Course Project:
Developing a Gaming Event Management System

Overview

For this project, you will be working with a fellow student to create a gaming event management system.

The project will span five weeks and is divided into weekly tasks. The project will cover the following topics:

- Database Design
- Data Wrangling
- Database Programming
- Advanced SQL Queries
- NoSQL and MySQL Databases

The only constraints to your software are the following:

- It must at least implement/create the functionality/deliverables described each week,
- You must use a database system to store the program's data. The database system must fit the requirements of the project.
  For instance, you cannot use SQLite, because the described software can be used by multiple users at a time.
- The project is a project, it is not an assignment with fixed boundaries. You may add, change and improve as you see fit. In fact, a creative approach is encouraged and will reward bonus points at the end.

You may use any programming language that the teachers can understand. If you are not sure; ask. You may use any type of database system, relational, document, graph, go wild, if you want to use some vague DBMS, make sure to discuss it with a teacher to see whether it will work. The project counts for **25%** of your total grade for this course.

During the upcoming labs there will still be opportunities for practicing topics related to databases with weekly exercises. However, the labs will also act as office hours where you can claim time from the teacher or assistant to discuss the choices you made in the project.

Deliverables:

- You will be assigned project tasks each week. These tasks also list which deliverables are part of it. Make sure to collect all of the weekly tasks' deliverables to hand them in ***at the end of the project***.
- Show your work in a screen recording presenting your software, queries and database. In the video you must show the same code that you upload to canvas. Give a 5 - 10 minute presentation of:

    - the choices you've made in terms of database API, programming paradigm and database system,
    - any extra assumptions that you made regarding the project,
    - show each week's deliverables, explaining the choices you've made and how you tackled the problem,
    - If you do not want to, you don't have to talk in your video. In that case, make sure you use another way to explain what you are presenting.
- Hand in the project on the last day of the course, please check Canvas for the exact day/time.

To start:

- Meet up with your fellow student. Subscribe to a group on canvas. Do not create your own group, join one of the pre-created groups in the "People" tab in Canvas.
- Choose your "stack", which programming language and database system you will use. Investigate what kind of database SDK you'd like and how to use it.
- Start with week 1!
- End with week 5!
- Enjoy 🙂

Rubric

Each weekly task has a rubric of its own. Your final grade is computed by summing over all rubrics (including bonus), dividing by the total (excluding bonus) and dividing this by 100. Your resulting grade is then *max(your_grade, 10)*. This grade will then count for 25% of your final grade.

Creativity Bonus

You will not just be assessed on how strictly you've adhered to the tasks, rather we expect you to think critically, find out the deeper meaning behind the question and act accordingly, while maintaining the spirit of the assignment.

The creativity bonus of up to **50 points** is up to the teacher's discretion. Here are some things that you can do to (partially) obtain it:

- Have an exceptional presentation in your video use flair and suave and show us you care about presenting
- Implement a cool extra feature in your program that was not part of the original task
- Use a unique or advanced technique to solve one of the problems
- Create a cool user interface or visualization for one or more of the tasks
- Solve one of the problems in a non-trivial creative method
- Making your program more interactive, perhaps by utilizing APIs to create a dynamic experience
- While not necessary for basic tasks, creating a highly efficient algorithm or solution could be rewarded
- Well-documented code, or a nice notebook could add to the creativity score. Emphasizing the importance of not just doing, but also explaining.
- Create something that serves a broader purpose. It could be an open-source tool, a contribution to an existing project, or a program with a social impact.

- If the project includes not just implementation but also some level of original research or an inventive problem-solving method, that could warrant extra points.

Feel free to tell us in the video what you would like to be considered for the bonus points.

The Game:
Mystic Quest: A Journey through the Lost Kingdoms

Game Lore
In a bygone age, Ethera was a utopia where arcane energies and earthly delights coexisted, sustained by the Orb of Unity—an emblem of the subtle power dynamics that governed its four realms: Valoria, Cimmeria, Elphora, and Orynthia. These kingdoms thrived not just in balance but in a harmonious interplay of power, knowledge, and magic, underpinned by the Orb's omnipresent influence. But the world's equilibrium was irrevocably disturbed when 'The Shadow' emerged, an entity that recognized the Orb not as a mechanism for harmony but as a locus of control.

Fragmenting the Orb, The Shadow imposed a new regime, a power-knowledge hierarchy that placed its minions in positions of authority, erasing centuries of collective wisdom and enlightenment. The kingdoms are now partitioned, their inhabitants subjugated into a dystopia where surveillance, censorship, and fear have supplanted unity and wonder. The very fabric of Ethera's society has been twisted into a complex web of control, resistance, and normalization, with The Shadow's overseers regulating even the most mundane aspects of daily life.

Yet, from the annals of Etherian lore comes a prophecy—a subversive narrative threatening the status quo. It whispers of an insurgent, a figure equipped not just with courage but with an understanding of the labyrinthine power structures they must dismantle. You, the player, are that catalytic agent, foretold to navigate the discursive battlefields and epistemic struggles that define the fractured kingdoms. Your quest, should you engage with its complexities and paradoxes, is to reclaim the shattered fragments of the Orb of Unity, challenge the hegemony of The Shadow, and liberate the discourses and realities of Ethera. By doing so, you're not just altering the balance of power; you're challenging the very paradigms that govern it.

Gameplay
"Mystic Quest" is an expansive online multiplayer role-playing game (MMORPG) that invites players to explore the world of Ethera. Players can choose between various character types such as Warriors, Mages, Archers, and Healers, each with their unique abilities and skill sets. The game is quest-centric, focusing on tasks like combat, exploration, crafting, and puzzle-solving.

Key Features

Character Customization:
- Create your hero with a choice of different races, classes, and skill trees.
- Personalize with cosmetics, including outfits, weapon skins, and emotes.
- Unlock abilities and talents through gameplay, allowing a unique gameplay experience.

Questing:
- Dynamic, branching quests that respond to player decisions.
- Adjustable difficulty levels offer challenges for all types of players.

- Reward system that balances risk with reward, including rare items and experience points.

Teams:
- Form or join a team with up to 5 players.
- Specialized team roles encourage cooperation.
- Bonus rewards for completing quests as a team.

Crafting:
- Gather resources from the world or loot from enemies.
- Use blueprints to craft items, weapons, and even buildings.
- Advanced crafting system for specialized items, requiring rare materials.

Events:
- Seasonal events corresponding to real-world holidays.
- Special events triggered by community milestones or achievements.
- Exclusive items and challenges only available during these periods.

Real-Time Combat:
- Realistic physics and hit detection.
- Diverse array of tactical options, including flanking, cover, and special abilities.
- PVP arenas and dueling options for competitive play.

Economy:
- Auction houses and direct trading between players.
- Dynamic supply and demand affecting item prices.
- Currency sinks to prevent inflation, such as repair and fast travel costs.

Analytics Dashboard:
- Detailed statistics on gameplay performance, time spent, and more.
- Achievement tracking and trophy system.
- Community leaderboards and personal milestones.

In-Game Entities

Kingdoms:
- Valoria: Known for its towering mountains and hardy warriors.
- Cimmeria: A lush, enchanted forest home to mystical creatures.
- Elphora: A harsh desert landscape, rich in mineral resources.
- Orynthia: A collection of tropical islands with unique ecosystems.

Classes:
- Warrior: Specializes in tanking damage and close combat.
- Mage: Offers both damage and support through a variety of spells.
- Archer: Excels at long-range damage with various types of arrows.
- Healer: Focuses on healing and buffing teammates.

Enemies:
- - Goblins: Low-level, swarm-type enemies.

- - Wolves: Fast and agile, often hunt in packs.
- - Shadow Minions: Magical creatures of the darkness.
- - Bosses: Epic fights like the Shadow King, requiring strategy and teamwork.

NPCs:

- Quest-givers: Assign tasks and offer rewards.
- Shopkeepers: Sell items and resources.
- Skill Trainers: Offer skill upgrades for a fee.
- Lore Masters: Provide backstory and clues to hidden secrets.

Guilds:

- Questing Guilds: Focus on cooperative PvE content.
- Trading Guilds: Specialize in crafting and the game's economy.
- Crafting Guilds: Provide crafting stations and blueprints to members.

Week 1: Database Schema Design

Project Background

Welcome to "The Realm of Aetheria," a sprawling, mystical world filled with diverse lands, ancient civilizations, and mythical creatures. In Aetheria, players embark on quests, engage in battles, trade items, and form guilds, all while exploring a vast landscape.

The World Structure:

Aetheria is divided into distinct regions, each with unique characteristics. These regions are filled with towns, dungeons, and landmarks. Regions are governed by rulers who reside in castles, enforcing laws and providing quests to adventurers.

The Inhabitants:

Players can create their own characters by selecting from a variety of classes such as Warrior, Mage, and Healer. They can also choose their starting region. As they gain experience, players can change classes by consulting a Master NPC.

Non-Player Characters (NPCs) populate the world and serve different roles, from shopkeepers to quest-givers, to enemies that players can fight.

Questing:

Players can undertake quests which can be as simple as collecting herbs or as complicated as defeating an ancient dragon. Quests can be provided by NPCs, rulers, or even discovered through interacting with environmental objects. Completing a quest grants players experience points and sometimes, special items.

Items and Trade:

The world is rich with items players can collect—weapons, armors, magical potions, and crafting materials. Players can buy items from shops or trade with other players. Special rare items can be earned by completing high-level quests or defeating difficult enemies.

Guilds:

Players can form or join guilds. Guilds can own property, set guild-specific quests, and wage wars against other guilds for control of valuable resources.

Combat:

Players can engage in combat with enemies and other players. Combat is turn-based, allowing each participant to choose actions like "attack," "defend," or "use item" during their turn. The outcome of the battle is determined based on the characters' attributes and choices made during the combat.

Currency:

The realm has its own economy with a standard currency, Gold, used to buy and sell items. Players can earn Gold by completing quests, selling items, or winning it in combat.

Summary:

In "The Realm of Aetheria," players can explore regions, complete quests, buy and sell items, engage in combat, and interact with NPCs and other players. They can form guilds, undertake guild-specific activities, and even participate in large-scale, guild-vs-guild events.

- **Entities and Relationships**: Based on the detailed description of "The Realm of Aetheria," identify the main entities, their attributes, and relationships.
- **Crow's Foot Notation**: Next, draw an ERD using Crow's Foot notation.
- **Key Attributes**: Clearly mark primary keys, foreign keys, and any unique constraints you envision necessary for the entities.
- **Database Setup**: Turn your ERD designs into actual databases. Create tables based on your ERD and set up constraints based on your schema design.
- **Data Insertion**: Populate your tables with some initial data. This should include at least 5 entries for each table you've created.

Assignment:

- Draw ERDs for your database schema.
- Create tables using SQL.
- Insert at least 5 rows of data into each table.

Deliverables:

- ERD in Crow’s foot notation.
- SQL scripts or schema diagram for table creation and data insertion.

Rubric:

- Correctness of ERD: 20 points
- Inclusion of all necessary entities and relationships: 10 points
- Correctness of table creation: 10 points
- Proper data insertion: 10 points

By the end of Week 1, you should have a well-defined database schema in place, complete with entities and relationships that reflect the complexities of a gaming event management system. This foundational step will serve as the backbone for all the data manipulation and querying tasks you'll undertake in the weeks to come. Note that you are allowed to change your database design after week 1. In that case you should also update your schema and ERD and hand in the updated versions.

Week 2: Database Programming and Data Loading

This week, you'll take your database to the next level by programming database interactions. A script will be provided to you that generates unstructured JSON data simulating real-world gaming events, players, quests, items, and guilds. Your task is to create a program that reads this data, interprets it, and populates your database.

Note that the data provided is often “messy” and incomplete. You will have to find ways to manage this. Here are some ways that you could do so (there’s no requirement to use any of these methods, they are merely suggestions to help you get started):

- Null Value Handling:
    - Drop rows or columns with too many missing values.
    - Impute missing values using statistical methods like mean, median, or mode.
    - Use machine learning algorithms to predict and fill missing values.
- Outlier Detection:
    - Apply statistical tests to identify outliers.
    - Use visualization techniques like box plots to visualize outliers.
    - Decide whether to cap, transform, or remove outliers based on their impact.
- Duplicate Removal:
    - Identify duplicate rows or entries.
    - Remove duplicates or average out their values if they serve a purpose.
- Normalization:
    - Standardize values to fall within a certain range or distribution.
    - This can involve scaling, transformation, or binning data points.
- Text Cleaning:
    - Remove extra spaces, special characters, and punctuation from text data.
    - Convert text to a common case (lower or upper).
- Data Type Conversion:
    - Convert data types to match the expected format, e.g., changing a string containing numbers to an integer or float.
    - Perform date/time conversions to standard formats.
- Encoding Categorical Variables:
    - Convert categorical variables into numerical form, using one-hot encoding or label encoding.
- Error Tagging:
    - Flag erroneous or suspicious records for manual review.

Some tips and best-practices:

- Keep versions of the cleaned dataset to track changes and facilitate rollback in case of errors.
- **Data Validation:** Use constraints in databases or validation in code to ensure only valid data gets stored.
- **Best Practices**
    - Always perform exploratory data analysis (EDA) first to understand the nature of "messiness" in your data.

    - Document all data cleaning steps thoroughly for transparency and reproducibility.
    - If possible, trace back to the source of the messy data and address the root issue to prevent future occurrences.

Assignment:

- Download and run the provided script to generate messy, missing, wrong and semi-structured JSON data.
- Write a program to read the JSON data and convert it into a structured format compatible with your database. You may not change the script, use it, as-is.
- Explore the data, you will run into a lot of problems and issues. Try to find the best solution you can for each problem.
- You will not end up with a perfect insertion of the data, this is not a requirement.
- Use the program to load at least three sets of generated data into your database.

Deliverables:

- Source code of the program you wrote to read and interpret the JSON data / or:
- or / SQL scripts that show the data has been loaded into your database.
- Brief documentation explaining any challenges you faced and how you overcame them. Mentioning specifically the benefits and downsides and consequences of your chosen approach.

Rubric:

- Correct interpretation of JSON data: 20 points
- Correctness of database interactions (INSERT, UPDATE statements): 20 points
- Successful data loading (demonstrated via SQL queries): 10 points
- Documentation of challenges and their solutions: 10 points

By the end of this week, you should be able to programmatically interact with your database, loading in generated game data that can later be queried and manipulated. This will set the foundation for more complex tasks in the upcoming weeks.

Week 3: A New Functionality

This week, you will enhance your gaming event management system by adding a real-time chat feature. The core of any multiplayer gaming experience is the ability to communicate with other players. Whether planning a quest, trading items, or just socializing, chatting is a critical functionality in contemporary online games.

There are two primary types of chats you will implement:

- **One-on-One Chats**: This allows two players to communicate privately. Messages are only visible to the two involved players.
- **Group Chats**: Players can join or create chat 'rooms' or 'channels,' often themed around a particular quest, item trading, or general discussion. All members of the chat group can see the messages sent here.
- **BONUS Feature: @Mentions**: To make the chat more interactive, you will also implement '@mentions.' When a player mentions another player using the '@' symbol followed by their username (e.g., @JohnDoe), that player will receive a special notification or highlight to alert them to the message.
  This feature is not mandatory, implementing it earns you bonus points toward your final grade.

By adding these chat functionalities, your system becomes more user-friendly and interactive, enhancing the overall gaming experience while adding a layer of complexity to your database and application logic.

Here are some of the possible concepts that you may find useful in implementing the chat functionality:

SQL Concepts:

- Use **triggers** to automatically notify players of new messages or when they are @mentioned in chats.
- Use **stored procedures** to encapsulate complex queries. For example, fetching the last 'n' messages in a chat or all messages in which you were mentioned.
- Leverage SQL **joins** to fetch usernames or other related data from the Players table while displaying messages.
- Always include timestamps to indicate when each message was sent.
- Create SQL **views** to simplify complex queries, particularly where multiple joins are involved.

Programming Concepts:

- Callbacks or Event Listeners: Implement callbacks or event listeners to handle events like incoming messages or notifications.
- Data Validation: Always validate the data before saving it into the database to prevent any inconsistencies.
- Security Measures: Don't forget to implement security measures, like escaping inputs to prevent SQL injection.
- Error Handling: Implement robust error handling to deal with scenarios like network errors or database unavailability.

Assignment:

1. **Extend Your Database Schema**
   Add new tables that will handle storing messages, keeping track of who sent them, and to which chat session they belong.
   Update existing tables if needed to link player information to chat logs.
2. **Client-side Interface**
   Create a minimalistic front-end through which players can send and receive messages. It can be as simple as a text-based interface in a terminal or as complex as a new webpage, depending on your preference and interest.
   Send and receive messages through the database. Meaning, use the database as a "message-broker" by using SQL to create records.
3. **Store Chats in the Database**
   Your system should automatically store chat logs in the database.
   Each message should have metadata attached, such as the sender, timestamp, and related chat session or group.

Deliverables:

- SQL schema extensions for the chat functionalities.
- Client-side interface files.
- A sample chat log showing that your system can successfully send, receive, and store messages.

Rubric:

- Correctness and completeness of the extended SQL schema: 25 points
- Correctness and effectiveness of the client-side interface: 20 points
- Integrity and proper storage of chat logs in the database: 20 points
- **BONUS:** Implementing *@mentions* in the database and interface: +20 points

By the end of this week, you should have a more dynamic gaming event management system that not only stores and manages game-related data but also enables real-time social interactions. Your system should demonstrate the seamless interplay of database design and application functionality, enriching both the user experience and technical robustness.

Week 4: Advanced SQL Queries

This week, you'll dive deep into advanced SQL queries to gather insights and detect anomalies in your gaming event management system. You'll focus on generating reports, analyzing player behaviors, and even implementing a basic fraud detection system.

Assignment:
For this week, you have a list of predefined questions related to the game's economy, quests, crafting, and players. Your task is to choose at least 15, but no more than 18, questions to answer using SQL queries. Your final grade will be calculated based on your best 15 queries.

Important Points:

- Query Selection: Out of the provided list of questions, choose at least 15 that you want to answer. You may choose up to a maximum of 18 queries to submit.
- Query Documentation: When you submit your queries, make sure to copy-paste the description of each question next to your SQL statements. This will help us understand what each query is intended to address.
- Database Modification: If you find that your existing database schema lacks any columns necessary to answer a particular question, you are allowed to modify the database. Add the necessary columns and populate them with data as needed. You do not have to provide an updated ERD or schema for these changes.

The questions:

- Player Behavior Analysis: Identify the five most commonly used combinations of first, second, and third items in player inventories during quests. Utilize CTEs and subqueries to extract this data.
- Chat Metrics: Calculate the average number of messages per minute sent in each chat room between 8 PM and 9 PM.
- Fraud Detection:
    - Rapid Item Acquisition: Identify any player who has acquired items with a total value exceeding a certain threshold (e.g., 1000 gold) in less than 24 hours.
    - Frequent Guild Changes: Flag players who have joined and left more than 'n' guilds within a 7-day period.
    - High Frequency Quest Completions: Detect players who have completed quests at an unusually fast rate, completing more than 'x' quests within a 24-hour period.
    - Large Money Transfers: Highlight transactions where large sums of in-game currency are transferred between accounts that are not in the same guild or have not interacted before.
    - Player Inactivity with High Transactions: Point out accounts that have been inactive for more than 'y' days but have received a significant number of high-value items or in-game currency in the meantime.
- Economy
    - Inflation Analysis: Track the average value of traded items over time to detect if the in-game economy is experiencing inflation or deflation.
    - Top Traded Items: Identify which items are most commonly traded or sold in-game, providing insight into what is considered valuable by the community.

    - High-Value Transactions: Find all transactions involving the trade or sale of high-value items and analyze the common characteristics among them.
    - Supply and Demand: Analyze the number of players seeking certain types of items versus the number of those items available for trade or sale. This can help to identify items that are in high demand but low supply, possibly commanding a higher market price.
    - Merchant Analysis: Identify players who are predominantly involved in trading activities rather than quests or combat. This could be a player strategy, or an account meant for in-game trading.
    - Market Hotspots: Find locations in the game where the most transactions or trades occur. This could indicate an in-game market or a hotspot for trading.
    - Currency Sink: Determine what in-game activities or transactions are the largest "sinks" for in-game currency (i.e., where currency is spent but not returned to the game economy).
    - Rare Item Ownership: List players who own the rarest items in the game, possibly indicating either advanced game expertise or involvement in trading networks.
    - Economic Imbalance: Detect players who have a disproportionate amount of in-game currency compared to their activity or level, which could be a sign of economy-unbalancing behavior.
    - Trade Partners: Identify frequent trade partnerships between players. Are the same players often involved in high-value trades with each other?
- Quests:
    - Trending Quests: List the top 3 quests that have shown the most significant percentage increase in player participation in the last month. Use window functions to rank them.
    - High Dropout Rates: Find quests with a high start but low completion rate, which might indicate they are too difficult or not engaging.
    - Quest Completions Over Time: Track the rate at which quests are being completed over time, possibly to identify if certain events or updates make them more or less popular.
    - Quest Rewards Analysis: Determine the average value of rewards from different quests to see which ones are most economically beneficial for players.
    - Quest Difficulty vs. Player Level: Analyze the levels of players who complete quests of varying difficulties to ensure that quests are well-matched to player ability.
    - Frequent Quest Partners: Identify pairs or groups of players who frequently complete quests together.
    - Time-to-Completion: Calculate the average time it takes for players to complete different quests. This could help in balancing or redesigning them.
- Crafting
    - Resource Utilization: Find out which crafting resources are most commonly used, indicating their value and possibly affecting their market price.
    - Crafting Skill Level vs. Player Level: Examine if there is a correlation between a player's overall level and their crafting skill levels.
    - Crafting Profitability: Compare the market value of crafted items to the cost of the raw materials to find the most profitable crafts.

- Players
    - Player Retention: Analyze the average session length and frequency for each player to identify patterns in player retention.
    - Level Distribution: Examine the distribution of player levels to understand the balance between newcomers, mid-level, and high-level players.
    - Player-to-Player Transactions: Investigate the types and frequencies of trades or transactions between players to uncover the most traded items or gold amounts.
    - Popular Locations: Find out which game zones or areas are most frequented by players, which could help in planning future updates or events.
    - Skill Mastery: Identify players who have maxed out particular skills, whether they be combat, crafting, or any other in-game skills.
    - Guild Affiliation: Examine the distribution of players in various guilds to find out which guilds are the most popular or exclusive.
    - Player Reporting: Track how often players report issues, bugs, or other players, to understand community health.
    - Time of Play: Find out what times of day are the most popular for playing, which could help in scheduling in-game events or maintenance.
    - Social Connectivity: Query friendships, team-ups, or any other social interactions to see how socially connected your player base is.
    - PvP Rankings: List players based on their performances in player-versus-player battles, potentially subdividing by various metrics like number of wins, kill-to-death ratio, etc.

Bonus: Custom Query

- **Worth 10 points:** Develop your own custom query to detect potential fraudulent or suspicious behavior not covered above. Points will be awarded based on the relevance and complexity of the query, at the teacher's discretion.

Deliverables:

- Your queries and their results should preferably be in a Jupyter notebook. Export the notebook to PDF or HTML format, ensuring that the query results are visible in the exported file. Note that .ipynb files will not be accepted. Alternatively, you may create a document containing the queries and their result sets.
  Alternatively, you can create a document containing the SQL queries and their result sets.
  !!! Make sure the results of your queries, and the question per query are in the document, or we cannot grade them !!!!

Rubric:

- Complexity of SQL queries: 2 points per query (max 30)
- Correctness of SQL queries: 2 points per query (max 30)
- **Bonus custom query:** 10 points (Teacher's discretion based on relevance and complexity)

By the end of this week, you should be adept at crafting complex SQL queries that not only answer key questions about your gaming system but also shed light on potentially suspicious behaviors.

Week 5: NoSQL Databases for Real-Time Game Functionality

As your gaming event management system expands, you are faced with new challenges that require more complex data handling. For instance, you want to provide real-time analytics to gamers about their social interactions and quest participation, all while maintaining high performance and low latency. Traditional SQL databases may not be the optimal solution for handling this degree of complexity, especially when relationships are multi-layered and queries become increasingly intricate. This is where NoSQL databases come in handy.

Why Schema-less Approach?
One of the significant benefits of NoSQL databases is their schema-less architecture. In the context of a gaming event management system, this becomes particularly beneficial when dealing with real-time analytics and intricate quests. Quests may have attributes that are not known at design time, and new features may be added without requiring a complete schema redesign. For example, a quest may suddenly introduce magic spells with new rules or new types of equipment, and these can be easily accommodated in a NoSQL database. This flexibility also extends to real-time friendships, where attributes like "shared quests" or "mutual achievements" could be introduced later.

Document-based or Graph-based NoSQL databases can provide flexible, scalable, and high-performance solutions for these specific challenges. Document databases excel in handling hierarchical, semi-structured data, while Graph databases are designed for complex relationship handling.

Problem Statement:
You are tasked with improving the real-time analytics of your gaming event management system. Specifically, you aim to:

- Track and analyze real-time friendships between players.
- Record and query intricate quest participation metrics (e.g., the quests a player has participated in, the items collected during those quests, etc.)
- The conventional relational database system you have been using is no longer efficient for these types of analytics. You need a more agile and performance-optimized solution.

Objectives:
- Understand the benefits of NoSQL databases in handling complex, interconnected data and real-time functionality.
- Design a NoSQL schema that is better suited for certain gaming features.
- Migrate SQL database to NoSQL database.

Assignment:
- Choose either a Document-based database (like MongoDB) or a Graph-based database (like Neo4j).
- Install and configure the database. Note down the steps, as you may need to submit them with your project.

- Design a NoSQL schema to handle intricate quest participation records and real-time friendships.

Instructions for Schema Design:

For a Document-based Database (e.g., MongoDB):

Quest Participation Records:
Create a collection named Quests. Each document should contain:
- Quest ID
- Quest name
- Description
- Minimum and maximum player level requirements
- A nested document for QuestLocation with attributes like coordinates or zone

An array of participating players, which includes:
- Player ID
- Items collected (as an array of item IDs and quantities)
- Points scored
- Time taken to complete the quest

An array of NPCs involved in the quest, each containing:
- Dialogue (as an array of dialogue lines per NPC)
- The name and location of the NPC

Real-Time Friendships:
Create a collection named Players.
Each document should contain:
- Player ID
- Username
- Avatar Image

An array of friends, each being a nested document containing:
- Friend's player ID
- The date when they became friends
- A list of shared quests
- A list of mutual enemies defeated

Graph-based Database (e.g., Neo4j):

Quest Participation Records:
Create nodes for each Quest, Player, and NPC. Create PARTICIPATED_IN edges from players to quests they participated in.

Annotate these edges with:
- Items collected
- Points scored
- Time taken to complete the quest
- Create ENCOUNTERED edges from players to NPCs they interacted with during quests. Annotate these edges with dialogue as an array of dialogue lines.

Real-Time Friendships:

- Create nodes for each Player and NPC.
- Create FRIENDS_WITH edges between players who are friends.
- Annotate these edges with:
- The date they became friends
- An array of shared quests
- A list of mutual enemies defeated

Create Sample Data and Queries:

Populate your Quests and Players collections or nodes with sample data:

- Add at least 5 quests, each with varying levels of complexity, rewards, and player level requirements.
- Include NPCs that players encounter during quests, each having various dialogue options.
- Add at least 10 players, each having various inventory items and at least 2 friends.

Queries:

Document-based Database:

- Find all quests suitable for a player of a specific level.
- List all players who became friends in a specific month.
- Retrieve all quests where a specific NPC was encountered, and list the dialogue options.
- Find all players who collected a specific item across all quests.
- List quests completed within a certain time range.
- Retrieve the top 5 players with the highest points scored in a particular quest.
- Find quests where fewer than X players participated.
- Identify players who have not participated in any quest for a given time period.

Graph-based Database:

- Find the shortest path of friendship between two specific players, including shared quests.
- Find all NPCs encountered by players who participated in a particular quest, and list the dialogue options.
- Retrieve the network of players who defeated a specific enemy.
- Identify players who are connected through friendship but have never participated in the same quest.
- Find all quests where a specific player and their friends have participated.
- Locate the closest mutual friend between two specific players, based on shared quests or mutual enemies.
- Identify the most popular NPC based on the number of players who have interacted with them.
- Retrieve all quests that involve a particular item and find out which players have collected it.

Deliverables:

- Your populated NoSQL database. Export your collections or graph into a format that can be easily imported for review (e.g., JSON export for Document-based databases, graph export for Graph-based databases). You only have to hand in the queries for 1 type of database, handing in both will not result in a higher grade.
- A Jupyter notebook containing the queries you've written and the results. Export the notebook to PDF or HTML, making sure that query results are visible. .ipynb files are not accepted.

Rubric:

- Choice and configuration of NoSQL database: 10 points
- Correctness and complexity of NoSQL schema: 10 points
- Creation of appropriate sample data: 20 points
- Correctness and complexity of queries: 20 points

By completing this assignment, you will gain hands-on experience with NoSQL databases, understanding how they can be better suited for complex and real-time functionalities in your gaming event management system.