DataBase Project

Mutua Fadhla Mohamed — SM3201434

July 14, 2024

1 Description of what the database does

This database manages a game world where players, NPCs (non-player characters), and quests interact. Players can join guilds, complete quests, and earn achievements that unlock dimensions. NPCs can belong to guilds and initiate quests. The database ensures data integrity and proper relationships between entities using triggers and constraints.

2 Queries

2.1 Triggers

2.2 lock_achievements_if_invalid_item

`General Description:` This trigger locks all achievements for a player if any of the player's items become invalid (state is false).

```
CREATE TRIGGER lock_achievements_if_invalid_item
AFTER UPDATE ON Player_Item
FOR EACH ROW
BEGIN
    DECLARE player_achievements_locked BOOLEAN DEFAULT FALSE;

    -- Check if any item for the player is invalid
    IF EXISTS (SELECT 1 FROM Player_Item
        WHERE player_id = NEW.player_id AND state = FALSE) THEN
        SET player_achievements_locked = TRUE;
    END IF;

    -- Create a temporary table to hold achievement IDs
    CREATE TEMPORARY TABLE Temp_Achievements AS
    SELECT ca.achievement_id
    FROM Check_Achievement ca
    JOIN Complete c ON c.quest_id = ca.quest_id
    WHERE c.player_id = NEW.player_id;

    -- Update the achievement status based on item state
    UPDATE Check_Achievement
    SET requires_all_player_items = NOT player_achievements_locked
    WHERE achievement_id IN (SELECT achievement_id FROM Temp_Achievements);

    -- Drop the temporary table
    DROP TEMPORARY TABLE Temp_Achievements;
END;
```

`Line-by-Line Explanation:`

- CREATE TRIGGER lock_achievements_if_invalid_item: Defines a new trigger named lock_achievements_if_invalid_item .

- AFTER UPDATE ON Player_Item: Specifies that this trigger will be executed after an update operation on the Player_Item table.
- FOR EACH ROW: Indicates that the trigger will be executed for each row that is being updated.
- BEGIN: Starts the block of SQL statements that make up the trigger.
- DECLARE player_achievements_locked BOOLEAN DEFAULT FALSE: Declares a variable player_achievements_locked and initializes it to FALSE.
- IF EXISTS (SELECT 1 FROM Player_Item
  WHERE player_id = NEW.player_id AND state = FALSE) THEN: Checks if any item for the player is invalid (state is false).
- SET player_achievements_locked = TRUE: If any invalid item is found, set the variable player_achievements_locked to TRUE.
- CREATE TEMPORARY TABLE Temp_Achievements AS SELECT ca.achievement_id
  FROM Check_Achievement ca JOIN Complete c ON c.quest_id = ca.quest_id
  WHERE c.player_id = NEW.player_id: Creates a temporary table to hold achievement IDs for the player.
- UPDATE Check_Achievement SET requires_all_player_items = NOT player_achievements_locked
  WHERE achievement_id IN (SELECT achievement_id FROM Temp_Achievements):
  Updates the achievement status based on the item state.
- DROP TEMPORARY TABLE Temp_Achievements: Drops the temporary table.
- END: Ends the trigger.

2.3 unlock_dimension

`General Description:` This trigger unlocks dimensions for a player if all their achievements are true.

```
CREATE TRIGGER unlock_dimension
AFTER UPDATE ON Complete
FOR EACH ROW
BEGIN
    DECLARE all_achievements_true BOOLEAN;
    DECLARE player_id INT;
    SET player_id = NEW.player_id;

    SELECT COUNT(*) = 0 INTO all_achievements_true
    FROM Achievement
    JOIN Check_Achievement ON Achievement.achievement_id = Check_Achievement.achievemen
    WHERE Check_Achievement.requires_all_player_items = TRUE
    AND Check_Achievement.quest_id NOT IN (
        SELECT quest_id
        FROM Complete
        WHERE player_id = player_id
    );

    IF all_achievements_true THEN
        INSERT IGNORE INTO Travel (player_id, dimension_id)
        SELECT player_id, dimension_id
        FROM Unlocks;
    END IF;
END;
```

`Line-by-Line Explanation:`

- CREATE TRIGGER unlock_dimension: Defines a new trigger named unlock_dimension.

- AFTER UPDATE ON Complete: Specifies that this trigger will be executed after an update operation on the Complete table.
- FOR EACH ROW: Indicates that the trigger will be executed for each row that is being updated.
- BEGIN: Starts the block of SQL statements that make up the trigger.
- DECLARE all_achievements_true BOOLEAN: Declares a variable all_achievements_true.
- DECLARE player_id INT: Declares a variable player_id.
- SET player_id = NEW.player_id: Sets the player_id to the ID of the player from the updated row.
- SELECT COUNT(*) = 0 INTO all_achievements_true FROM Achievement
  JOIN Check_Achievement ON Achievement.achievement_id = Check_Achievement.achievement_id
  WHERE Check_Achievement.requires_all_player_items = TRUE AND
  Check_Achievement.quest_id NOT IN (SELECT quest_id FROM Complete
  WHERE player_id = player_id):
  Checks if all achievements are true for the player.
- IF all_achievements_true THEN: If all achievements are true, then proceed.
- INSERT IGNORE INTO Travel (player_id, dimension_id)
  SELECT player_id, dimension_id FROM Unlocks: Inserts records into the Travel table to unlock dimensions for the player.
- END IF: Ends the conditional statement.
- END: Ends the trigger.

2.4 assign_quests_after_talk

`General Description:` This trigger assigns quests to a player based on their guild affiliation after talking to an NPC.

```
CREATE TRIGGER assign_quests_after_talk
AFTER INSERT ON Talks
FOR EACH ROW
BEGIN
    DECLARE npc_guild_id INT;
    DECLARE player_guild_id INT;

    SELECT guild_id INTO npc_guild_id FROM NPC WHERE npc_id = NEW.npc_id;
    SELECT guild_id INTO player_guild_id FROM Player WHERE player_id = NEW.player_id;

    -- If NPC has no guild affiliation and player has no guild, assign quest
    IF npc_guild_id IS NULL AND player_guild_id IS NULL THEN
        INSERT INTO Complete (player_id, quest_id)
        SELECT NEW.player_id, quest_id
        FROM Initiate
        WHERE npc_id = NEW.npc_id
        LIMIT 1;
    END IF;

    -- If NPC and player have the same guild affiliation, assign quest
    IF npc_guild_id = player_guild_id THEN
        INSERT INTO Complete (player_id, quest_id)
        SELECT NEW.player_id, quest_id
        FROM Initiate
        WHERE npc_id = NEW.npc_id
        LIMIT 1;
    END IF;
END;
```

`Line-by-Line Explanation:`

- CREATE TRIGGER assign_quests_after_talk: Defines a new trigger named assign_quests_after_talk .
- AFTER INSERT ON Talks: Specifies that this trigger will be executed after an insert operation on the Talks table.
- FOR EACH ROW: Indicates that the trigger will be executed for each row that is being inserted.
- BEGIN: Starts the block of SQL statements that make up the trigger.
- DECLARE npc_guild_id INT: Declares a variable npc_guild_id.
- DECLARE player_guild_id INT: Declares a variable player_guild_id .
- SELECT guild_id INTO npc_guild_id FROM NPC WHERE npc_id = NEW.npc_id: Retrieves the guild ID of the NPC and stores it in npc_guild_id.
- SELECT guild_id INTO player_guild_id FROM Player WHERE player_id = NEW.player_id: Retrieves the guild ID of the player and stores it in player_guild_id .
- IF npc_guild_id IS NULL AND player_guild_id IS NULL THEN: Checks if both the NPC and player have no guild affiliation.
- INSERT INTO Complete (player_id, quest_id)
  SELECT NEW.player_id, quest_id FROM Initiate WHERE npc_id = NEW.npc_id
  LIMIT 1: Assigns a quest to the player.
- END IF: Ends the conditional statement.
- IF npc_guild_id = player_guild_id THEN: Checks if the NPC and player have the same guild affiliation.
- INSERT INTO Complete (player_id, quest_id)
  SELECT NEW.player_id, quest_id FROM Initiate WHERE npc_id = NEW.npc_id
  LIMIT 1: Assigns a quest to the player.
- END IF: Ends the conditional statement.
- END: Ends the trigger.

2.5 Queries

2.6 Essential Queries

2.6.1 Test_UpdatePlayerItem

`General Description:` This procedure tests updating player items to see if they remain true.

```
CREATE PROCEDURE Test_UpdatePlayerItem()
BEGIN
    UPDATE Player_Item
    SET state = TRUE
    WHERE player_item_id IN (SELECT player_item_id FROM Player_Item LIMIT 5);
END;
```

`Line-by-Line Explanation:`

- CREATE PROCEDURE Test_UpdatePlayerItem(): Defines a stored procedure named Test_UpdatePlayerItem.
- BEGIN: Starts the block of SQL statements that make up the procedure.
- UPDATE Player_Item SET state = TRUE WHERE player_item_id IN
  (SELECT player_item_id FROM Player_Item LIMIT 5): Updates the state of the first five player items to TRUE.
- END: Ends the procedure.

2.6.2 Test_InsertQuestCompletion

General Description: This procedure tests inserting quest completion records.

```
CREATE PROCEDURE Test_InsertQuestCompletion()
BEGIN
    INSERT INTO Complete (player_id, quest_id)
    VALUES (1, 1), (2, 2), (3, 3), (4, 4), (5, 5);
END;
```

Line-by-Line Explanation:

- CREATE PROCEDURE Test_InsertQuestCompletion(): Defines a stored procedure named Test_InsertQuestCompletion.
- BEGIN: Starts the block of SQL statements that make up the procedure.
- INSERT INTO Complete (player_id, quest_id) VALUES (1, 1), (2, 2), (3, 3), (4, 4), (5, 5): Inserts completion records for the specified player and quest IDs.
- END: Ends the procedure.

2.6.3 ViewAllPlayerItemsDetailed

General Description: This query retrieves detailed information about all player items.

```
CREATE PROCEDURE ViewAllPlayerItemsDetailed()
BEGIN
    SELECT pi.player_item_id, pi.player_id, pi.item_id, i.item_name, pi.state
    FROM Player_Item pi
    JOIN Item i ON pi.item_id = i.item_id;
END;
```

Line-by-Line Explanation:

- CREATE PROCEDURE ViewAllPlayerItemsDetailed(): Defines a stored procedure named ViewAllPlayerItemsDetailed.
- BEGIN: Starts the block of SQL statements that make up the procedure.
- SELECT pi.player_item_id, pi.player_id, pi.item_id, i.item_name, pi.state FROM Player_Item pi JOIN Item i ON pi.item_id = i.item_id: Selects detailed information about all player items, including item names and states.
- END: Ends the procedure.

2.6.4 ViewAllQuestsDetailed

General Description: This query retrieves detailed information about all quests.

```
CREATE PROCEDURE ViewAllQuestsDetailed()
BEGIN
    SELECT quest_id, quest_name, state
    FROM Quest;
END;
```

Line-by-Line Explanation:

- CREATE PROCEDURE ViewAllQuestsDetailed(): Defines a stored procedure named ViewAllQuestsDetailed.
- BEGIN: Starts the block of SQL statements that make up the procedure.
- SELECT quest_id, quest_name, state FROM Quest: Selects detailed information about all quests, including their states.
- END: Ends the procedure.

2.6.5 ViewAllCompletedQuestsDetailed

`General Description:` This query retrieves detailed information about all completed quests.

```
CREATE PROCEDURE ViewAllCompletedQuestsDetailed()
BEGIN
    SELECT c.player_id, p.player_name, q.quest_name, q.state
    FROM Complete c
    JOIN Player p ON c.player_id = p.player_id
    JOIN Quest q ON c.quest_id = q.quest_id
    ORDER BY p.player_id, q.quest_name;
END;
```

`Line-by-Line Explanation:`

- CREATE PROCEDURE ViewAllCompletedQuestsDetailed(): Defines a stored procedure named ViewAllCompletedQuestsDetailed.
- BEGIN: Starts the block of SQL statements that make up the procedure.
- SELECT c.player_id, p.player_name, q.quest_name, q.state FROM Complete c
  JOIN Player p ON c.player_id = p.player_id
  JOIN Quest q ON c.quest_id = q.quest_id
  ORDER BY p.player_id, q.quest_name: Selects detailed information about all completed quests, including player names and quest states.
- END: Ends the procedure.
- ViewNPCGuilds: Retrieves and displays NPCs along with their guild names, defaulting to an empty string if they do not belong to a guild.

3 Conceptual schema

3.1 Entities and Attributes

Entity	Description
Player	Represents players in the game.
NPC	Represents non-player characters.
Guild	Represents guilds.
Quest	Represents quests.
Achievement	Represents achievements.
Dimension	Represents dimensions.
Item	Represents items.
Player_Item	Represents items owned by players.

3.2 Relationships

3.3 Relationship Descriptions

- player−complete−quest: This relationship indicates that a player has completed a specific quest.
- player−talks−npc: This relationship records interactions between players and NPCs.
- player−belong−guild: This relationship denotes which guild a player belongs to.
- player−own−player_item: This relationship indicates which items are owned by a player.
- player−travel−dimension: This relationship indicates the dimensions a player has traveled to.
- guild− affiliated −npc: This relationship indicates the affiliation between guilds and NPCs.

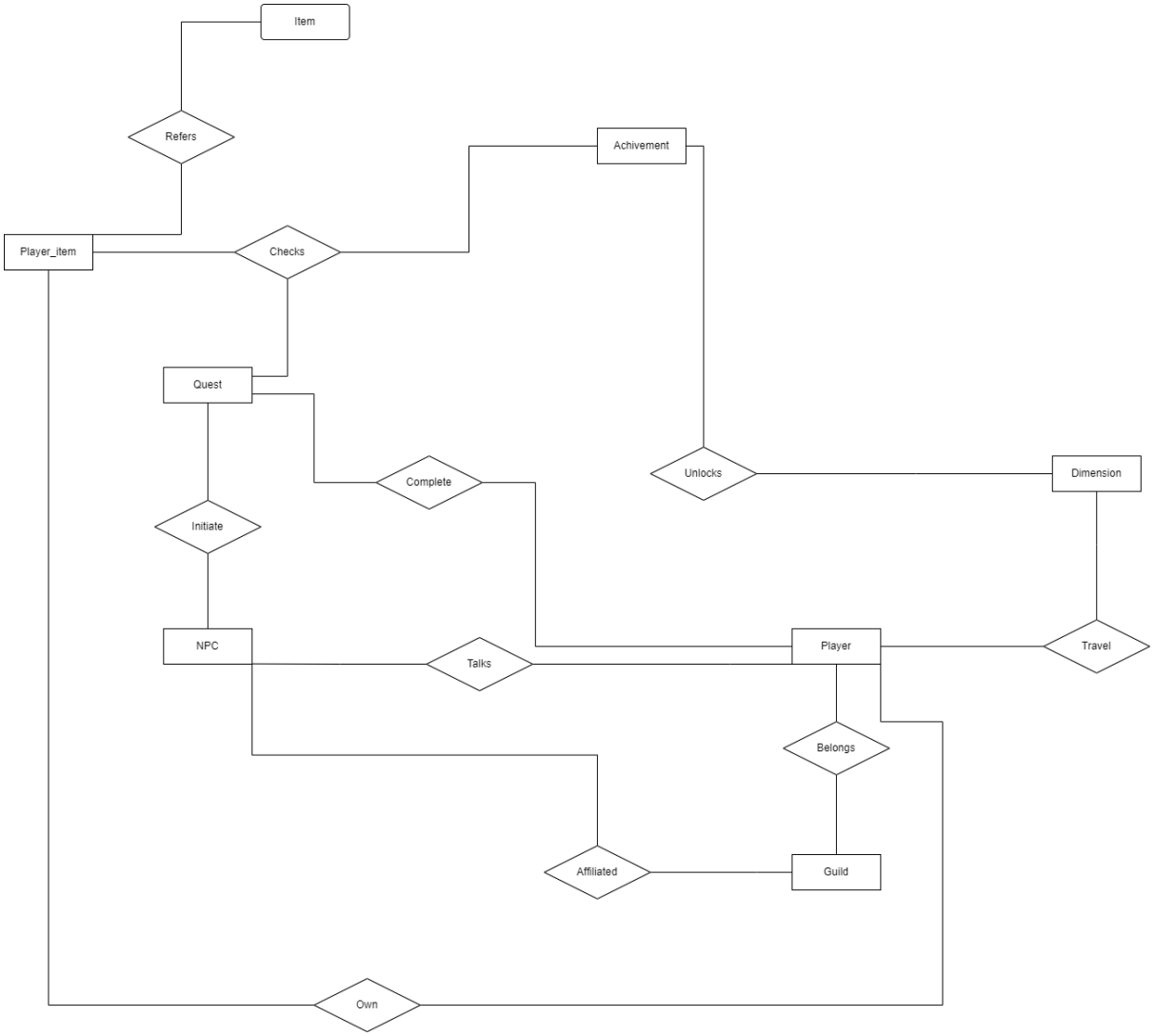

Figure 1: Conceptual schema

- npc−initiate−quest: This relationship denotes the quests initiated by NPCs.
- quest−check−achievement: This relationship indicates the quests that are required for achieving specific achievements.
- quest−check−player_item: This relationship indicates the quests that require specific player items.
- player_item−check−achievement: This relationship indicates which player items are checked for achievements.
- player_item−refers−item: This relationship denotes which items are referred to by player items.
- achievement−unlocks−dimension: This relationship indicates which achievements unlock specific dimensions.

Entity (0/1;1/n)	**Relationship**	**Entity (0/1;1/n)**
Player (0/1)	Complete	Quest (1/n)
Player (0/1)	Talks	NPC (1/n)
Player (0/n)	Belong	Guild (1/1)
Player (0/1)	Own	Player_Item (0/n)
Continued on next page		

Table 2 – continued from previous page

Entity (0/1;1/n)	Relationship	Entity (0/1;1/n)
Player (0/1)	Travel	Dimension (1/n)
Guild (0/1)	Affiliated	NPC (1/n)
NPC (0/1)	Initiate	Quest (1/n)
Quest (0/1)	Check Achievement	Achievement (1/n)
Quest (0/1)	Check Player_Item	Player_Item (1/n)
Player_Item (0/1)	Check Achievement	Achievement (1/n)
Player_Item (0/1)	Refers	Item (1/n)
Achievement (0/1)	Unlocks	Dimension (1/n)

4 Detailed Redundancy Analysis

- **Player_Item and Quest:**
  - **Redundancy in Checking Items and Achievements:**
    - Quest table includes attributes to check player items and achievements.
    - Player_Item also checks achievements, leading to redundancy.
    - Separate tables are used to check relationships between quests, items, and achievements, causing overlapping data.
  - **Overlapping Attributes:**
    - Quest and Check_Achievement both reference achievement requirements for quests.
    - Player_Item and Check_Achievement both reference player item checks.
- **Guild and NPC:**
  - **Redundant Relationships:**
    - Guild-Affiliated-NPC and Player-Belong-Guild relationships overlap.
    - Information about guild affiliation is duplicated.

5 Removal/addition caused by redundancy analysis

- Removal of Overlapping Attributes:
  - Merged the relationships between Quest, Player_Item, and Check_Achievement into a unified structure.
- Simplification of Relationships:
  - Unified guild affiliation information to avoid redundancy.

5.1 Resulting Conceptual Schema

Entity	Description
Player	Represents players in the game.
NPC	Represents non-player characters.
Guild	Represents guilds.
Quest	Represents quests.
Achievement	Represents achievements.
Dimension	Represents dimensions.
Item	Represents items.
Player_Item	Represents items owned by players.

6 Logical schema

6.1 Entities and Attributes

Entity	Attributes
Player	player_id (INT, PK), player_name (VARCHAR), guild_id (INT, FK)
NPC	npc_id (INT, PK), npc_name (VARCHAR), guild_id (INT, FK)
Guild	guild_id (INT, PK), guild_name (VARCHAR)
Quest	quest_id (INT, PK), quest_name (VARCHAR), state (BOOLEAN)
Achievement	achievement_id (INT, PK), achievement_name (VARCHAR)
Dimension	dimension_id (INT, PK), dimension_name (VARCHAR)
Item	item_id (INT, PK), item_name (VARCHAR)
Player_Item	player_item_id (INT, PK), player_id (INT, FK), item_id (INT, FK), state (BOOLEAN)

6.2 Relationships

Relationship	Attributes
Complete	player_id (INT, FK), quest_id (INT, FK)
Talks	player_id (INT, FK), npc_id (INT, FK)
Belong	player_id (INT, FK), guild_id (INT, FK)
Own	player_id (INT, FK), player_item_id (INT, FK)
Travel	player_id (INT, FK), dimension_id (INT, FK)
Affiliated	guild_id (INT, FK), npc_id (INT, FK), affiliation (VARCHAR)
Initiate	npc_id (INT, FK), quest_id (INT, FK)
Check_Achievement	quest_id (INT, FK), achievement_id (INT, FK), requires_all_player_items (BOOLEAN)
Check_Player_Item	quest_id (INT, FK), player_item_id (INT, FK)
Refers	player_item_id (INT, FK), item_id (INT, FK)
Unlock	achievement_id (INT, FK), dimension_id (INT, FK)

7 Normalization of logical schema

7.1 First Normal Form (1NF)

- Player: Each player has a unique player_id, player_name, and guild_id.
- NPC: Each NPC has a unique npc_id, npc_name, and guild_id.
- Guild: Each guild has a unique guild_id and guild_name.
- Quest: Each quest has a unique quest_id, quest_name, and state.
- Achievement: Each achievement has a unique achievement_id and achievement_name.
- Dimension: Each dimension has a unique dimension_id and dimension_name.
- Item: Each item has a unique item_id and item_name.
- Player_Item: Each player item has a unique player_item_id, player_id, item_id, and state.

7.2 Second Normal Form (2NF)

- Player: player_name and guild_id are fully dependent on player_id.
- NPC: npc_name and guild_id are fully dependent on npc_id.
- Guild: guild_name is fully dependent on guild_id.
- Quest: quest_name and state are fully dependent on quest_id.
- Achievement: achievement_name is fully dependent on achievement_id.
- Dimension: dimension_name is fully dependent on dimension_id.
- Item: item_name is fully dependent on item_id.
- Player_Item: player_id, item_id, and state are fully dependent on player_item_id.

7.3 Third Normal Form (3NF)

- Player: No transitive dependencies exist.
- NPC: No transitive dependencies exist.
- Guild: No transitive dependencies exist.
- Quest: No transitive dependencies exist.
- Achievement: No transitive dependencies exist.
- Dimension: No transitive dependencies exist.
- Item: No transitive dependencies exist.
- Player_Item: No transitive dependencies exist.