULL,
    utilizzabile boolean not NULL DEFAULT 1,
    vendibile boolean not NULL DEFAULT 1,
    valore float not NULL DEFAULT 0,

    CONSTRAINT chk_bersaglio CHECK (bersaglio IN ('sè', 'alleato singolo',
                                    'alleato gruppo(5m)', 'alleato gruppo(10m)',
                                    'nemico singolo', 'nemico gruppo(5m)',
                                    'nemico gruppo(10m)')),
    CONSTRAINT chk_categoria CHECK (categoria IN ('armatura', 'arma', 'pozione',
                                    'pergamena', 'genere alimentare', 'abito',
                                    'missione', 'varie')),

    PRIMARY KEY(nome)
) ENGINE=INNODB;
```

```sql
CREATE TABLE Influenzare
(
    oggetto varchar(30) not NULL,
    statistica varchar(15) not NULL,
    durata int unsigned not NULL,
    valore smallint not NULL DEFAULT 0,

    PRIMARY KEY(oggetto, statistica),
    FOREIGN KEY(oggetto) REFERENCES Oggetti(nome)
        ON DELETE CASCADE
        ON UPDATE CASCADE,
    FOREIGN KEY(statistica) REFERENCES Statistiche(nome)
        ON DELETE CASCADE
        ON UPDATE CASCADE
) ENGINE=INNODB;



CREATE TABLE Utilizzare_Classe
(
    classe varchar(15) not NULL,
    oggetto varchar(30) not NULL,

    PRIMARY KEY(oggetto, classe),
    FOREIGN KEY(oggetto) REFERENCES Oggetti(nome)
        ON DELETE CASCADE
        ON UPDATE CASCADE,
    FOREIGN KEY(classe) REFERENCES Classi(nome)
        ON DELETE CASCADE
        ON UPDATE CASCADE
) ENGINE=INNODB;



CREATE TABLE Utilizzare_Razza
(
    razza varchar(15) not NULL,
    oggetto varchar(15) not NULL,

    PRIMARY KEY(oggetto, razza),
    FOREIGN KEY(oggetto) REFERENCES Oggetti(nome)
        ON DELETE CASCADE
        ON UPDATE CASCADE,
    FOREIGN KEY(razza) REFERENCES Razze(nome)
        ON DELETE CASCADE
        ON UPDATE CASCADE
) ENGINE=INNODB;



CREATE TABLE Inventario
(
    personaggio varchar(20) not NULL,
    oggetto varchar(30) not NULL,
    quantità smallint unsigned not NULL DEFAULT 1,

    CONSTRAINT chk_quantita CHECK (quantità > 0),

    PRIMARY KEY(personaggio, oggetto),
    FOREIGN KEY(personaggio) REFERENCES Personaggi(nome)
        ON DELETE CASCADE
        ON UPDATE CASCADE,
    FOREIGN KEY(oggetto) REFERENCES Oggetti(nome)
        ON DELETE CASCADE
        ON UPDATE CASCADE
) ENGINE=INNODB;
```

```sql
CREATE TABLE Luoghi
(
    nome varchar(20) not NULL,
    latitudine float not NULL UNIQUE,
    longitudine float not NULL UNIQUE,

    PRIMARY KEY(nome)
) ENGINE=INNODB;



CREATE TABLE Missioni
(
    ID smallint unsigned not NULL AUTO_INCREMENT,
    min_partecipanti smallint not NULL DEFAULT 1,
    max_partecipanti smallint not NULL DEFAULT 1,
    grado tinyint unsigned not NULL DEFAULT 1,
    esperienza int unsigned not NULL,
    monete int unsigned not NULL DEFAULT 0,
    rivolta_a varchar(10) not NULL DEFAULT 'pubblica',
    luogo varchar(20) not NULL,

    CONSTRAINT chk_partecipanti CHECK(min_partecipanti <= max_partecipanti)
    CONSTRAINT chk_rivolto CHECK (rivolta_a IN ('pubblica', 'gilda')),

    PRIMARY KEY(id),
    FOREIGN KEY(luogo) REFERENCES Luoghi(nome)
        ON DELETE RESTRICT
        ON UPDATE CASCADE
) ENGINE=INNODB;



CREATE TABLE Assegnare
(
    personaggio varchar(20) not NULL,
    missione smallint unsigned not NULL,
    stato varchar(10) not NULL,

    CONSTRAINT chk_stato CHECK(stato IN ('Attiva', 'Completata') ),

    PRIMARY KEY(personaggio, missione),
    FOREIGN KEY(personaggio) REFERENCES Personaggi(nome)
        ON DELETE CASCADE
        ON UPDATE CASCADE,
    FOREIGN KEY(missione) REFERENCES Missioni(ID)
        ON UPDATE CASCADE
) ENGINE=INNODB;



CREATE TABLE Ricompense
(
    missione smallint unsigned not NULL,
    oggetto varchar(30) not NULL,
    quantità smallint unsigned not NULL DEFAULT 1,

    PRIMARY KEY(missione, oggetto),
    FOREIGN KEY(missione) REFERENCES Missioni(ID)
        ON UPDATE CASCADE,
    FOREIGN KEY(oggetto) REFERENCES Oggetti(nome)
        ON DELETE CASCADE
        ON UPDATE CASCADE
) ENGINE=INNODB;
```

```sql
CREATE TABLE Attività_commerciali
(
    ID smallint unsigned not NULL AUTO_INCREMENT,
    tipologia varchar(15) not NULL,
    luogo varchar(20) not NULL,

    CONSTRAINT chk_tipologia CHECK (tipologia IN ('taverne', 'negozi',
                                    'botteghe', 'venditori', 'ambulanti')),

    PRIMARY KEY(ID),
    FOREIGN KEY(luogo) REFERENCES Luoghi(nome)
        ON DELETE RESTRICT
        ON UPDATE CASCADE
) ENGINE=INNODB;


CREATE TABLE Vendere
(
    oggetto varchar(30) not NULL,
    attività smallint unsigned not NULL,
    prezzo float not NULL,
    'disponibile' int unsigned NOT NULL DEFAULT 0,
    'max' int unsigned NOT NULL,

    CONSTRAINT check_disponibilita CHECK (max >= disponibile);

    PRIMARY KEY(oggetto, attività),
    FOREIGN KEY(oggetto) REFERENCES Oggetti(nome)
        ON DELETE CASCADE
        ON UPDATE CASCADE,
    FOREIGN KEY(attività) REFERENCES Attività_commerciali(ID)
        ON DELETE CASCADE
        ON UPDATE CASCADE
) ENGINE=INNODB;
```

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:

```
INSERT INTO `abilita` VALUES ('Absolve','descrizione abilità: Absolve',31),('Ambush','descrizione abilità: Ambush',200),
('Backstab','descrizione abilità: Backstab',99),('Barrage','descrizione abilità: Barrage',226),
('Behemoth Slash','descrizione abilità: Behemoth Slash',17),('Betrayal Slam','descrizione abilità: Betrayal Slam',68),
('Blackout','descrizione abilità: Blackout',157),('Blazing Strike','descrizione abilità: Blazing Strike',161),
('Blocking Bash','descrizione abilità: Blocking Bash',195),('Bloodlust','descrizione abilità: Bloodlust',35),
('Bombardment','descrizione abilità: Bombardment',148),('Chop','descrizione abilità: Chop',255),
('Chopchop','descrizione abilità: Chopchop',23),('Cleaving Strikes','descrizione abilità: Cleaving Strikes',213),
('Clobber','descrizione abilità: Clobber',237),('Commanding Slice','descrizione abilità: Commanding Slice',126),
('Corrupt','descrizione abilità: Corrupt',192),('Corrupting Strikes','descrizione abilità: Corrupting Strikes',152),
('Corrupting Thrust','descrizione abilità: Corrupting Thrust',200),('Crackling Shot','descrizione abilità: Crackling Shot',62),
('Cripple','descrizione abilità: Cripple',102),('Destruction','descrizione abilità: Destruction',116),
('Devastate','descrizione abilità: Devastate',109),('Disassemble','descrizione abilità: Disassemble',134),
('Disrupt','descrizione abilità: Disrupt',172),('Disruption Burst','descrizione abilità: Disruption Burst',123),
('Disruption Gash','descrizione abilità: Disruption Gash',128),('Dissolve','descrizione abilità: Dissolve',6),
('Distracting Bash','descrizione abilità: Distracting Bash',157),('Division Gash','descrizione abilità: Division Gash',253),
ò('Enveloping Gash','descrizione abilità: Enveloping Gash',51),('Exorcism','descrizione abilità: Exorcism',14),
('Fade','descrizione abilità: Fade',30),('Ferocity','descrizione abilità: Ferocity',164),
('Fire Strike','descrizione abilità: Fire Strike',216),('Flare','descrizione abilità: Flare',195),
('Frenzied Slash','descrizione abilità: Frenzied Slash',32),('Garrote','descrizione abilità: Garrote',195),
('Howl','descrizione abilità: Howl',162),('Immobilizing Rip','descrizione abilità: Immobilizing Rip',227),
('Impale','descrizione abilità: Impale',140),('Incinerate','descrizione abilità: Incinerate',17),
('Judgment Force','descrizione abilità: Judgment Force',144),('Knockout Force','descrizione abilità: Knockout Force',14),
('Lacerate','descrizione abilità: Lacerate',81),('Lava Gash','descrizione abilità: Lava Gash',111),
('Maim','descrizione abilità: Maim',60),('Mangle','descrizione abilità: Mangle',117),
('Mighty Slam','descrizione abilità: Mighty Slam',186),('Mighty Strike','descrizione abilità: Mighty Strike',169),
('Mind Disrupt','descrizione abilità: Mind Disrupt',185),('Mirror Slice','descrizione abilità: Mirror Slice',178),
('Molten Rip','descrizione abilità: Molten Rip',5),('Mortal Shot','descrizione abilità: Mortal Shot',85),
('Mutilate','descrizione abilità: Mutilate',254),('Needle Strike','descrizione abilità: Needle Strike',173),
('Overload','descrizione abilità: Overload',216),('Paralyze','descrizione abilità: Paralyze',222),
('Perforating Crash','descrizione abilità: Perforating Crash',58),('Pierce','descrizione abilità: Pierce',101),
('Piercing Rip','descrizione abilità: Piercing Rip',5),('Rampage Slash','descrizione abilità: Rampage Slash',201),
('Rupture','descrizione abilità: Rupture',22),('Sacrificial Force','descrizione abilità: Sacrificial Force',79),
('Savage Shot','descrizione abilità: Savage Shot',247),('Savage Slash','descrizione abilità: Savage Slash',81),
('Shadow Strikes','descrizione abilità: Shadow Strikes',206),('Shock','descrizione abilità: Shock',198),
('Shocking Bash','descrizione abilità: Shocking Bash',33),('Shocking Strike','descrizione abilità: Shocking Strike',88),
('Shove','descrizione abilità: Shove',120),('Shriek','descrizione abilità: Shriek',195)
,('Singing Bash','descrizione abilità: Singing Bash',230),('Singing Shot','descrizione abilità: Singing Shot',178),
('Sinister Shot','descrizione abilità: Sinister Shot',14),('Skiver','descrizione abilità: Skiver',2),
('Slide','descrizione abilità: Slide',181),('Smoldering Smash','descrizione abilità: Smoldering Smash',74),
('Smother','descrizione abilità: Smother',101),('Spite','descrizione abilità: Spite',109),
('Stealth Smash','descrizione abilità: Stealth Smash',217), ('Sting','descrizione abilità: Sting',71),
('Summoning Thrust','descrizione abilità: Summoning Thrust',61),('Sundering Strikes','descrizione abilità: Sundering Strikes',172),
('Titan Smash','descrizione abilità: Titan Smash',195),('Trance Slash','descrizione abilità: Trance Slash',219),
('Turbulence','descrizione abilità: Turbulence',123),('Vanish','descrizione abilità: Vanish',28),
('Vanishing Smash','descrizione abilità: Vanishing Smash',56),
('Wail','descrizione abilità: Wail',39),('Whispering Crash','descrizione abilità: Whispering Crash',53);
```

```sql
INSERT INTO `personaggi` VALUES ('AlmondGoblins',8046,'itTucJud34','Elfo','Bardo',1,'gilda41'),
('AlpineAncients',6831,'esMasTer32','Mezzelfo','Bardo',1,'gilda41'),
('AmazonNewts',2132,'PieLee20','Umano','Barbaro',122,'gilda93'),
('AmberJackalopes',8055,'hnRitDap15','Mezzorco','Ranger',73,'gilda33'),
('AmphibianGenies',585,'iaColMar22','Halfling','Paladino',102,'gilda69'),
('AncientGorgons',7111,'onMarJas40','Umano','Stregone',79,'gilda3'),
('AquaMenace',9699,'ttPenSco96','Nano','Druido',49,'gilda44'),
('ArcticDraugr',4943,'lBelPau49','Nano','Mago',125,'gilda99'),
('ArcticGhouls',8330,'KniKay75','Nano','Bardo',77,'gilda99'),
('ArcticSphinxes',6746,'onMarJas40','Umano','Mago',1,'gilda19'),
('AshDemons',722,'frRilJef89','Mezzelfo','Paladino',1,'gilda99'),
('AshSuccubi',2111,'arWalEdw91','Mezzorco','Ladro',83,'gilda60'),
('AutumnBiters',8958,'nPatJoh48','Nano','Barbaro',128,'gilda63'),
('AutumnWispmother',9519,'isMeiChr6','Mezzelfo','Barbaro',71,'gilda40'),
('AwokenSouleaters',6451,'nePadKen1','Halfling','Druido',73,'gilda63'),
('BeardedBeasts',1073,'arWalEdw91','Mezzelfo','Guerriero',1,'gilda38'),
('BeardedHellhounds',6874,'anHatSus80','Halfling','Ladro',105,'gilda7'),
('BeardedValkyries',2009,'elTruRos76','Nano','Druido',49,'gilda47'),
('BlazingWyrms',7464,'nHawJoh92','Mezzorco','Chierico',66,'gilda23'),
('BlondeHobgoblins',1360,'DavSue82','Umano','Druido',118,'gilda70'),
('BloodGhouls',6817,'maCroTho70','Elfo','Guerriero',1,'gilda94'),
('BlueMummies',1487,'nAikJoh33','Mezzelfo','Druido',1,'gilda83'),
('BlueWendigos',2346,'nPatJoh48','Elfo','Warlock',1,'gilda99'),
('BlushingWispmother',7247,'idBacDav78','Mezzorco','Paladino',1,'gilda61'),
('BlushingYetis',800,'isGloMel66','Tiefling','Bardo',1,'gilda79'),
('BogBigfoots',1438,'arWalEdw91','Halfling','Monaco',1,'gilda18'),
('BogBunyips',4586,'rlGuzShi4','Elfo','Ladro',120,'gilda57'),
('BogSuccubi',2473,'dlJohBra14','Mezzelfo','Monaco',1,'gilda6'),
('BrassGorgons',2329,'iCraJod38','Dragonide','Monaco',152,'gilda99'),
('BrightAngels',403,'ixMeiFel85','Halfling','Ladro',59,'gilda60'),
('BrightHags',2326,'onMarJas40','Dragonide','Ranger',1,'gilda25'),
('BronzeBiters',9342,'epBerJos52','Elfo','Paladino',42,'gilda0'),
('BronzeGoblins',9772,'kMarMar8','Halfling','Ladro',89,'gilda35'),
('BrownTaurens',9523,'lBelPau49','Dragonide','Ladro',1,'gilda83'),
('BrownWyrms',7264,'iaMayKen11','Mezzorco','Barbaro',1,'gilda6'),
('BurntFauns',1379,'baSkiBar16','Dragonide','Guerriero',123,'gilda46'),
('BurntTroglodytes',528,'isMeiChr6','Dragonide','Druido',15,'gilda99'),
('BurrowingMelusines',2863,'buDamWil60','Nano','Ladro',61,'gilda97'),
('CavalierChangelings',6995,'itTucJud34','Dragonide','Bardo',168,'gilda99'),
('CavernSouleaters',533,'haWatMic71','Tiefling','Barbaro',1,'gilda14'),
('ChaosMummies',9760,'iCraJod38','Mezzelfo','Stregone',132,'gilda47'),
('ChaosNewts',7853,'neBakRod58','Tiefling','Druido',65,'gilda45'),
('ChaosNymphs',6763,'heJohKat77','Tiefling','Stregone',61,'gilda49'),
('CloakedLeapers',543,'ieFelJul19','Halfling','Druido',1,'gilda99'),
('CloakedTerrors',5599,'onMarJas40','Umano','Bardo',71,'gilda70'),
('CoastalLeviathans',6637,'epBerJos52','Tiefling','Warlock',37,'gilda25'),
('CoiledWerewolves',6528,'KniKay75','Elfo','Barbaro',1,'gilda20'),
('CommonLiches',4878,'rlGuzShi4','Tiefling','Barbaro',1,'gilda77'),
('CopperHydras',3035,'eEvaDav63','Tiefling','Monaco',1,'gilda73'),
('CreepingChangelings',7204,'SinEva83','Mezzelfo','Barbaro',125,'gilda34'),
('CrestedLynxes',69,'umLauTat31','Halfling','Barbaro',61,'gilda41'),
('CrimsonChimeras',776,'anHatSus80','Mezzorco','Chierico',1,'gilda99'),
('CrimsonLeapers',3576,'aMckDan98','Elfo','Paladino',1,'gilda99'),
('CrimsonMonsters',6276,'saAndAni9','Halfling','Druido',102,'gilda97'),
('CrossedMermaids',6181,'eeCalNor18','Tiefling','Monaco',73,'gilda14'),
('CrossedUnicorns',6761,'DavSue82','Mezzorco','Warlock',72,'gilda83'),
('DarkCrawlers',4704,'KniKay75','Tiefling','Paladino',68,'gilda64'),
('DarkLeapers',8450,'DavSue82','Dragonide','Stregone',79,'gilda19'),
('DeathlordPegasi',7409,'isGloMel66','Nano','Druido',1,'gilda3'),
('DenTrolls',7661,'aMolGen67','Dragonide','Guerriero',55,'gilda63'),
```

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;
```

gilda	nome	TOT_Esperienza
gilda99	ArcticDraugr	20153700
gilda99	ArcticGhouls	9048906
gilda99	AshDemons	13050293
gilda99	BlueWendigos	7723711
gilda99	BrassGorgons	25532658
gilda99	BurntTroglodytes	15211988
gilda99	CavalierChangelings	50468834
gilda99	CloakedLeapers	38655092
gilda99	CrimsonLeapers	6476407
gilda99	DiamondWendigos	5537475
gilda99	DuskGnomes	41157291
gilda99	DwarfDevils	16623980
gilda99	EternalGremlins	22164401
gilda99	FireHalflings	18557372
gilda99	GloriousSpirits	10035376
gilda99	GloriousTerrors	3155432
gilda99	GlowingWisps	6487804
gilda99	GraciousSouleaters	10120994
gilda99	GreenCorruptions	5008221
gilda99	GreenPisces	3497185
gilda99	GreyHellhounds	56392721
gilda99	HellHydras	30970636
gilda99	HighlandBunyips	13427083

2. **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 ISNU
LL(personaggi.gilda)
GROUP BY personaggi.gilda
ORDER BY Tot_Esperienza DESC;
```

gilda	TOT_Esperienza
gilda99	475488618
gilda73	90417539
gilda83	86550432
gilda35	76193555
gilda11	66378723
gilda63	65245808
gilda32	61835357
gilda50	59820440
gilda34	59299929
gilda94	59220792
gilda39	58466966
gilda22	56926146
gilda38	55649752
gilda84	54142248
gilda81	51560269
gilda44	47235610
gilda6	45542798
gilda49	42432380
gilda48	42329465
gilda74	41771980
gilda80	40798279

3. 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;
```

Personaggio	oggetto	descrizione	bersaglio	categoria	utilizzabile	vendibile	valore
AlmondGoblins	Mask of Darknes	descrizione oggetto: Mask of Darkness	sè	varie	0	1	450.05
AlmondGoblins	Prime Sword	descrizione oggetto: Prime Sword	alleato gruppo(5m)	pozione	1	1	219.71
AlmondGoblins	Slumber Pillar	descrizione oggetto: Slumber Pillar	nemico gruppo(5m)	abito	1	1	12.87
AlmondGoblins	Staff of Paraly	descrizione oggetto: Staff of Paralysis	nemico gruppo(10m)	missione	0	0	0
AlmondGoblins	Triumph Disc	descrizione oggetto: Triumph Disc	alleato gruppo(10m)	varie	0	1	40.2
AlmondGoblins	Twilight Elixir	descrizione oggetto: Twilight Elixir	alleato gruppo(10m)	arma	1	1	956.42
AlpineAncients	All-Seeing Lett	descrizione oggetto: All-Seeing Letters	nemico gruppo(10m)	pozione	1	1	434.66
AlpineAncients	Athanasia Arch	descrizione oggetto: Athanasia Arch	sè	varie	0	1	820.62
AlpineAncients	Banishing Fount	descrizione oggetto: Banishing Fountain	sè	pozione	1	1	285.71
AlpineAncients	Burning Key	descrizione oggetto: Burning Key	nemico singolo	abito	1	1	486.3

4. **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 personagg
i ON personaggi.nome = inventario.personaggio
GROUP BY personaggi.nome
HAVING TOT_Oggetti > 30;
```

Personaggio	TOT_Oggetti
AmazonNewts	44
AmphibianGenies	31
AwokenSouleaters	46
BeardedBeasts	37
BlazingWyrms	43
BlondeHobgoblins	43
BloodGhouls	32
BogBunyips	45
BogSuccubi	39
BrassGorgons	48
BrightAngels	36
BurntFauns	39
CavernSouleaters	43
ChaosNewts	31
CloakedLeapers	43
CrestedLynxes	32
CrimsonChimeras	39
CrimsonMonsters	34
CrossedUnicorns	44
DireUnicorns	32

5. **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”;
```

mittente	Corpo	nome
ytDejCla72	Illusione disperata sorrideva ch riconobbe su. Dov	nome14

6. **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.miss
ione = missioni.ID
WHERE assegnare.stato = "Completata" AND personaggi.classe = "Guerriero"
ORDER BY personaggi.nome LIMIT 10;
```

nome	classe	oggetto
BeardedBeasts	Guerriero	Crown of Vengea
BeardedBeasts	Guerriero	Exiled Gauntlet
BeardedBeasts	Guerriero	Hand of Desire
BeardedBeasts	Guerriero	Hero Mask
BeardedBeasts	Guerriero	Instrument of M
BeardedBeasts	Guerriero	Rebirth Lamp
BeardedBeasts	Guerriero	Slumber Pillar
BeardedBeasts	Guerriero	Sword of Spells
BeardedBeasts	Guerriero	Arch of Fate
BeardedBeasts	Guerriero	Benediction Tom

7. **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 personagg
i ON personaggi.nome = inventario.personaggio
GROUP BY personaggi.nome
ORDER BY TOT_Oggetti DESC
LIMIT 1;
```

proprietario	TOT_Oggetti
SpringElementals	50

8. 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;
```

nome	latitudine	longitudine	prezzo
Zoetermegen	32.8179	45.7572	4.68338
Loughblayney	87.007	-19.372	13.3692
Gailhöring	-61.6598	18.9156	18.6401
Cotafe	-29.5971	69.6576	35.3206
Hallhausen	-45.2037	-81.1356	58.4352
Casafati	-3.01709	46.1273	70.5354
Mödtal	52.1966	72.9522	84.8572

9. 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;
```

missione	Livello	N_personaggi
400	1	5
400	125	2
400	TOTALE ->	7

10. 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 = og
getti.nome
WHERE influenzare.statistica IN ("salute_attuale", "salute_massima")
GROUP BY missioni.ID LIMIT 10;
```

ID	monete	esperienza
301	4682	2632089
304	310	3155432
305	2281	6476407
306	7705	21874694
308	259	3898028
309	8660	2424835
310	6751	6573886
311	9675	16325217
312	3642	22675539
313	7528	10749374

11. 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 m
issioni ON missioni.ID = assegnare.missione
WHERE missioni.grado >= 125 LIMIT 10;
```

nome	livello
BlueMummies	1
BronzeBiters	42
DarkLeapers	79
EasternSprites	61
EmeraldSpriggans	76
FireHalflings	42
FlameNight Stalkers	39
GloriousTerrors	1
GreyPhoenix	1
HighlandBunyips	59

12. 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";
```

razza	nome	descrizione	bersaglio	categoria	utilizzabile	vendibile	valore
Dragonide	Band of Contagi	descrizione oggetto: Band of Contagion	alleato singolo	armatura	1	1	486.67
Dragonide	Cup of Paralysi	descrizione oggetto: Cup of Paralysis	nemico gruppo(5m)	missione	0	0	0
Dragonide	Damnation Feath	descrizione oggetto: Damnation Feather	alleato gruppo(5m)	pergamena	1	1	161.61
Dragonide	Feather of Serv	descrizione oggetto: Feather of Service	nemico singolo	abito	1	1	920.25
Dragonide	Hallowed Statue	descrizione oggetto: Hallowed Statue	nemico gruppo(5m)	armatura	1	1	938.19
Dragonide	Hand of Spellbi	descrizione oggetto: Hand of Spellbinding	nemico singolo	armatura	1	1	664.2
Dragonide	Immortal Crown	descrizione oggetto: Immortal Crown	sè	pozione	1	1	563.61
Dragonide	Key of Annihila	descrizione oggetto: Key of Annihilation	alleato gruppo(5m)	missione	0	0	0
Dragonide	Mending Seal	descrizione oggetto: Mending Seal	alleato gruppo(5m)	varie	0	1	615.03
Dragonide	Necrotic Ark	descrizione oggetto: Necrotic Ark	alleato gruppo(10m)	pozione	1	1	271.98
Dragonide	Pillar of Fire	descrizione oggetto: Pillar of Fire	alleato gruppo(5m)	abito	1	1	297.06
Dragonide	Prosperous Chal	descrizione oggetto: Prosperous Chalice	nemico gruppo(5m)	arma	1	1	932.19
Dragonide	Sword of Spells	descrizione oggetto: Sword of Spells	alleato gruppo(5m)	varie	0	1	763.71
Dragonide	Symbols of Agon	descrizione oggetto: Symbols of Agony	alleato singolo	armatura	1	1	443.14
Dragonide	Truth Circlet	descrizione oggetto: Truth Circlet	alleato singolo	arma	1	1	565.52

13. 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.cl
asse, 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;
```

utente	orario	nome	razza	classe	livello	gilda
arBakPil5	2019-03-06 04:13:01	PurpleBiters	Tiefling	Monaco	1	gilda4
arBakPil5	2019-03-06 04:13:01	TimberwolfAtranochs	Tiefling	Monaco	1	gilda33
ueRayMig90	2019-03-29 02:40:54	MireSouls	Halfling	Chierico	134	gilda71
ueRayMig90	2019-03-29 02:40:54	RedBerserkers	Halfling	Stregone	54	gilda45
ueRayMig90	2019-03-29 02:40:54	TigerKraken	Dragonide	Druido	50	gilda34
swBerEll42	2019-03-30 05:45:06	HiddenBasilisks	Elfo	Mago	77	gilda12
swBerEll42	2019-03-30 05:45:06	SandAnomalies	Elfo	Paladino	1	gilda5
swBerEll42	2019-03-30 05:45:06	UndergroundJackalope	Tiefling	Warlock	67	gilda52
dlJohBra14	2019-04-05 06:06:53	BogSuccubi	Mezzelfo	Monaco	1	gilda6
dlJohBra14	2019-04-05 06:06:53	GreyPhoenix	Mezzelfo	Monaco	1	gilda80

14. 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 r
azze.abilita = abilita.nome) JOIN utilizzare_razza ON utilizzare_razza.razza = razze.nome
WHERE utilizzare_razza.oggetto = "Sanctifying Sta";
```

nome	abilita	statistica	durata	valore
Halfling	Piercing Rip	saggezza	1745	670
Umano	Exorcism	tempra	9944	100

15. 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;
```

statistica	nome	descrizione	bersaglio	categoria	utilizzabile	vendibile	valore
carisma	All-Seeing Lett	descrizione oggetto: All-Seeing Letters	nemico gruppo(10m)	pozione	1	1	434.66
carisma	Banishing Fount	descrizione oggetto: Banishing Fountain	sè	pozione	1	1	285.71
carisma	Box of Light	descrizione oggetto: Box of Light	alleato singolo	pozione	1	1	146.26
carisma	Chest of Dement	descrizione oggetto: Chest of Dementia	nemico gruppo(5m)	armatura	1	1	631.07
carisma	Darkness Cup	descrizione oggetto: Darkness Cup	alleato singolo	abito	1	1	123.13
carisma	Disc of Petrifi	descrizione oggetto: Disc of Petrification	alleato gruppo(5m)	armatura	1	1	410.08
carisma	Elixir of Servi	descrizione oggetto: Elixir of Service	nemico gruppo(5m)	pergamena	1	1	495.38
carisma	Evil Circlet	descrizione oggetto: Evil Circlet	alleato singolo	abito	1	1	386.9
carisma	Evil Texts	descrizione oggetto: Evil Texts	nemico gruppo(10m)	genere alimentare	1	1	361.05
carisma	Feather of Temp	descrizione oggetto: Feather of Temptation	alleato singolo	abito	1	1	866.16

16. 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 attivit
à_commerciali ON attività_commerciali.luogo = luoghi.nome LIMIT 10;
```

nome	latitudine	longitudine	ID_missione	ID_attiv
Affolgarten	-20.7967	-79.9484	NULL	NULL
Agrilio	-43.0912	68.6222	327	NULL
Albander	42.3976	-15.9967	NULL	NULL
Albugueiras	-38.7621	-87.8165	378	NULL
Alenfort	-75.1278	0.396932	365	578
Aligo	-49.5147	-13.5372	332	NULL
Alirife	-78.2858	-90.6934	NULL	NULL
Alirrasa	34.2125	66.3958	NULL	NULL
Alkborg	-52.9276	-26.5866	393	NULL
Alshausen	-1.22049	36.8021	NULL	505

17. 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 LI
KE "i%" OR utenti.username LIKE "o%" OR utenti.username LIKE "u%"
GROUP BY utenti.username WITH ROLLUP;
```

username	TOT_Messaggi
aColIrm54	1
aMckDan98	1
aMolGen67	1
anHatSus80	1
arBakPil5	1
arWalEdw91	1
eEvaDav63	1
elLowAur64	1
elTruRos76	1
epBerJos52	1
epParJos2	1
erBoyRob17	1
erMooVal95	1
erWinHub57	1
esKraJam55	1
esMasTer32	1
iaColMar22	1
iaMayKen11	1
iChaLor10	1
iCraJod38	1
idBacDav78	1
ieRepMar23	1
isBroLou93	1
isGloMel66	1
isMeiChr6	1
itTucJud34	1
oDavYok69	1
onMarJas40	1
orHolDol59	1
orLopDeb62	1
ueRayMig90	1
umLauTat31	1
TOT	32

18. 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;
```

gilde	membri	Totale_Gilde	Media_Gilde
gilda0	2	7	3.5000
gilda1	2	5	2.5000
gilda11	5	18	3.6000
gilda12	3	2	1.0000
gilda14	2	4	2.0000
gilda15	2	6	3.0000
gilda16	2	7	3.5000
gilda17	3	6	3.0000
gilda18	5	4	1.3333
gilda19	3	9	3.0000

19. 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;
```

nome	Tipi_Oggetti	Totale_Oggetti
LurkingHumans	46	2750
CloakedLeapers	43	2477
GroaningSphinxes	44	2420
BlazingWyrms	43	2295
BogBunyips	45	2166
LustingLeviathans	42	2148
DuskSuccubi	42	2122
GoldSouls	43	2112
FrigidSpriggans	38	2055
LegendaryJackalopes	46	2045

20. 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;
```

Nome	Completate	Attive	Differenza
CrimsonChimeras	0	54896185	-54896185
ForestValkyries	5583362	53853259	-48269897
GlassUndeads	15776074	61436998	-45660924
VioletCentaurs	0	45319569	-45319569
BlushingYetis	6453945	51406573	-44952628
WhistlingIncubi	0	40800803	-40800803
SupremeReapers	304054	39961655	-39657601
VioletGremlins	0	38292665	-38292665
SandCrawlers	723704	37080642	-36356938
PreyingCockatrice	0	36339170	-36339170

10 rows in set (0.006 sec)

21. STAMPA LE MISSIONI COMPLETATE E ATTIVE DEI MEMBRI DELLA GILD A DEL PERSONAGGIO "FrigidYetis" CHE SIANO STATE COMPLETATE O AT TIVATE 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";
```

gilda	nome	missione
gilda94	GrizzlyNight Stalker	357

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_commerci
ali ON attività_commerciali.luogo = luoghi.nome;
```

ALGEBRA RELAZIONALE:

$\prod$ 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 ra
zze ON razze.abilita = abilita.nome) JOIN utilizzare_razza ON utilizzare_razza.raz
za = 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

```
1   DROP USER IF EXISTS 'dilullo'@'localhost';
2   CREATE USER 'dilullo'@'localhost' IDENTIFIED WITH mysql_native_password;
3   SET old_passwords = 0;
4   ALTER USER 'dilullo'@'localhost' IDENTIFIED BY 'password';
5   ALTER USER 'dilullo'@'localhost' WITH MAX_USER_CONNECTIONS 5;
6   GRANT ALL PRIVILEGES ON Elysium.* TO 'dilullo'@'localhost';
7   FLUSH PRIVILEGES;
```

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.

```sql
DROP USER IF EXISTS 'player'@'127.0.0.1';
CREATE USER 'player'@'localhost' IDENTIFIED WITH mysql_native_password;
SET old_passwords = 0;
ALTER USER 'player'@'localhost' IDENTIFIED BY 'player_password';
GRANT SELECT ON Elysium.* TO 'player'@'127.0.0.1';
REVOKE ALL PRIVILEGES ON Elysium.Statistiche FROM 'player'@'127.0.0.1';
REVOKE ALL PRIVILEGES ON Elysium.Accessi FROM 'player'@'127.0.0.1';
REVOKE ALL PRIVILEGES ON Elysium.Messaggi FROM 'player'@'127.0.0.1';
REVOKE ALL PRIVILEGES ON Elysium.Utenti FROM 'player'@'127.0.0.1';
GRANT INSERT ON Elysium.gilde TO 'player'@'127.0.0.1';
GRANT INSERT, DELETE ON Elysium.equipaggiare TO 'player'@'127.0.0.1';
GRANT INSERT ON Elysium.messaggi TO 'player'@'127.0.0.1';
FLUSH PRIVILEGES;
```

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.

```sql
/*PROCEDURA PER INSERIMENTO DI RECORD NELLA TABELLA CHAT*/

DROP PROCEDURE IF EXISTS inserisci_chat;
DELIMITER //
CREATE PROCEDURE inserisci_chat(IN loops int)
BEGIN
    DECLARE v1 int;

    SET @v1 = 0;

    WHILE @v1 < loops DO
        INSERT IGNORE INTO chat(nome, data_creazione) VALUES (CONCAT("nome_chat",@v1), NOW());
        SET @v1 = @v1 + 1;

    END WHILE;
END; //
DELIMITER ;
```

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".

```
25  DROP PROCEDURE IF EXISTS inserisci_amici;
26  DELIMITER //
27  CREATE PROCEDURE inserisci_amici (IN loops int)
28  BEGIN
29      DECLARE utente1 VARCHAR(20);
30      DECLARE utente2 varchar(20);
31      DECLARE v1 int;
32      DECLARE stato tinyint(1);
33
34      SET @v1 = 0;
35
36      WHILE @v1 < loops DO
37
38          SET @utente1 = (SELECT username FROM Utenti ORDER BY RAND() LIMIT 1);
39          SET @utente2 = (SELECT username FROM Utenti ORDER BY RAND() LIMIT 1);
40
41          IF @stato = 1 THEN
42              SET @stato = 0;
43          ELSE
44              SET @stato = 1;
45          END IF;
46
47          IF @utente1 <> @utente2 THEN
48              INSERT IGNORE INTO Amici VALUES (@utente1, @utente2, @stato);
49              INSERT IGNORE INTO Amici VALUES (@utente2, @utente1, @stato);
50          END IF;
51
52          SET @v1 = @v1 + 1;
53      END WHILE;
54
55  END //
56  DELIMITER ;
```

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

```
5   DELIMITER //
6
7   DROP TRIGGER IF EXISTS blocca_delete_livello//
8   CREATE DEFINER=`dilullo`@`localhost` TRIGGER blocca_delete_livello
9       BEFORE DELETE ON livelli
10      FOR EACH ROW
11  BEGIN
12      SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Impossibile Eliminare Livelli';
13  END//
14
15  DROP TRIGGER IF EXISTS blocca_delete_abilita//
16  CREATE DEFINER=`dilullo`@`localhost` TRIGGER blocca_delete_abilita
17      BEFORE DELETE ON ABILITA
18      FOR EACH ROW
19  BEGIN
20      SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Impossibile Eliminare abilita';
21  END //
22
23  DROP TRIGGER IF EXISTS blocca_delete_missioni//
24  CREATE DEFINER=`dilullo`@`localhost` TRIGGER blocca_delete_missioni
25      BEFORE DELETE ON ABILITA
26      FOR EACH ROW
27  BEGIN
28      SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Impossibile Eliminare missioni';
29  END //
30
31  DELIMITER ;
```

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

```
DELIMITER //
DROP TRIGGER IF EXISTS check_equipaggiare//
CREATE DEFINER=`dilullo`@`localhost` TRIGGER check_equipaggiare
    BEFORE INSERT ON equipaggiare
    FOR EACH ROW
BEGIN

    DECLARE x int;

    SET @x = (SELECT COUNT(*) FROM equipaggiare WHERE personaggio = NEW.personaggio GROUP BY personaggio);

    IF @x >= 20 THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = "Impossibile aggiungere una nuova abilità per
                                                    il personaggio selezionato! out of limit";
    END IF;

END//

DELIMITER ;
```

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.

```
75  DELIMITER //
76  DROP TRIGGER IF EXISTS controlla_appartenenza//
77
78
79  CREATE DEFINER=`dilullo`@`localhost` TRIGGER controlla_appartenenza
80      BEFORE UPDATE ON personaggi
81      FOR EACH ROW
82  BEGIN
83      DECLARE x int;
84      IF NEW.gilda <> OLD.gilda THEN
85          SET @x = (SELECT COUNT(*) FROM gilde WHERE capo = OLD.nome);
86
87          IF @x > 0 THEN
88              SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = "Impossibile aggiornare gilda!
89                                                          Personaggio già a capo di un'altra gilda";
90          END IF;
91      END IF;
92  END//
93
94  DELIMITER ;
```

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

```
106  DELIMITER //
107  DROP TRIGGER IF EXISTS controlla_destinatario//
108
109  CREATE DEFINER=`dilullo`@`localhost` TRIGGER controlla_destinatario
110      BEFORE INSERT ON messaggi
111      FOR EACH ROW
112  BEGIN
113
114      DECLARE condizione int;
115      DECLARE destinatario int;
116
117      SET @destinatario = NEW.chat;
118      SET @condizione = (SELECT COUNT(*) FROM partecipare WHERE username = NEW.mittente AND id_chat = NEW.chat );
119
120      IF @condizione = 0 THEN
121          SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = "Impossibile inviare messaggio! Il mittente non fa parte della chat";
122      END IF;
123  END//
124
125  DELIMITER ;
```

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.

```
143  DELIMITER //
144  DROP TRIGGER IF EXISTS controlla_livello_insert//
145
146  CREATE DEFINER=`dilullo`@`localhost` TRIGGER controlla_livello_insert
147      AFTER INSERT ON assegnare
148      FOR EACH ROW
149  BEGIN
150      DECLARE x int;
151      DECLARE liv int;
152
153      IF NEW.stato = "Completata" THEN
154          SET @x = (SELECT COALESCE(SUM(missioni.esperienza),0)
155                    FROM missioni, assegnare
156                    WHERE assegnare.personaggio = NEW.personaggio AND
157                              assegnare.missione = missioni.ID
158                              AND assegnare.stato = 'Completata');
159          SET @liv = (SELECT COUNT(*) FROM livelli WHERE esperienza <= @x);
160
161          IF @liv > 0 THEN
162              UPDATE personaggi SET livello = @liv WHERE personaggi.nome = NEW.personaggio;
163          END IF;
164      END IF;
165  END//
```

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.

```
205  DELIMITER //
206  DROP TRIGGER IF EXISTS controlla_abilitaInnata//
207
208  CREATE DEFINER=`dilullo`@`localhost` TRIGGER controlla_abilitaInnata
209      BEFORE INSERT ON equipaggiare
210      FOR EACH ROW
211  BEGIN
212      DECLARE razza varchar(15);
213      DECLARE x int;
214
215      SET @razza = (SELECT razza FROM personaggi WHERE personaggi.nome = NEW.personaggio);
216      SET @x = (SELECT COUNT(*) FROM razze WHERE nome = @razza AND abilita = NEW.abilita);
217
218      IF @x > 0 THEN
219          SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = "Impossibile equipaggiare!
220                                                      E' un'abilità innata del personaggio!";
221      END IF;
222
223  END//
224  DELIMITER ;
```

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.

```
1   DELIMITER //
2   DROP PROCEDURE IF EXISTS leggi_messaggi //
3   CREATE PROCEDURE leggi_messaggi ( IN utente varchar(20), IN ID_chat int unsigned )
4
5   BEGIN
6       SELECT messaggi.*
7       FROM messaggi, partecipare
8       WHERE messaggi.chat = partecipare.id_chat AND utente = partecipare.username AND partecipare.id_chat = ID_chat;
9   END //
10
11  DELIMITER ;
```

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.

```
3   SET GLOBAL event_scheduler = ON;
4   DROP EVENT IF EXISTS rifornimento;
5   CREATE EVENT rifornimento
6   ON SCHEDULE EVERY 1 DAY STARTS '2020-06-24 00:00:00'
7   DO
8   UPDATE elysium.vendere SET disponibile = disponibile_max;
```

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

```
13  DELIMITER //
14  DROP PROCEDURE IF EXISTS acquisto //
15  CREATE PROCEDURE acquisto(IN attivita smallint(5),
16                            IN personaggio varchar(20),
17                            IN oggetto varchar(30),
18                            IN quantita smallint(5))
19  BEGIN
20      DECLARE prezzo float;
21      DECLARE condizione int;
22
23      START TRANSACTION;
24          SET @prezzo = (SELECT prezzo
25                         FROM vendere
26                         WHERE vendere.attività = attivita AND
27                               vendere.oggetto = oggetto);
28          UPDATE vendere SET disponibile = disponibile - quantita
29                         WHERE vendere.attività = attivita AND
30                               vendere.oggetto = oggetto;
31          UPDATE personaggi SET monete = monete - (@prezzo * quantità)
32                            WHERE nome = personaggio;
33
34          SET @condizione = (SELECT COUNT(*)
35                             FROM inventario
36                             WHERE inventario.personaggio = personaggio AND
37                                   inventario.oggetto = oggetto);
38          IF @condizione > 0 THEN
39              UPDATE inventario SET inventario.quantità = inventario.quantità + quantita
40                                WHERE inventario.personaggio = personaggio AND
41                                      inventario.oggetto = oggetto;
42          ELSE
43              INSERT INTO inventario VALUES (personaggio, oggetto, quantita);
44          END IF;
45      COMMIT;
46  END //
47  DELIMITER ;
```

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:

```
{
    "_id":  "nome personaggio",
    "utente": "username utente",
    "monete": "float(n_monete)",
    "razza": "nome razza",
    "classe": "nome classe",
    "gilda": "nome gilda",
    "livello": "int(livello)",
    "statistiche": {
        "nome statistica1": "int(valore)",
        "nome statistica2": "int(valore)"
    },
    "inventario":[
        {
            "nome": "nome oggetto1",
            "quantita": "int(quantita)"
        },
        {
            "nome": "nome oggetto2",
            "quantita": "int(quantita)"
        }
    ],
    "missioni":[
        {
            "id": "id missione1",
            "stato": "stato missione1"
        },
        {
            "id": "id missione2",
            "stato": "stato missione2"
        }
    ],
    "abilita": ["nome abilita1", "nome abilita2", "nome abilita3"]
}
```

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

```
{
    "nome": "nome oggetto",
    "descrizione":"descrizione oggetto",
    "bersaglio" : "bersaglio oggetto",
    "categoria" : "nome categoria",
    "utilizzabile" : int(0/1),
    "vendibile" : int(0/1),
    "valore" : float(valore),
    "statistiche": [
        {
            "nome statistica": int(valore),
            "durata": int(durata)
        },
        {
            "nome statistica": int(valore),
            "durata": int(durata)
        },
        ...
    ]
}
```

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:

```
> use elysium2
switched to db elysium2
> db.personaggi.count()
1000000
> db.oggetti.count()
28070
>
```

Adesso proviamo ad eseguire interrogazioni più elaborate:

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

```
> db.personaggi.find( { $where: function(){
...     check = true;
...     for(i = 0; i < this.missioni.length; i++){
...         if(this.missioni[i].stato == "Completata")
...         {
...             check = false;
...             break;
...         }
...     }
...     return check;
... } } , {inventario:0, abilita:0, statistiche:0}).pretty()
```

```
{
        "_id" : "BrilliantNight Stalkers265",
        "utente" : "toillE eeneR",
        "monete" : 1687,
        "razza" : "Mezzelfo",
        "classi" : "o",
        "gilda" : "gilda_47",
        "livello" : 205,
        "missioni" : [ ]
}
{
        "_id" : "JadeKraken289",
        "utente" : "letraM leinaD",
        "monete" : 1034,
        "razza" : "Elfo",
        "classi" : "o",
        "gilda" : "gilda_43",
        "livello" : 117,
        "missioni" : [
                {
                        "id" : "missione_50",
                        "stato" : "Attiva"
                }
        ]
}
{
        "_id" : "SuccubiGracious384",
        "utente" : "lleB acisseJ",
        "monete" : 2509,
        "razza" : "Dragonide",
        "classi" : "o",
        "gilda" : "gilda_39",
        "livello" : 111,
        "missioni" : [
                {
                        "id" : "missione_40",
                        "stato" : "Attiva"
                },
                {
                        "id" : "missione_79",
                        "stato" : "Attiva"
                },
                {
                        "id" : "missione_75",
                        "stato" : "Attiva"
                }
        ]
}
```


Verifichiamone la correttezza cercando, ad esempio, il personaggio "SuccubiGracious384":

```
> db.personaggi.find({"_id":"SuccubiGracious384"},{"_id":1, "utente":1,"missioni":1}).pretty()
{
        "_id" : "SuccubiGracious384",
        "utente" : "lleB acisseJ",
        "missioni" : [
                {
                        "id" : "missione_40",
                        "stato" : "Attiva"
                },
                {
                        "id" : "missione_79",
                        "stato" : "Attiva"
                },
                {
                        "id" : "missione_75",
                        "stato" : "Attiva"
                }
        ]
}
```

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

```
> db.personaggi.aggregate([
...     {$unwind: "$missioni"},
...     {
...         $group: {
...             _id:"$_id",
...             N_Completate: {$sum : {$cond: [ { $eq: ["$missioni.stato","Completata"] } , 1, 0]  }},
...             N_Attive: {$sum : {$cond: [ { $eq: ["$missioni.stato","Attiva"] } , 1, 0]  }},
...         }
...     },
...     {
...         $addFields: {
...             Differenza: {$subtract: ["$N_Completate", "$N_Attive"] }
...         }
...     }
... ],
... { allowDiskUse: true}
... )
{ "_id" : "AldabraAliens130276", "N_Completate" : 16, "N_Attive" : 19, "Differenza" : -3 }
{ "_id" : "AldabraAliens140479", "N_Completate" : 26, "N_Attive" : 29, "Differenza" : -3 }
{ "_id" : "AldabraAliens31413", "N_Completate" : 29, "N_Attive" : 23, "Differenza" : 6 }
{ "_id" : "AldabraAliens386933", "N_Completate" : 17, "N_Attive" : 11, "Differenza" : 6 }
{ "_id" : "AldabraAliens413565", "N_Completate" : 14, "N_Attive" : 28, "Differenza" : -14 }
{ "_id" : "AldabraAliens490719", "N_Completate" : 27, "N_Attive" : 25, "Differenza" : 2 }
{ "_id" : "AldabraAliens542419", "N_Completate" : 26, "N_Attive" : 44, "Differenza" : -18 }
{ "_id" : "AldabraAliens557476", "N_Completate" : 10, "N_Attive" : 13, "Differenza" : -3 }
{ "_id" : "AldabraAliens594784", "N_Completate" : 31, "N_Attive" : 22, "Differenza" : 9 }
{ "_id" : "AldabraAliens654758", "N_Completate" : 26, "N_Attive" : 30, "Differenza" : -4 }
{ "_id" : "AldabraAliens770699", "N_Completate" : 45, "N_Attive" : 44, "Differenza" : 1 }
{ "_id" : "AldabraAliens836488", "N_Completate" : 35, "N_Attive" : 42, "Differenza" : -7 }
{ "_id" : "AldabraAliens837203", "N_Completate" : 24, "N_Attive" : 28, "Differenza" : -4 }
{ "_id" : "AldabraAliens899822", "N_Completate" : 16, "N_Attive" : 27, "Differenza" : -11 }
{ "_id" : "AldabraAncients127260", "N_Completate" : 8, "N_Attive" : 8, "Differenza" : 0 }
{ "_id" : "AldabraAncients138571", "N_Completate" : 28, "N_Attive" : 38, "Differenza" : -10 }
{ "_id" : "AldabraAncients271911", "N_Completate" : 26, "N_Attive" : 30, "Differenza" : -4 }
{ "_id" : "AldabraAncients531933", "N_Completate" : 27, "N_Attive" : 40, "Differenza" : -13 }
{ "_id" : "AldabraAncients66305", "N_Completate" : 16, "N_Attive" : 11, "Differenza" : 5 }
{ "_id" : "AldabraAncients716282", "N_Completate" : 39, "N_Attive" : 34, "Differenza" : 5 }
Type "it" for more
```

Verifichiamo la correttezza cercando le missioni di “AldabraAncients127260”:

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


```
"_id" : "AldabraAncients127260",
"missioni" : [
        {
                "id" : "missione_15",
                "stato" : "Attiva"
        },
        {
                "id" : "missione_39",
                "stato" : "Completata"
        },
        {
                "id" : "missione_80",
                "stato" : "Completata"
        },
        {
                "id" : "missione_54",
                "stato" : "Attiva"
        },
        {
                "id" : "missione_78",
                "stato" : "Completata"
        },
        {
                "id" : "missione_5",
                "stato" : "Completata"
        },
        {
                "id" : "missione_56",
                "stato" : "Completata"
        },
        {
                "id" : "missione_33",
                "stato" : "Completata"
        },
                {
                        "id" : "missione_29",
                        "stato" : "Attiva"
                },
                {
                        "id" : "missione_68",
                        "stato" : "Completata"
                },
                {
                        "id" : "missione_29",
                        "stato" : "Attiva"
                },
                {
                        "id" : "missione_92",
                        "stato" : "Attiva"
                },
                {
                        "id" : "missione_42",
                        "stato" : "Attiva"
                },
                {
                        "id" : "missione_18",
                        "stato" : "Completata"
                },
                {
                        "id" : "missione_77",
                        "stato" : "Attiva"
                },
                {
                        "id" : "missione_15",
                        "stato" : "Attiva"
                }
        ]
```

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

```
> db.personaggi.aggregate([
...     {$group:
...         {
...             "_id": null,
...             totale_personaggi: {$sum:1},
...             totale_inventario:{ $sum: {$sum: "$inventario.quantita"}},
...             media:{ $avg : {$sum: "$inventario.quantita"}}
...         }
...     }
... ])
{ "_id" : null, "totale_personaggi" : 1000000, "totale_inventario" : NumberLong("2499869281"), "media" : 2499.869281 }
```

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

```
> db.personaggi.aggregate([
...     {$unwind: "$missioni"},
...     {
...         $group: {
...             _id:"$gilda",
...             N_Completate: {$sum : {$cond: [ { $eq: ["$missioni.stato","Completata"] } , 1, 0]  }},
...             Totale: {$sum : 1}
...         }
...     },
...     {
...         $addFields: {
...             Percentuale: {
...                 $concat: [
...                     { $convert:{
...                         input: { $round: [ {$divide: [ {$multiply: ["$N_Completate", 100] }, "$Totale"]}, 2] },
...                         to: "string"
...                     }},
...                     "%"
...                 ]
...             }
...         }
...     }
... ],
... { allowDiskUse: true}
... )
{ "_id" : "gilda_37", "N_Completate" : 243458, "Totale" : 487109, "Percentuale" : "49.98%" }
{ "_id" : "gilda_19", "N_Completate" : 245106, "Totale" : 489304, "Percentuale" : "50.09%" }
{ "_id" : "gilda_75", "N_Completate" : 246122, "Totale" : 491586, "Percentuale" : "50.07%" }
{ "_id" : "gilda_89", "N_Completate" : 240724, "Totale" : 482383, "Percentuale" : "49.9%" }
{ "_id" : "gilda_22", "N_Completate" : 247402, "Totale" : 494858, "Percentuale" : "49.99%" }
{ "_id" : "gilda_14", "N_Completate" : 244530, "Totale" : 489634, "Percentuale" : "49.94%" }
{ "_id" : "gilda_71", "N_Completate" : 245475, "Totale" : 492389, "Percentuale" : "49.85%" }
{ "_id" : "gilda_33", "N_Completate" : 245524, "Totale" : 489944, "Percentuale" : "50.11%" }
{ "_id" : "gilda_35", "N_Completate" : 248043, "Totale" : 496758, "Percentuale" : "49.93%" }
{ "_id" : "gilda_81", "N_Completate" : 241221, "Totale" : 483561, "Percentuale" : "49.88%" }
{ "_id" : "gilda_21", "N_Completate" : 245921, "Totale" : 491941, "Percentuale" : "49.99%" }
{ "_id" : "gilda_11", "N_Completate" : 243792, "Totale" : 487704, "Percentuale" : "49.99%" }
{ "_id" : "gilda_47", "N_Completate" : 244646, "Totale" : 490394, "Percentuale" : "49.89%" }
{ "_id" : "gilda_100", "N_Completate" : 246543, "Totale" : 493933, "Percentuale" : "49.91%" }
{ "_id" : "gilda_30", "N_Completate" : 245697, "Totale" : 492420, "Percentuale" : "49.9%" }
{ "_id" : "gilda_76", "N_Completate" : 240012, "Totale" : 479800, "Percentuale" : "50.02%" }
{ "_id" : "gilda_20", "N_Completate" : 243878, "Totale" : 487722, "Percentuale" : "50%" }
{ "_id" : "gilda_53", "N_Completate" : 245360, "Totale" : 490869, "Percentuale" : "49.98%" }
{ "_id" : "gilda_31", "N_Completate" : 244373, "Totale" : 488942, "Percentuale" : "49.98%" }
{ "_id" : "gilda_46", "N_Completate" : 242744, "Totale" : 485451, "Percentuale" : "50%" }
Type "it" for more
```

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

```
> db.personaggi.aggregate([
...     {
...         $lookup: {
...             "localField": "inventario.nome",
...             "from": "oggetti",
...             "foreignField": "nome",
...             "as": "lista_oggetti"
...         }
...     },
...     {
...         $unwind : "$lista_oggetti",
...     },
...     {
...         $match : {
...             "lista_oggetti.effetti.carisma": {$exists:true}
...         }
...     },
...     {
...         $project:{
...             "_id":"$_id",
...             "nome_oggetto":"$lista_oggetti.nome",
...             "carisma":"$lista_oggetti.effetti.carisma"
...         }
...     }
... ]
... ).pretty()
{
        "_id" : "IcyTaurens0",
        "nome_oggetto" : "Impurity Box",
        "carisma" : [
                1000
        ]
}
{
        "_id" : "IcyTaurens0",
        "nome_oggetto" : "Ichor of Lightness",
        "carisma" : [
                744
        ]
}
{
        "_id" : "IcyTaurens0",
        "nome_oggetto" : "Maniacal Ring",
        "carisma" : [
                851
        ]
}
```

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:

```
MariaDB [elysium]> SELECT  SUM(TABLE_ROWS)
    ->      FROM INFORMATION_SCHEMA.TABLES
    ->      WHERE TABLE_SCHEMA = 'elysium';
+-----------------+
| SUM(TABLE_ROWS) |
+-----------------+
|          102502 |
+-----------------+
1 row in set (0.001 sec)
```

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)

```
> execution_millis = after - before
287
```

MySQL

```
MariaDB [elysium]> SELECT personaggi.nome,
    ->        SUM(case when stato = 'Completata' then 1 else 0 end)
    ->             AS Completate
    -> FROM personaggi LEFT JOIN assegnare ON personaggi.nome = assegnare.personaggio
    -> GROUP BY personaggi.nome
    -> HAVING Completate = 0;
```

```
443 rows in set (0.004 sec)
```

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.

```
MariaDB [elysium]>  select count(*) from assegnare;
+----------+
| count(*) |
+----------+
|     4984 |
+----------+
1 row in set (0.002 sec)

MariaDB [elysium]> select count(*) from personaggi
    -> ;
+----------+
| count(*) |
+----------+
|     1479 |
+----------+
```

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):

```
> execution_millis = after - before
23938
```

MySQL

```
MariaDB [elysium]> SELECT SUM(quantità), AVG(quantità)
    -> FROM inventario;
+---------------+---------------+
| SUM(quantità) | AVG(quantità) |
+---------------+---------------+
|       2119055 |       50.1101 |
+---------------+---------------+
1 row in set (0.014 sec)
```

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.

```
> db.personaggi.aggregate([
...     {$group:
...         {
...             "_id": null,
...             media:{ $avg : {$size: "$inventario"}}
...         }
...     }
... ])
{ "_id" : null, "media" : 49.508971 }
```

(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

```
MariaDB [elysium]> select count(*) from inventario;
+----------+
| count(*) |
+----------+
|    42288 |
+----------+
```

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