SQL Database Design and Queries for Dimensional Transfer Game

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.

SQL Operations

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 ;
```

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 ;
```

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 ;
```

SQL Database Design and Queries for Dimensional Transfer Game

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 ;
```

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 ;
```

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 ;
```

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 ;
```

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 ;
```

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 ;
```

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 ;
```

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 ;
```

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 ;
```

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 ;
```

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 ;
```

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 ;
```

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 ;
```

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 ;
```

SQL Database Design and Queries for Dimensional Transfer Game

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 ;
```

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 ;
```

Remove Illegal Items from Player Inventory

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

```
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 ;
```

Schema Concettuale

Schema Concettuale

Entity 1	Relationship	Entity 2
Player (1:N)	Belong	Guild (1:1)

SQL Database Design and Queries for Dimensional Transfer Game

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)

Schema Logico - Entità

Schema Logico - Entità

Entità	**Descrizione**	**Attributi 1**	**Attributi 2**	**Attributi 3**
Player	A user of the game	player_id	name	level
	experience	guild_id	quest_id	
	player_score			
NPC	Non-player character	npc_id	name	role
	alignment	guild_id		
Quest	A task or mission	quest_id	name	description
	reward	quest_status		
Player_Item	Items owned by a player	player_item_id	player_id	item_id
	quantity	item_condition		
Item	Items in the game	item_id	name	type
	value	rarity		
Achievement	Achievements earned by players	achievement_id	name	description
	achievement_status	date_earned		
Guild	Groups that players can join	guild_id	name	alignment
	guild_leader	guild_points		

Dimension	Different game worlds or levels	dimension_id	name	description
	difficulty_level			

Schema Logico - Relazioni

Schema Logico - Relazioni

Relazioni	**Descrizione**	**Attributi 1**	**Attributi 2**	**Attributi 3**
Completion	Record of completed quests	player_id	quest_id	state
Belong	Membership of players in guilds	player_id	guild_id	guild_name
Own	Ownership of items by players	player_item_id	player_id	item_id
		quantity	item_condition	
Affiliation	NPC affiliation with guilds	npc_id	guild_id	guild_name

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.

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.

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.

SQL Database Design and Queries for Dimensional Transfer Game

Output after Redundancy Removal

The resulting entities and relationships after removing redundancy are:

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

Relationships

Completion: player_id, quest_id, state

SQL Database Design and Queries for Dimensional Transfer Game

Belong: player_id, guild_id

Own: player_item_id, player_id, item_id, quantity

Affiliation: npc_id, guild_id