SQL Database Design and Queries for Dimensional Transfer Game

July 12, 2024

1 Project Description

The project involves the creation of a database schema for a game. The database will track various entities such as players, non-player characters (NPCs), quests, items, achievements, guilds, and dimensions. It will also record relationships and actions such as quest completion, item ownership, and guild membership.

2 SQL Operations

2.1 View Player Stats

This procedure displays the basic stats of a specific player, including their name, level, experience, and player score.

```
DELIMITER //
CREATE PROCEDURE ViewPlayerStats(IN playerID INT)
BEGIN
    SELECT name, level, experience, player_score
    FROM Player
    WHERE player_id = playerID;
END //
DELIMITER ;
```

Listing 1: View Player Stats

2.2 View Inventory

This procedure lists all items in a player's inventory, showing item name, quantity, type, and value.

```
DELIMITER //
CREATE PROCEDURE ViewInventory(IN playerID INT)
BEGIN
    SELECT Item.name AS ItemName, Player_Item.quantity, Item.type,
        Item.value
    FROM Player_Item
```

```
    INNER JOIN Item ON Player_Item.item_id = Item.item_id
    WHERE Player_Item.player_id = playerID;
END //
DELIMITER ;
```

Listing 2: View Inventory

2.3 View Completed Quests

This procedure shows the completed quests for a specific player by displaying the names of quests that the player has completed.

```
DELIMITER //
CREATE PROCEDURE ViewCompletedQuests(IN playerID INT)
BEGIN
    SELECT Quest.name AS QuestName
    FROM Completion
    INNER JOIN Quest ON Completion.quest_id = Quest.quest_id
    WHERE Completion.player_id = playerID AND Completion.state =
        TRUE;
END //
DELIMITER ;
```

Listing 3: View Completed Quests

2.4 View Current Quests

This procedure shows the current quests for a specific player by displaying the names of quests that the player is currently on.

```
DELIMITER //
CREATE PROCEDURE ViewCurrentQuests(IN playerID INT)
BEGIN
    SELECT Quest.name AS QuestName
    FROM Player
    INNER JOIN Quest ON Player.quest_id = Quest.quest_id
    WHERE Player.player_id = playerID;
END //
DELIMITER ;
```

Listing 4: View Current Quests

2.5 View Achievements

This procedure lists all achievements of a player, showing the achievement name and whether the player has earned it.

```
DELIMITER //
CREATE PROCEDURE ViewAchievements(IN playerID INT)
BEGIN
    SELECT Achievement.name AS AchievementName, Earn.state
    FROM Earn
```

```
    INNER JOIN Achievement ON Earn.achievement_id = Achievement.
        achievement_id
    WHERE Earn.player_id = playerID;
END //
DELIMITER ;
```

Listing 5: View Achievements

2.6 View Guild Information

This procedure displays guild information for a specific player, including the guild name, alignment, and guild leader.

```
DELIMITER //
CREATE PROCEDURE ViewGuildInfo(IN playerID INT)
BEGIN
    SELECT Guild.name AS GuildName, Guild.alignment, Guild.
        guild_leader
    FROM Player
    INNER JOIN Guild ON Player.guild_id = Guild.guild_id
    WHERE Player.player_id = playerID;
END //
DELIMITER ;
```

Listing 6: View Guild Information

2.7 Grant Permissions to a User

This procedure grants specified permissions to a user on the database, allowing them to perform SELECT, INSERT, UPDATE, and DELETE operations.

```
DELIMITER //
CREATE PROCEDURE GrantPermissions(IN username VARCHAR(255), IN
    hostname VARCHAR(255))
BEGIN
    SET @query = CONCAT('GRANT SELECT, INSERT, UPDATE, DELETE ON
        Dimensional_Transfer.* TO ?@?');
    SET @user = username;
    SET @host = hostname;
    PREPARE stmt FROM @query;
    EXECUTE stmt USING @user, @host;
    DEALLOCATE PREPARE stmt;
END //
DELIMITER ;
```

Listing 7: Grant Permissions to a User

2.8 Add Last Login Column

This procedure adds a column to the Player table for tracking the last login time of players.

```
DELIMITER //
CREATE PROCEDURE AddLastLoginColumn()
BEGIN
    ALTER TABLE Player
    ADD COLUMN last_login DATETIME;
END //
DELIMITER ;
```

Listing 8: Add Last Login Column

2.9 Update Last Login Time

This procedure updates the last login time for a specific player to the current timestamp.

```
DELIMITER //
CREATE PROCEDURE UpdateLastLogin(IN playerID INT)
BEGIN
    UPDATE Player
    SET last_login = NOW()
    WHERE player_id = playerID;
END //
DELIMITER ;
```

Listing 9: Update Last Login Time

2.10 Add a New Player

This procedure adds a new player to the database, ensuring no duplicates, and checks the legality of items owned by the player.

```
DELIMITER //
CREATE PROCEDURE AddPlayer(IN playerName VARCHAR(255), IN guildID
    INT, IN questID INT)
BEGIN
    DECLARE newPlayerID INT;

    IF NOT EXISTS (SELECT 1 FROM Player WHERE name = playerName)
        THEN
        INSERT INTO Player (name, guild_id, quest_id) VALUES (
            playerName, guildID, questID);
        SET newPlayerID = LAST_INSERT_ID();
        CALL CheckLegality(newPlayerID);
    END IF;
END //
DELIMITER ;
```

Listing 10: Add a New Player

2.11 Update Player Progression

This procedure updates a player's experience and level, ensuring data consistency by using transactions.

```
DELIMITER //
CREATE PROCEDURE UpdatePlayerProgression(IN playerID INT, IN
    experienceGain INT)
BEGIN
    START TRANSACTION;
    UPDATE Player SET experience = experience + experienceGain
        WHERE player_id = playerID;
    UPDATE Player SET level = level + 1 WHERE player_id = playerID;
    CALL CheckLegality(playerID);
    COMMIT;
END //
DELIMITER ;
```

Listing 11: Update Player Progression

2.12 Reset Player Progression

This procedure resets a player's experience, level, and player score.

```
DELIMITER //
CREATE PROCEDURE ResetPlayerProgression(IN playerID INT)
BEGIN
    UPDATE Player
    SET experience = 0, level = 1, player_score = 0
    WHERE player_id = playerID;
END //
DELIMITER ;
```

Listing 12: Reset Player Progression

2.13 Create Index on Player Name

This procedure creates an index on the name column in the Player table to improve search performance.

```
DELIMITER //
CREATE PROCEDURE CreatePlayerNameIndex()
BEGIN
    CREATE INDEX idx_player_name ON Player(name);
END //
DELIMITER ;
```

Listing 13: Create Index on Player Name

2.14 Get Players by Guild

This procedure lists players belonging to a specific guild.

```
DELIMITER //
CREATE PROCEDURE GetPlayersByGuild(IN guildID INT)
BEGIN
    SELECT Player.name AS PlayerName
```

```
    FROM Player
    WHERE Player.guild_id = guildID;
END //
DELIMITER ;
```

Listing 14: Get Players by Guild

2.15 Get Player Inventory Value

This procedure calculates the total value of items in a player's inventory.

```
DELIMITER //
CREATE PROCEDURE GetPlayerInventoryValue(IN playerID INT)
BEGIN
    SELECT SUM(Item.value * Player_Item.quantity) AS TotalValue
    FROM Player_Item
    INNER JOIN Item ON Player_Item.item_id = Item.item_id
    WHERE Player_Item.player_id = playerID;
END //
DELIMITER ;
```

Listing 15: Get Player Inventory Value

2.16 Get Players with Specific Achievement

This procedure lists players who have earned a specific achievement.

```
DELIMITER //
CREATE PROCEDURE GetPlayersWithAchievement(IN achievementName
    VARCHAR(255))
BEGIN
    SELECT Player.name AS PlayerName
    FROM Earn
    INNER JOIN Player ON Earn.player_id = Player.player_id
    INNER JOIN Achievement ON Earn.achievement_id = Achievement.
        achievement_id
    WHERE Achievement.name = achievementName AND Earn.state = TRUE;
END //
DELIMITER ;
```

Listing 16: Get Players with Specific Achievement

2.17 Check for Illegal Items

This procedure lists players with illegal items.

```
DELIMITER //
CREATE PROCEDURE CheckForIllegalItems()
BEGIN
    SELECT Player.name AS PlayerName
    FROM Player
    WHERE player_id IN (
        SELECT player_id
```

```
        FROM Player_Item
        WHERE item_id IN (SELECT item_id FROM Item WHERE legality =
            FALSE)
    );
END //
DELIMITER ;
```

Listing 17: Check for Illegal Items

2.18 Get Players with Illegal Items

This procedure retrieves players with illegal items.

```
DELIMITER //
CREATE PROCEDURE GetPlayersWithIllegalItems()
BEGIN
    SELECT Player.name
    FROM Player
    WHERE player_id IN (
        SELECT player_id
        FROM Player_Item
        WHERE item_id IN (SELECT item_id FROM Item WHERE legality =
            FALSE)
    );
END //
DELIMITER ;
```

Listing 18: Get Players with Illegal Items

2.19 Get Total Completed Quests by Players

This procedure lists players with the total number of completed quests.

```
DELIMITER //
CREATE PROCEDURE GetTotalCompletedQuestsByPlayers()
BEGIN
    SELECT Player.name AS PlayerName, COUNT(Completion.quest_id) AS
        TotalCompletedQuests
    FROM Completion
    INNER JOIN Player ON Completion.player_id = Player.player_id
    WHERE Completion.state = TRUE
    GROUP BY Player.name
    ORDER BY TotalCompletedQuests DESC;
END //
DELIMITER ;
```

Listing 19: Get Total Completed Quests by Players

2.20 Remove Illegal Items from Player Inventory

This procedure removes illegal items from a player's inventory.

```sql
DELIMITER //
CREATE PROCEDURE RemoveIllegalItemsFromInventory(IN playerID INT)
BEGIN
    DELETE FROM Player_Item
    WHERE player_id = playerID AND item_id IN (SELECT item_id FROM
        Item WHERE legality = FALSE);
END //
DELIMITER ;
```

Listing 20: Remove Illegal Items from Player Inventory

3 Schema Concettuale

Entity 1	**Relationship**	**Entity 2**
Player (1:N)	Belong	Guild (1:1)
Player (1:N)	Complete	Quest (1:N)
Player (1:N)	Own	Player_Item (1:N)
Player_Item (1:N)	Legal_item	Item (1:N)
NPC (1:1)	Affiliation	Guild (1:1)
Dimension (1:1)	Complete	Quest (1:N)

Table 1: Entity-Relationship Descriptions

4 Schema Logico

4.1 Entità

4.2 Relazioni

5 Redundancy Analysis

Redundancy in the unnormalized schema can lead to data anomalies and inefficiencies. Detailed analysis of redundancy is as follows:

- **Player**: Contains redundant attributes guild_name and quest_name.
- **NPC**: Contains a redundant attribute guild_name.
- **Player_Item**: Contains a redundant attribute item_condition.
- **Achievement**: Contains a redundant attribute achievement_status.
- **Guild**: Contains redundant attributes guild_leader and guild_points.

Entità	Attributi 1	Attributi 2	Attributi 3
Player	player_id	name	level
	experience	[guild_id]	[quest_id]
	player_score	[guild_name]	[quest_name]
NPC	npc_id	name	role
	alignment	[guild_id]	[guild_name]
Quest	quest_id	name	description
	reward	quest_status	
Player_Item	player_item_id	[player_id]	[item_id]
	quantity	item_condition	
Item	item_id	name	type
	value	rarity	
Achievement	achievement_id	name	description
	achievement_status	date_earned	
Guild	guild_id	name	alignment
	guild_leader	guild_points	
Dimension	dimension_id	name	description
	difficulty_level		

Table 2: Entities and their Attributes

Relazioni	Attributi 1	Attributi 2	Attributi 3
Completion	[player_id]	[quest_id]	state
Belong	[player_id]	[guild_id]	[guild_name]
Own	player_item_id	[player_id]	[item_id]
	quantity	item_condition	
Affiliation	[npc_id]	[guild_id]	[guild_name]

Table 3: Relationships and their Attributes

6 Restructuring with Analysis of Redundancy and Eventual Additions/Removals

In this section, we remove redundancy from the schema. A derived attribute is one that can be calculated or inferred from other attributes in the database. We will remove such attributes and show the updated schema.

6.1 Removal of Redundancy

By removing the redundant attributes, the schema is optimized to avoid data anomalies and inefficiencies. Here is the detailed description of the changes made:

- **Player**: The attributes guild_name and quest_name were removed. These attributes can be derived from guild_id and quest_id, respectively.

- **NPC**: The attribute guild_name was removed because it can be derived from guild_id.
- **Player_Item**: The attribute item_condition was removed because it is a derived or calculated attribute.
- **Achievement**: The attribute achievement_status was removed because it can be inferred from date_earned.
- **Guild**: The attributes guild_leader and guild_points were removed because they may be derived or unnecessary depending on the use case.

6.2 Output after Redundancy Removal

The resulting entities and relationships after removing redundancy are:

6.2.1 Entities

- **Player**: `player_id, name, level, experience, guild_id, quest_id, player_score`
- **NPC**: `npc_id, name, role, alignment, guild_id`
- **Quest**: `quest_id, name, description, reward`
- **Player_Item**: `player_item_id, player_id, item_id, quantity`
- **Item**: `item_id, name, type, value`
- **Achievement**: `achievement_id, name, description, date_earned`
- **Guild**: `guild_id, name, alignment`
- **Dimension**: `dimension_id, name, description, difficulty_level`

6.2.2 Relationships

- **Completion**: `player_id, quest_id, state`
- **Belong**: `player_id, guild_id`
- **Own**: `player_item_id, player_id, item_id, quantity`
- **Affiliation**: `npc_id, guild_id`