Progetto DataBase

Mutua Fadhla Mohamed — SM3201434

July 14, 2024

1 Descrizione del database

Questo database è progettato per gestire un gioco di ruolo in cui i giocatori e i personaggi non giocanti (NPC) interagiscono in un mondo virtuale. I giocatori possono unirsi a gilde, completare missioni e guadagnare obiettivi che sbloccano nuove dimensioni. Le gilde sono gruppi organizzati che possono includere sia giocatori che NPC, influenzando le missioni disponibili e gli obiettivi sbloccabili. Le missioni sono iniziate dai giocatori attraverso gli NPC. Gli oggetti, che possono essere acquisiti e utilizzati dai giocatori, hanno uno stato che può influenzare le abilità del giocatore. Gli obiettivi, raggiungibili completando missioni, possono sbloccare nuove dimensioni, ognuna delle quali offre nuove sfide e missioni.

2 Query

Procedure e Trigger SQL

Trigger: lock_weapon_state

Descrizione: Questo trigger impedisce di cambiare lo stato di un oggetto del giocatore da FALSE a TRUE se è stato impostato a FALSE.

```
DELIMITER //
CREATE TRIGGER lock_weapon_state
BEFORE UPDATE ON Player_Item
FOR EACH ROW
BEGIN
    IF OLD.state = FALSE
    AND NEW.state = TRUE
    THEN
        SET NEW.state = FALSE;
    END IF;
END;
//
DELIMITER //
```

Spiegazione Riga per Riga:

- `CREATE TRIGGER lock_weapon_state`: Definisce un nuovo trigger chiamato `lock_weapon_state`.
- `BEFORE UPDATE ON Player_Item`: Specifica che questo trigger verrà eseguito prima di un aggiornamento sulla tabella `Player_Item`.
- `FOR EACH ROW`: Indica che il trigger verrà eseguito per ogni riga che viene aggiornata.
- `BEGIN`: Inizia il blocco di istruzioni SQL che compongono il trigger.
- `IF OLD.state = FALSE AND NEW.state = TRUE THEN`: Controlla se lo stato dell'oggetto è cambiato da `FALSE` a `TRUE`.
- `SET NEW.state = FALSE;`: Impedisce il cambio di stato a `TRUE` se era `FALSE`.
- `END IF`: Fine del blocco `IF`.
- `END`: Termina il trigger.

InsertPlayerItem

Descrizione: Questa procedura inserisce un oggetto per un giocatore. Se l'oggetto non esiste nella tabella Item, imposta lo stato degli oggetti del giocatore a FALSE. Se l'oggetto esiste, inserisce l'oggetto con stato TRUE.

```
DELIMITER //
CREATE PROCEDURE InsertPlayerItem(p_player_id INT,
p_item_id INT)
BEGIN
    IF NOT EXISTS (SELECT 1
    FROM Item WHERE item_id = p_item_id)
    THEN
        UPDATE Player_Item
        SET state = FALSE
        WHERE player_id = p_player_id;
    ELSE
        INSERT INTO Player_Item
        (player_id, item_id, state)
        VALUES (p_player_id, p_item_id, TRUE);
    END IF;
END;
//
DELIMITER //
```

Spiegazione Riga per Riga:

- `CREATE PROCEDURE InsertPlayerItem(p_player_id INT, p_item_id INT)`: Definisce una nuova procedura chiamata `InsertPlayerItem`.
- `IF NOT EXISTS (SELECT 1 FROM Item WHERE item_id = p_item_id) THEN`: Controlla se l'oggetto esiste nella tabella `Item`.
- `UPDATE Player_Item SET state = FALSE WHERE player_id = p_player_id`: Se l'oggetto non esiste, imposta lo stato degli oggetti del giocatore a `FALSE`.
- `ELSE`: Altrimenti (se l'oggetto esiste).
- `INSERT INTO Player_Item (player_id, item_id, state) VALUES (p_player_id, p_item_id, TRUE)`: Inserisce l'oggetto per il giocatore con stato `TRUE`.
- `END IF`: Fine del blocco `IF`.

Test_InsertPlayerItem

Descrizione: Questa procedura testa la procedura InsertPlayerItem tentando di inserire sia oggetti validi che non validi e controllando lo stato della tabella Player_Item.

```
DELIMITER //
CREATE PROCEDURE Test_InsertPlayerItem(in who_to_check int)
BEGIN
    declare temp_value int;

    SELECT 'Before␣Insert' AS status,
    pi.player_item_id, p.player_name,
    i.item_name, pi.state
    FROM Player_Item pi
    JOIN Player p ON pi.player_id = p.player_id
    JOIN Item i ON pi.item_id = i.item_id
    WHERE pi.player_id = who_to_check;

    create temporary table tmp as
    select min(item_id) as temp_value
    from player_item;
```

```
    CALL InsertPlayerItem(who_to_check, temp_value);

    SELECT 'After␣Inserting␣Valid␣Item' AS status,
    pi.player_item_id, p.player_name,
    i.item_name, pi.state
    FROM Player_Item pi
    JOIN Player p ON pi.player_id = p.player_id
    JOIN Item i ON pi.item_id = i.item_id
    WHERE pi.player_id = who_to_check;

    CALL InsertPlayerItem(who_to_check, temp_value-1);

    SELECT 'After␣Attempting␣to␣Insert␣Invalid␣Item' AS status,
    pi.player_item_id, p.player_name,
    i.item_name, pi.state
    FROM Player_Item pi
    JOIN Player p ON pi.player_id = p.player_id
    JOIN Item i ON pi.item_id = i.item_id
    WHERE pi.player_id = who_to_check;

    UPDATE Player_Item SET state = TRUE
    WHERE player_item_id = who_to_check;

    SELECT 'After␣Attempting␣to␣Change␣Back' AS status,
    pi.player_item_id, p.player_name,
    i.item_name, pi.state
    FROM Player_Item pi
    JOIN Player p ON pi.player_id = p.player_id
    JOIN Item i ON pi.item_id = i.item_id
    WHERE pi.player_id = who_to_check;

    drop temporary table tmp;
END;
//
DELIMITER //
```

Spiegazione Riga per Riga:

- `CREATE PROCEDURE Test_InsertPlayerItem()`: Definisce una nuova procedura chiamata `Test_InsertPlayerItem`.
- `SELECT 'Before Insert' AS status, pi.player_item_id, p.player_name, i.item_name, pi.state FROM Player_Item pi JOIN Player p ON pi.player_id = p.player_id JOIN Item i ON pi.item_id = i.item_id WHERE pi.player_id = 1;`: Visualizza lo stato di `Player_Item` prima di qualsiasi modifica.
- `CALL InsertPlayerItem(1, 1);`: Tenta di inserire un oggetto valido per il giocatore 1.
- `SELECT 'After Inserting Valid Item' AS status, pi.player_item_id, p.player_name, i.item_name, pi.state FROM Player_Item pi JOIN Player p ON pi.player_id = p.player_id JOIN Item i ON pi.item_id = i.item_id WHERE pi.player_id = 1;`: Visualizza lo stato di `Player_Item` dopo aver inserito un oggetto valido.
- `CALL InsertPlayerItem(1, 999);`: Tenta di inserire un oggetto non valido per il giocatore 1.
- `SELECT 'After Attempting to Insert Invalid Item' AS status, pi.player_item_id, p.player_name, i.item_name, pi.state FROM Player_Item pi JOIN Player p ON pi.player_id = p.player_id JOIN Item i ON pi.item_id = i.item_id WHERE pi.player_id = 1;`: Visualizza lo stato di `Player_Item` dopo aver tentato di inserire un oggetto non valido.
- `UPDATE Player_Item SET state = TRUE WHERE player_item_id = 1;`: Tenta di cambiare lo stato di `Player_Item` a `TRUE` (non dovrebbe essere possibile a causa del trigger).
- `SELECT 'After Attempting to Change Back' AS status, pi.player_item_id, p.player_name, i.item_name, pi.state FROM Player_Item pi JOIN Player p ON pi.player_id = p.player_id JOIN Item i ON pi.item_id = i.item_id WHERE pi.player_id = 1;`: Verifica se lo stato di `Player_Item` rimane `FALSE`.

CheckDimensionUnlock

Descrizione: Questa procedura verifica se un giocatore può sbloccare la dimensione successiva in base al completamento delle missioni rilevanti.

```
DELIMITER //
CREATE PROCEDURE CheckDimensionUnlock(IN p_player_id INT)
BEGIN
    DECLARE current_dimension_id INT;
    DECLARE next_dimension_id INT;
    DECLARE relevant_quests_completed BOOLEAN;

    SELECT dimension_id INTO current_dimension_id
    FROM Travel WHERE player_id = p_player_id;
    SET next_dimension_id = current_dimension_id + 1;

    SELECT COUNT(*) = 0 INTO relevant_quests_completed
    FROM Quest q JOIN Check_Achievement ca
    ON q.quest_id = ca.quest_id
    WHERE ca.requires_all_player_items = TRUE
    AND q.quest_id NOT IN (SELECT quest_id
    FROM Complete WHERE player_id = p_player_id);

    IF relevant_quests_completed THEN
        SET @unlock_allowed = TRUE;
    ELSE
        SET @unlock_allowed = FALSE;
    END IF;
END;
//
DELIMITER //
```

Spiegazione Riga per Riga:

- `CREATE PROCEDURE CheckDimensionUnlock(IN p_player_id INT)`: Definisce una nuova procedura chiamata `CheckDimensionUnlock`.
- `DECLARE current_dimension_id INT;`: Dichiara una variabile `current_dimension_id` di tipo `INT`.
- `DECLARE next_dimension_id INT;`: Dichiara una variabile `next_dimension_id` di tipo `INT`.
- `DECLARE relevant_quests_completed BOOLEAN;`: Dichiara una variabile `relevant_quests_completed` di tipo `BOOLEAN`.
- `SELECT dimension_id INTO current_dimension_id FROM Travel WHERE player_id = p_player_id;`: Ottiene l'ID della dimensione corrente del giocatore.
- `SET next_dimension_id = current_dimension_id + 1;`: Imposta l'ID della dimensione successiva.
- `SELECT COUNT(*) = 0 INTO relevant_quests_completed FROM Quest q JOIN Check_Achievement ca ON q.quest_id = ca.quest_id WHERE ca.requires_all_player_items = TRUE AND q.quest_id NOT IN (SELECT quest_id FROM Complete WHERE player_id = p_player_id);`: Verifica se tutte le missioni rilevanti sono completate per sbloccare la dimensione.
- `IF relevant_quests_completed THEN SET @unlock_allowed = TRUE; ELSE SET @unlock_allowed = FALSE; END IF;`: Imposta il flag di sblocco in base al completamento delle missioni.

ChangeDimension

Descrizione: Questa procedura permette a un giocatore di passare alla dimensione successiva se è la dimensione successiva in sequenza.

```
DELIMITER //
CREATE PROCEDURE ChangeDimension(IN p_player_id INT,
IN new_dimension_id INT)
BEGIN
```

```
    DECLARE current_dimension_id INT;

    SELECT dimension_id INTO current_dimension_id
    FROM Travel WHERE player_id = p_player_id;

    IF new_dimension_id = current_dimension_id + 1
    THEN
        UPDATE Travel SET dimension_id = new_dimension_id
        WHERE player_id = p_player_id;
        SELECT 'Move␣to␣successive␣dimension␣OK'
        AS message;
    ELSE
        SELECT 'Going␣back␣in␣dimension␣not␣allowed'
        AS message;
    END IF;
END;
//
DELIMITER //
```

Spiegazione Riga per Riga:

- `CREATE PROCEDURE ChangeDimension(IN p_player_id INT, IN new_dimension_id INT)`: Definisce una nuova procedura chiamata `ChangeDimension`.
- `DECLARE current_dimension_id INT;`: Dichiara una variabile `current_dimension_id` di tipo `INT`.
- `SELECT dimension_id INTO current_dimension_id FROM Travel WHERE player_id = p_player_id;`: Ottiene l'ID della dimensione corrente del giocatore.
- `IF new_dimension_id = current_dimension_id + 1 THEN`: Controlla se la nuova dimensione è la successiva dimensione sequenziale.
- `UPDATE Travel SET dimension_id = new_dimension_id WHERE player_id = p_player_id;`: Aggiorna l'ID della dimensione per il giocatore.
- `SELECT 'Move to successive dimension OK' AS message;`: Messaggio di successo per il cambio di dimensione.
- `ELSE SELECT 'Skipping dimension not allowed' AS message;`: Messaggio di errore per salto di dimensione non consentito.
- `END IF;`: Fine del blocco `IF`.

Test_LockAchievementsIfInvalidItem

Descrizione: Questa procedura testa il trigger lock_weapon_state e verifica se gli obiettivi sono bloccati quando un oggetto del giocatore è non valido.

```
DELIMITER //
CREATE PROCEDURE Test_LockAchievementsIfInvalidItem(in who_is_trying int)
BEGIN
    SELECT 'Before␣Update' AS status,
    ca.quest_id, ca.player_id,
    ca.requires_all_player_items
    FROM Check_Achievement ca
    JOIN Complete c ON ca.quest_id = c.quest_id
    WHERE c.player_id = who_is_trying;

    UPDATE Player_Item SET state = FALSE
    WHERE player_item_id = who_is_trying;

    SELECT 'After␣Update' AS status,
    ca.quest_id, ca.player_id,
    ca.requires_all_player_items
    FROM Check_Achievement ca
```

```
    JOIN Complete c ON ca.quest_id = c.quest_id
    WHERE c.player_id = who_is_trying;

    UPDATE Player_Item SET state = TRUE
    WHERE player_item_id = who_is_trying;

    SELECT 'After␣Attempting␣to␣Change␣Back' AS status,
    pi.player_item_id, p.player_name,
    i.item_name, pi.state
    FROM Player_Item pi
    JOIN Player p ON pi.player_id = p.player_id
    JOIN Item i ON pi.item_id = i.item_id
    WHERE pi.player_id = who_is_trying;
END;
//
DELIMITER //
```

Spiegazione Riga per Riga:

- `CREATE PROCEDURE Test_LockAchievementsIfInvalidItem()`: Definisce una nuova procedura chiamata `Test_LockAchievementsIfInvalidItem`.
- `SELECT 'Before Update' AS status, ca.quest_id, ca.player_id, ca.requires_all_player_items FROM Check_Achievement ca JOIN Complete c ON ca.quest_id = c.quest_id WHERE c.player_id = 1;`: Visualizza gli obiettivi prima dell'aggiornamento.
- `UPDATE Player_Item SET state = FALSE WHERE player_item_id = 1;`: Aggiorna lo stato di un oggetto valido a `FALSE`.
- `SELECT 'After Update' AS status, ca.quest_id, ca.player_id, ca.requires_all_player_items FROM Check_Achievement ca JOIN Complete c ON ca.quest_id = c.quest_id WHERE c.player_id = 1;`: Visualizza gli obiettivi dopo l'aggiornamento.
- `UPDATE Player_Item SET state = TRUE WHERE player_item_id = 1;`: Tenta di ripristinare lo stato (questo dovrebbe fallire a causa del trigger).
- `SELECT 'After Attempting to Change Back' AS status, pi.player_item_id, p.player_name, i.item_name, pi.state FROM Player_Item pi JOIN Player p ON pi.player_id = p.player_id JOIN Item i ON pi.item_id = i.item_id WHERE pi.player_id = 1;`: Verifica se lo stato rimane `FALSE`.

Test Procedures Calls

Descrizione: Chiama le procedure di test per verificare il loro funzionamento.

```
CALL ChangeDimension(1, 2);
CALL Test_InsertPlayerItem(3);
CALL Test_LockAchievementsIfInvalidItem(3);
```

Spiegazione Riga per Riga:

- `CALL ChangeDimension(1, 2);`: Chiama la procedura `ChangeDimension` per il giocatore 1 per passare alla dimensione 2.
- `CALL Test_InsertPlayerItem();`: Chiama la procedura `Test_InsertPlayerItem` per verificare il funzionamento dell'inserimento degli oggetti.
- `CALL Test_LockAchievementsIfInvalidItem();`: Chiama la procedura `Test_LockAchievementsIfInvalidItem` per verificare il funzionamento del trigger e il blocco degli obiettivi.

3 Schema concettuale

3.1 Entità e Attributi

Entità	Descrizione
Giocatore	Rappresenta i giocatori nel gioco.
NPC	Rappresenta i personaggi non giocanti.
Gilda	Rappresenta le gilde.
Missione	Rappresenta le missioni.
Obiettivo	Rappresenta gli obiettivi.
Dimensione	Rappresenta le dimensioni.
Oggetto	Rappresenta gli oggetti.
Giocatore_Oggetto	Rappresenta gli oggetti posseduti dai giocatori.

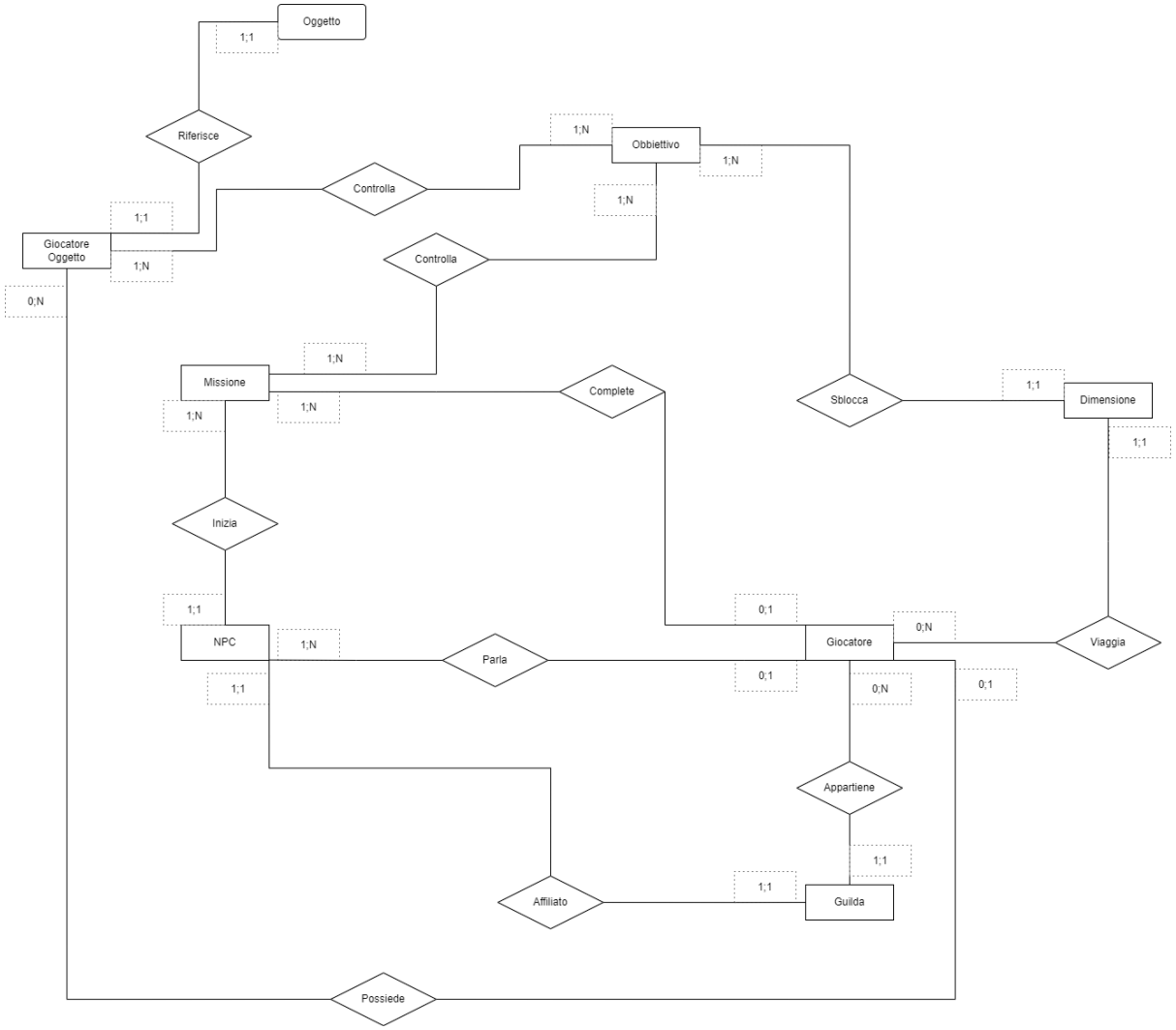

Figure 1: Schema concettuale

3.2 Relazioni

3.2.1 Descrizione delle Relazioni

- `giocatore-completa-missione`: Questa relazione indica che un giocatore ha completato una missione specifica.
- `giocatore-parla-npc`: Questa relazione registra le interazioni tra giocatori e NPC.
- `giocatore-appartiene-gilda`: Questa relazione denota a quale gilda appartiene un giocatore.

- `giocatore-possiede-giocatore_oggetto`: Questa relazione indica quali oggetti sono posseduti da un giocatore.
- `giocatore-viaggia-dimensione`: Questa relazione indica le dimensioni a cui un giocatore ha viaggiato.
- `gilda-affiliato-npc`: Questa relazione indica l'affiliazione tra gilde e NPC.
- `npc-inizia-missione`: Questa relazione denota le missioni iniziate dagli NPC.
- `missione-controlla-obiettivo`: Questa relazione indica le missioni richieste per ottenere obiettivi specifici.
- `giocatore_oggetto-controlla-obiettivo`: Questa relazione indica quali oggetti del giocatore sono controllati per gli obiettivi.
- `giocatore_oggetto-riferisce-ogggetto`: Questa relazione denota quali oggetti sono riferiti dagli oggetti del giocatore.
- `obiettivo-sblocca-dimensione`: Questa relazione indica quali obiettivi sbloccano specifiche dimensioni..

Entità (0/1;1/n)	**Relazione**	**Entità (0/1;1/n)**
Giocatore (0/1)	completa	Missione (1/n)
Giocatore (0/1)	Parla	NPC (1/n)
Giocatore (0/n)	appartiene	Gilda (1/1)
Giocatore (0/1)	possiede	Giocatore_Oggetto (0/n)
Giocatore (0/n)	viaggia	Dimensione (1/1)
Gilda (1/1)	affiliato	NPC (1/1)
NPC (1/1)	inizia	Missione (1/n)
Missione (1/n)	controlla(obiettivo)	Obiettivo (1/n)
Giocatore_Oggetto (1/n)	controlla(obiettivo)	Obiettivo (1/n)
Giocatore_Oggetto (1/1)	riferisce	Oggetto (1/1)
Obiettivo (1/n)	sblocca	Dimensione (1/1)

4 Analisi della ridondanza

Per analizzare la ridondanza, è importante esaminare le entità e le relazioni per individuare eventuali duplicazioni o dati che possono essere derivati da altre informazioni esistenti. Di seguito è riportata l'analisi di ridondanza per ciascuna entità e relazione:

Entità

- **Giocatore**: Necessaria per rappresentare i giocatori nel gioco.
- **NPC**: Necessaria per rappresentare i personaggi non giocanti.
- **Gilda**: Necessaria per rappresentare le gilde.
- **Missione**: Necessaria per rappresentare le missioni.
- **Obiettivo**: Necessaria per rappresentare gli obiettivi.
- **Dimensione**: Necessaria per rappresentare le dimensioni del gioco.
- **Oggetto**: Necessaria per rappresentare gli oggetti nel gioco.
- **Giocatore_Oggetto**: Necessaria per rappresentare gli oggetti posseduti dai giocatori.

Non sembra esserci ridondanza, poiché ciascuna di esse rappresenta un concetto distinto nel contesto del gioco.

Relazioni

- `giocatore-completa-missione`: Necessaria per tracciare quali missioni sono state completate da ciascun giocatore.
- `giocatore-parla-npc`: Necessaria per registrare le interazioni tra giocatori e NPC.
- `giocatore-appartiene-gilda`: Necessaria per denotare l'appartenenza di un giocatore a una gilda.
- `giocatore-possiede-giocatore_oggetto`: Necessaria per indicare quali oggetti sono posseduti da un giocatore.
- `giocatore-viaggia-dimensione`: Necessaria per tracciare le dimensioni visitate da un giocatore.
- `gilda-affiliato-npc`: Necessaria per indicare l'affiliazione tra gilde e NPC.
- `npc-inizia-missione`: Necessaria per denotare le missioni iniziate dagli NPC.
- `missione-controlla-obiettivo`: Necessaria per indicare le missioni richieste per ottenere specifici obiettivi.
- `giocatore_oggetto-controlla-obiettivo`: Necessaria per indicare quali oggetti del giocatore sono controllati per gli obiettivi.
- `giocatore_oggetto-riferisce-oggetto`: Necessaria per evitare che i giocatori vadano in soft-lock se i loro oggetti non sono nella lista degli oggetti ammessi.
- `obiettivo-sblocca-dimensione`: Necessaria per indicare quali obiettivi sbloccano specifiche dimensioni.

Dopo il chiarimento, nessuna delle relazioni risulta ridondante. Ogni relazione rappresenta un aspetto cruciale delle dinamiche del gioco e non può essere derivata direttamente da altre relazioni senza perdere informazioni importanti.

Dopo l'analisi, non è stata identificata alcuna ridondanza nelle entità e nelle relazioni e perciò lo schema resta uguale

5 Schema logico

5.1 Entità e Attributi

Entità	**Attributi**
Giocatore	player_id (INT, PK), player_name (VARCHAR), guild_id (INT, FK)
NPC	npc_id (INT, PK), npc_name (VARCHAR), guild_id (INT, FK)
Gilda	guild_id (INT, PK), guild_name (VARCHAR)
Missione	quest_id (INT, PK), quest_name (VARCHAR), state (BOOLEAN)
Obiettivo	achievement_id (INT, PK), achievement_name (VARCHAR)
Dimensione	dimension_id (INT, PK), dimension_name (VARCHAR)
Oggetto	item_id (INT, PK), item_name (VARCHAR)
Giocatore_Oggetto	player_item_id (INT, PK), player_id (INT, FK), item_id (INT, FK), state (BOOLEAN)

5.2 Relazioni

Relazione	**Attributi**
Completa	player_id (INT, FK), quest_id (INT, FK)
Continua alla pagina successiva	

Table 4 – continua dalla pagina precedente

Relazione	Attributi
Parla	player_id (INT, FK), npc_id (INT, FK)
Appartiene	player_id (INT, FK), guild_id (INT, FK)
Possiede	player_id (INT, FK), player_item_id (INT, FK)
Viaggia	player_id (INT, FK), dimension_id (INT, FK)
Affiliato	guild_id (INT, FK), npc_id (INT, FK), affiliation (VARCHAR)
Inizia	npc_id (INT, FK), quest_id (INT, FK)
Controlla	quest_id (INT, FK), player_id (INT, FK)
Riferisce	player_item_id (INT, FK), item_id (INT, FK)
Sblocca	achievement_id (INT, FK), dimension_id (INT, FK)

6 Normalizzazione dello schema logico

6.1 Prima Forma Normale (1NF)

- Giocatore: Ogni giocatore ha un unico player_id, player_name e guild_id.
- NPC: Ogni NPC ha un unico npc_id, npc_name e guild_id.
- Gilda: Ogni gilda ha un unico guild_id e guild_name.
- Missione: Ogni missione ha un unico quest_id, quest_name e state.
- Obiettivo: Ogni obiettivo ha un unico achievement_id e achievement_name.
- Dimensione: Ogni dimensione ha un unico dimension_id e dimension_name.
- Oggetto: Ogni oggetto ha un unico item_id e item_name.
- Giocatore_Oggetto: Ogni oggetto del giocatore ha un unico player_item_id, player_id, item_id e state.

6.2 Seconda Forma Normale (2NF)

- Giocatore: player_name e guild_id dipendono interamente da player_id.
- NPC: npc_name e guild_id dipendono interamente da npc_id.
- Gilda: guild_name dipende interamente da guild_id.
- Missione: quest_name e state dipendono interamente da quest_id.
- Obiettivo: achievement_name dipende interamente da achievement_id.
- Dimensione: dimension_name dipende interamente da dimension_id.
- Oggetto: item_name dipende interamente da item_id.
- Giocatore_Oggetto: player_id, item_id e state dipendono interamente da player_item_id.

6.3 Terza Forma Normale (3NF)

- Giocatore: Non esistono dipendenze transitive.
- NPC: Non esistono dipendenze transitive.
- Gilda: Non esistono dipendenze transitive.
- Missione: Non esistono dipendenze transitive.
- Obiettivo: Non esistono dipendenze transitive.
- Dimensione: Non esistono dipendenze transitive.
- Oggetto: Non esistono dipendenze transitive.
- Giocatore_Oggetto: Non esistono dipendenze transitive.