תרגיל סיכום

קורס: תכנון וניהול מסדי נתונים NOSQL
6168

מרצאות:
ד"ר גרדוביץ נועה
גב' כהן סיון

מגישות:
אילנה גמיל -206411563
יובל יקר - 204896542
אורי שי - 312421852

תוכן עניינים

שאלה 1:

<u>א. שאלת המחקר</u>

מהי ההשפעה של רמת הקושי של משימות במשחקי MMORPG על שיעור נטישה של שחקנים חדשים ? המטרה המרכזית: להבין כיצד רמת הקושי של משימות ההתחלה משפיעה על החלטות השחקנים בהמשך ?

MMORPG - Massively Multiplayer Online Role-Playing Game
ובעברית "משחק תפקידים המוני ברשת", זהו סוג של משחק מחשב מקוון שבו אלפי שחקנים יכולים לשחק יחד בעולם וירטואלי ענק ומורכב.

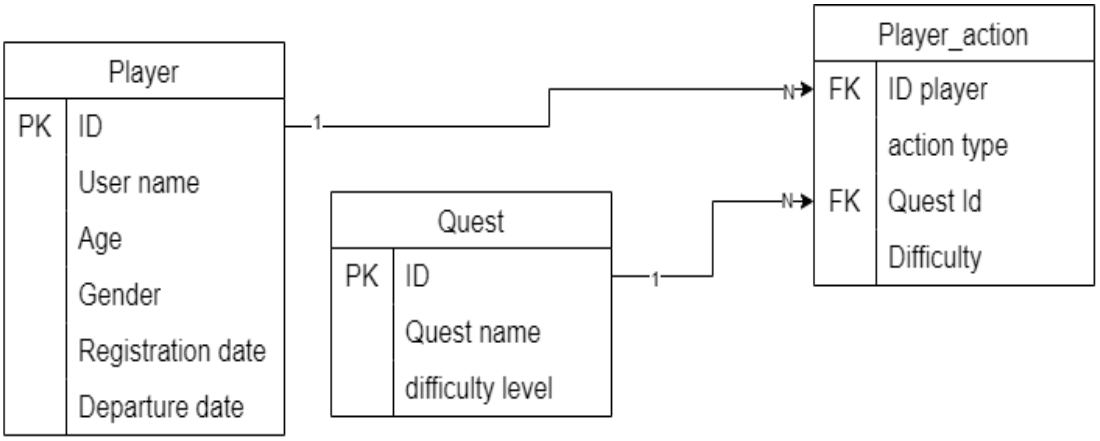
במשחקים אלו, שחקנים חדשים לעיתים קרובות מתמודדים עם עקומת למידה אקספוננציאלית ולא מעט משימות ברמת קושי גבוהה כבר בתחילת המשחק.
במחקר זה נרצה לבדוק כיצד רמת הקושי של המשימות הללו ישפיעו על החלטת השחקנים האם להמשיך לשחק או לעזוב.

ב. איך ישמרו הנתונים בבסיסי הנתונים השונים -

ישנם שלושה אובייקטים עיקריים:

- **שחקנים:** האובייקט יכיל מידע דמוגרפי בסיסי, זמן התחלת משחק, זמן עזיבה (במידה ורלוונטי).
- **משימות:** פרטי המשימה, רמת קושי (תיוג המשימה כקלה, בינונית, קשה), זמן ממוצע לסיום המשימה ועוד פרמטרים המתארים את המשימה.
- **פעולת שחקן (סטטוס התקדמות התקדמות):** משימות שהושלמו, משימות שננטשו, זמן השקעה במשחק.

מודל בסיס נתונים - RDBMS



- **טבלאות:**
  - **Player**
  - **Quest**
  - **Player_action**

- **מפתחות (Keys):**

מפתח ראשי (Primary Key):

- **player_id בטבלת השחקנים**: מזהה ייחודי עבור כל שחקן.
- **quest_id בטבלת המשימות**: מזהה ייחודי לכל משימה.

מפתח זר (Foreign Key):

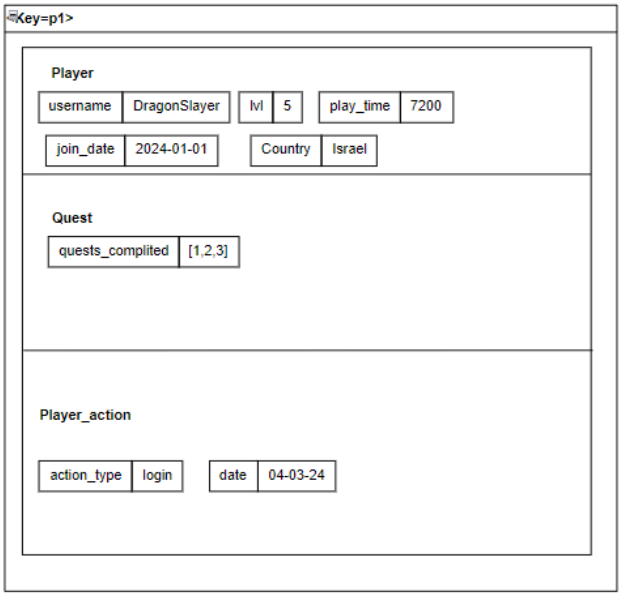
- **player_id בטבלת הפעולות**: מקשר את הפעולה לשחקן שביצע אותה.
- **quest_id בטבלת הפעולות**: מקשר את הפעולה למשימה שהושלמה.

- **יתרונות:**
  - גמישות ביכולות חיפושים ובניתוחים.
  - שלמות נתונים גבוהה בזכות אילוצי שלמות.
  - תמיכה רחבה בכלי ניתוח נתונים.

- **חסרונות:**
  - עלול להיות מורכב לתחזוקה עבור נתונים לא מובנים.
  - רמת הביצועים עשויה להיפגע עבור כמות גדולה של נתונים.

Key-Value DB

- **מפתחות:** ID שחקן (מפתח), מילון של כל הנתונים הרלוונטיים (ערך).
- **יתרונות:**
  - פשוט וקל לשימוש.
  - ביצועים גבוהים לקריאות פשוטות.
- **חסרונות:**
  - מוגבלות בחיפושים מורכבים.
  - קשה לשמור על שלמות נתונים.



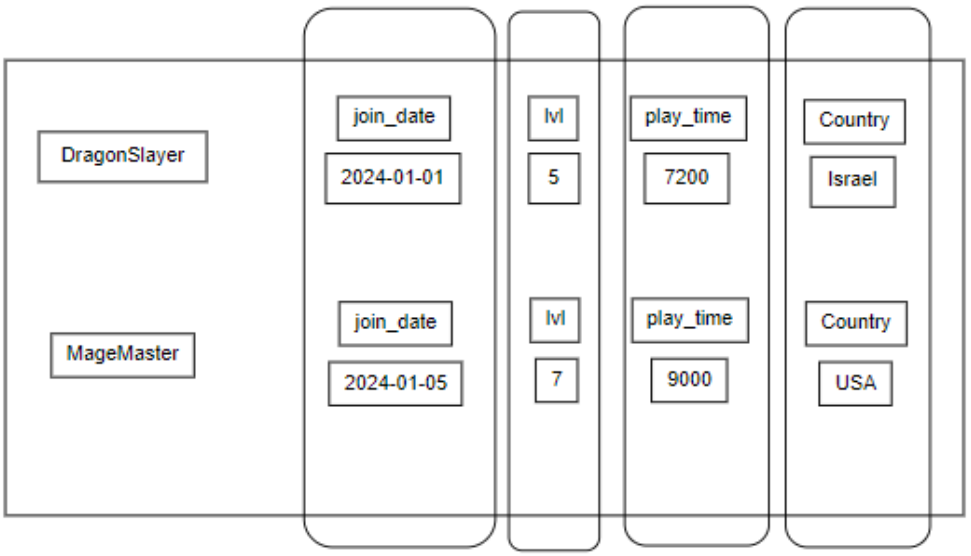
Column DB

- **טבלאות:**
  - **שחקנים:** ID, שם משתמש, גיל, מין, תאריך הרשמה, תאריך עזיבה.
  - **משימות:** ID, שם משימה, רמת קושי, סדר ביצוע.
  - **התקדמות:** ID שחקן, ID משימה, זמן התחלה, זמן סיום, האם הושלמה.

- **עמודות:** כל עמוד מייצג סוג נתונים מסוים (למשל, גיל, זמן התחלה).

- **יתרונות:**
  - ביצועים גבוהים לקריאות על עמודות ספציפיות.
  - דחיסה יעילה של נתונים.

- **חסרונות:**
  - פחות גמישות בחיפושים מורכבים.



Document DB

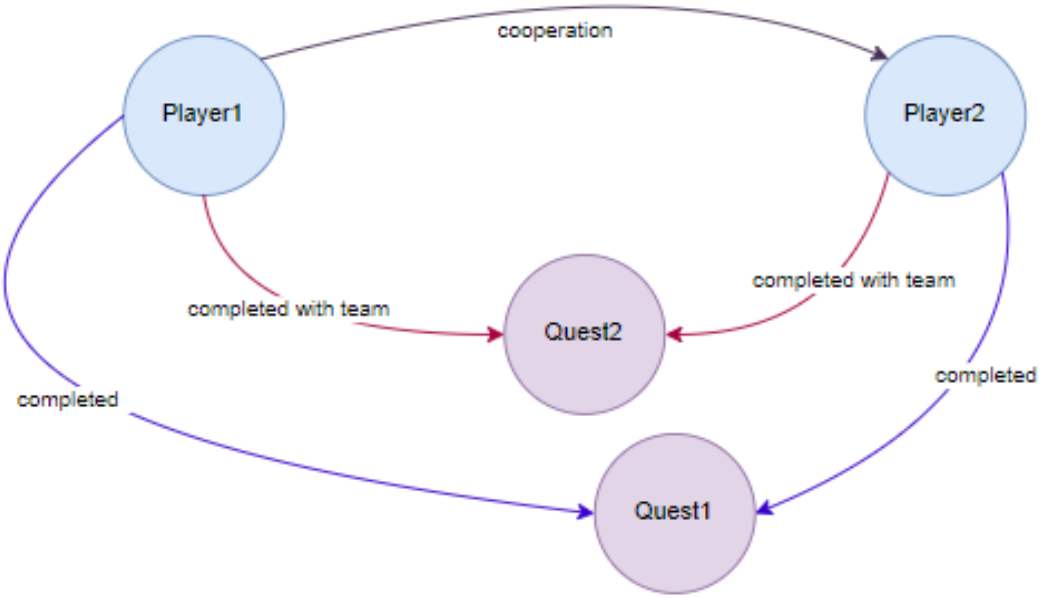
- **מסמכים:**
  - **Player**
  - **Quest**
  - **player_action**
- **יתרונות:**
  - גמישות רבה במבנה הנתונים.
  - קל לאחסון נתונים לא מובנים.
- **חסרונות:**
  - קשה יותר לבצע חיפושים מורכבים.
  - פחות מתאים לניתוחים סטטיסטיים מורכבים.
-

```
{
  "player_id": "p1",
  "username": "DragonSlayer",
  "join_date": "2024-01-01",
  "level": 5,
  "total_play_time": 7200,
  "actions": [
    {
      "action_type": "Quest Completion",
      "quest_id": "q106",
      "difficulty": 4,
      "completion_date": "2024-01-20"
    }
  ]
}
```

```
{
  "quest_id": "q105",
  "quest_name": "Infiltrate the Fortress",
  "difficulty": 3,
  "average_completion_time": 160,
  "failure_rate": 0.15,
  "rewards": ["XP", "Item"]
}
```

Graph DB

- **גרפים:**
  - **צמתים:** שחקנים, משימות.
  - **קשתות:** מייצגות קשרים בין שחקנים למשימות (למשל, "השלים משימה").
- **יתרונות:**
  - מצוין למיפוי קשרים בין ישויות.
  - ביצועים גבוהים לחיפושים על גרפים.
- **חסרונות:**
  - מורכב יותר לתכנון ומימוש.
  - פחות מתאים לאחסון נתונים מספריים.
  -



ג. בחירת בסיס נתונים

למטרת המחקר אנו נבחר בתצורת **Document DB**, מהסיבות הבאות:

1. **גמישות במבנה נתונים** :

בתעשיית המשחקים, במיוחד ב-MMORPGs, הנתונים שמתקבלים מגוונים מאוד – מפרופילים אישיים של שחקנים, דרך רכישות בתוך המשחק ועד הישגים ואינטראקציות חברתיות.
בסיסי נתונים מהסוג שבחרנו, הם פתרון נפוץ, מכיוון שאין להם צורך בסכמות קבועות כמו בבסיסי נתונים מסורתיים.
מסדי נתונים מבוססי מסמכים מאפשרים למפתחי המשחקים לאחסן נתונים מגוונים ללא צורך בהגדרה מראש של מבנה קשיח.
זה חשוב במיוחד במחקר שבו סוגי נתונים משתנים כל הזמן, כמו נתוני התנהגות שחקנים, שימוש בפריטים במשחק או ביצועים במשימות.
מפתחים יכולים לשאול שאלות מחקר ולנתח נתונים באופן גמיש יותר מבלי לבצע שינויים משמעותיים במבנה המסד לצורכי השאלות.

2. **התאמה לנתונים "חצי-מובנים"** :

MMORPGs מייצרים נתונים חצי מובנים בזמן שהשחקנים מתקדמים בתרחישים שונים, מבצעי פעולות מגוונות ומקיימים אינטראקציות עם שחקנים אחרים.
חברות רבות מעדיפות להשתמש במסדי נתונים מהסוג שבחרנו מכיוון שהם מאפשרים טיפול יעיל בנתונים הללו בלי הצורך בהגדרות סכימה נוקשות מראש.
מחקרים רבים בתעשיית המשחקים מתמקדים בנתונים חצי-מובנים, כמו אינטראקציות חברתיות, הודעות בצ'אט או התנהגות בזמן אמת במשחק.
בסיס נתונים מסוג זה אידיאלי לסוגי נתונים כאלה ומאפשרים ניתוח מהיר של תבניות, מגמות ודפוסי משחק לא סטנדרטיים.

3. **Scalability (יכולת גידול)**:

יכולות גידול זו דרישה אינהרנטית במשחקים מהסוג שתואר, שבהם המוני שחקנים עשויים להיות פעילים בו זמנית בשעות העומס.
מסדי נתונים מהסוג שבחרנו מצטיינים ביכולות גדילה אופקית, מה שהופך אותם לבחירה רלוונטית עבור משחקים שצריכים להתמודד עם גידול מתמיד בבסיס המשתמשים ועם נפחי נתונים גדולים, מבלי לפגוע בביצועים.
שאלות מחקר בתעשיית המשחקים דורשות לעיתים קרובות ניתוחים על כמויות עצומות של נתונים, במיוחד במשחקים מרובי משתתפים שבהם עשרות או מאות מיליוני שחקנים פעילים.
מסדי נתונים מבוססי מסמכים תומכים במדרגיות אופקית, כך שניתן להוסיף בקלות שרתים נוספים כדי לנהל את נפח הנתונים העצום, מה שמאפשר למפתחים לבצע שאילתות ולנתח נתונים בקנה מידה גדול ללא פגיעה בביצועים.

4. **ביצועים בניתוח נתונים ויכולות דינאמיות**:

במחקרים על משחקים, יש צורך לייצר שאילתות דינמיות כדי לנתח דפוסי משחק שונים בזמן אמת, כמו התנהגות שחקנים במהלך אירועים מיוחדים במשחק או תגובות לעדכונים ותכונות חדשות.
מסדי נתונים אלו מאפשרים תגובה מהירה לביצוע מחקרים כאלה תוך זמן קצר, מה שנותן למפתחים תובנות עומק בזמן אפס.

5. **קלות שימוש ופיתוח מהיר**:

עולם פיתוח המשחקים הוא דינאמי ומהיר, תוכן חדש ועדכונים לשחקנים הם משהו שהיצרנים מפיצים באופן תדיר, בסיס נתונים כפי שבחרנו מציע יתרון משמעותי בכך שהוא מפחית את הצורך בשינויים מורכבים של סכמות נוקשות.
זה מאפשר פיתוח מהיר יותר והוצאת עדכונים ותכונות חדשות לשוק בזמן קצר יותר.
מפתחים בחברות כמו Riot Games, המוכרת בזכות League of Legends, נהנים מהפשטות הזו כאשר הם נדרשים לפרוס עדכונים תכופים ולטפל בנתוני שחקנים משתנים בצורה דינמית.

6. **יכולת אינטגרציה גבוהה עם כלי אנליזה שונים**:

מסדי נתונים מבוססי מסמכים משתלבים בצורה טובה עם כלים אנליטיים שונים, דבר שמקל על המפתחים לשלוף נתונים לצורך ניתוחים מתקדמים, מחקר התנהגותי ותחזיות.
שילוב זה מאפשר למפתחים לגשת לנתונים בקלות ולהפיק מהם תובנות מחקריות שימושיות לפיתוח עתידי.

7. **פשטות הפיתוח והניהול:**

MongoDB פשוטה יותר לשימוש בהשוואה ל-Graph DB כאשר המיקוד הוא על מסמכים ולא על ניתוח קשרים. היא מאפשרת שליפה מהירה של נתונים עם שאילתות גמישות ואינדקסים שיכולים לייעל את השאילתות בצורה משמעותית.

לסיכום,
MongoDB נבחרה כי היא מספקת פתרון יעיל ונוח לאחסון ושליפה של נתונים בביצועים גבוהים, הקשורים לשחקנים באופן אינדיבידואלי, בלי צורך במיקוד על קשרים מסובכים בין שחקנים, אלא אנו מתמקדים במעקב אחר הביצועים והפעולות של כל שחקן בנפרד.

שאלה 2:

א. תיאור המערכת

בחרנו במערכת משחק מחשב מסוג
MMORPG (Massively Multiplayer Online Role-Playing Game),
משחק שבו מספר רב של שחקנים יכולים לשחק יחד בעולם וירטואלי.
המערכת מאפשרת מעקב אחר התנהגות השחקנים ומספקת תובנות עמוקות ומשמעותיות,
המסייעת בשיפור חווית המשתמש.

שאלות למחקר

1. מי הם השחקנים הנמצאים בשלב 3 ומעלה?
2. מי הוא השחקן המתקדם ביותר, לפי מדינה?
3. כמה קבוצות שחקנים ממדינות שונות עובדות יחד על משימות משותפות וכמה מהן מאותה מדינה?

ב. סכמת שמירת הנתונים

יש לנו 4 אובייקטים:

1. player

- אוסף המייצג את כל השחקנים במשחק.
- **שדות עיקריים**:
  - player_id: מזהה ייחודי לכל שחקן.
  - username: שם המשתמש של השחקן.
  - join_date: תאריך הצטרפות השחקן למשחק.
  - level: רמת השחקן במשחק.
  - total_play_time: זמן המשחק הכולל של השחקן (ב-seconds).
  - country: מדינה שבה השחקן נמצא.
-

```
_id: ObjectId('66d4a311e44e7d6218cae585')
player_id : "p1"
username : "DragonSlayer"
join_date : "2024-01-01"
level : 5
total_play_time : 7200
country : "Israel"
```

2. player_action

- אוסף המייצג את פעולות השחקן במשחק.
- **שדות עיקריים**:
  - player_id: מזהה השחקן שביצע את הפעולה.
  - action_type: סוג הפעולה (למשל, "Quest Completion", "Login", "Logout").
  - quest_id: מזהה המשימה אם מדובר בפעולה שקשורה במשימה.
  - difficulty: רמת הקושי של המשימה (כאשר action_type הוא "Quest Completion").
  - completion_date: תאריך ביצוע הפעולה (כאשר action_type הוא "Quest Completion").
  - login_date, logout_date: תאריכים בהם השחקן נכנס או יצא מהמשחק.
-

```
_id: ObjectId('66d497c9a8ee0febee40f148')
player_id : "p1"
action_type : "Quest Completion"
quest_id : "q101"
difficulty : 3
completion_date : "2024-01-10"
```

3. quest

- אוסף המייצג את המשימות הזמינות במשחק.
- **שדות עיקריים**:
  - quest_id: מזהה ייחודי לכל משימה.
  - quest_name: שם המשימה.
  - difficulty: רמת הקושי של המשימה.
  - average_completion_time: זמן ממוצע לסיום המשימה (ב-seconds).
  - failure_rate: שיעור כישלון בביצוע המשימה.
  - rewards: פרסים שמקבל השחקן לאחר סיום המשימה (כמו "Gold", "XP", "Item").
-

```
_id: ObjectId('66d497daa8ee0febee40f154')
quest_id : "q101"
quest_name : "Clear the Dungeon"
difficulty : 3
average_completion_time : 150
failure_rate : 0.14
▾ rewards : Array (2)
  0: "Gold"
  1: "XP"
```

4. shared_quests

- אוסף המייצג את הקשרים בין משימות לשחקנים שמשתפים פעולה בהן.
- **שדות עיקריים**:
  - quest_id: מזהה המשימה.
  - player_ids: רשימת מזהי השחקנים המעורבים במשימה זו.
-

```
_id: ObjectId('66d4a30be44e7d6218cae57c')
quest_id : "q101"
▾ player_ids : Array (3)
    0: "p1"
    1: "p2"
    2: "p3"
```

כל אוסף מייצג חלק מהמידע המנוהל במשחק ומספק מידע קריטי להבנת התנהגות השחקנים, ניהול המשימות וניתוח הביצועים במשחק.

ג. דרכי גדילה

בסיסי נתונים ב- MongoDB מאפשרים מספר אפשרויות גדילה, מתאימות לצרכים השונים של המפתחים:

Vertical Scaling (הרחבה אנכי)

הרחבה אנכית מתייחסת לשדרוג היכולות של שרת יחיד על ידי הוספת משאבים כמו זיכרון RAM, כוח עיבוד או נפח אחסון.
הגדלת המשאבים מאפשרת לשרת להתמודד עם עומסים גדולים יותר ולבצע יותר פעולות במקביל.
פתרון זה לרוב פשוט ליישום ולא מצריך שינויים משמעותיים במבנה המערכת.

החיסרון בתצורה זו היא המוגבלות ביכולת ההרחבה, מכיוון שיש גבול פיזי למשאבים שניתן להוסיף ולעיתים עלות שדרוג השרתים עשויה להיות גבוהה באופן קיצוני משיטות אחרות.

Horizontal Scaling (הרחבה אופקי)

בהרחבה אופקית, מוסיפים שרתים נוספים למערכת במקום לשדרג ו"לחזק" את השרתים הקיימים.
הנתונים מחולקים בין השרתים, כך שכל שרת מטפל בחלק מהנתונים, מה שמאפשר פיזור עומסים ובאופן עקיף הגדלת יכולת העיבוד.

היתרון בשיטה היא הגמישות ויכולת הרחבה כמעט בלתי מוגבלת, מאפשר ניהול יעיל של כמויות נתונים גדולות במיוחד.

החסריון בשיטה זו שהיא דורשת תכנון מורכב יותר וניהול רחב יותר (לעיתים מסובך) של השרתים, מה שעלול להוביל לאתגרים באחידות ובעקביות של הנתונים.

Sharded Cluster (אשכול מחולק לשארדים)

בשיטה זו, הנתונים מחולקים למקטעים קטנים יותר שנקראים "שארדים".
כל שארד מאוחסן על שרת נפרד, כך שהמערכת יכולה להתרחב אופקית על ידי הוספת שארדים נוספים לפי הצורך.
כל שארד מתפקד כיחידת מסד נתונים עצמאית, אך כל יחדיו יוצרים מסד נתונים אחיד.

היתרונות בשיטה זו שהיא מאפשרת הרחבה כמעט בלתי מוגבלת של יכולות האחסון והביצועים, תוך שמירה על ניהול מרכזי ואחוד של הנתונים.
החיסרון שעלול להיות בשיטה זו הוא הניהול שהופך מורכב יותר, במיוחד בניהול החלוקה של הנתונים בין השארדים והבטחת עקביות הנתונים.

בחרנו בדרך הגדילה - sharded cluster

בסופו של דבר, אחרי שסקרנו את שלושת השיטות, הבנו ששיטת ה-Sharded Cluster מספקת לנו גמישות מרבית ויכולת להתרחב בקלות ככל שהמערכת גדלה, תוך שמירה על ביצועים גבוהים וניהול יעיל של הנתונים.
שיטה זו מאפשרת לנו להתמודד עם נפחי נתונים עצומים האופייניים למשחקי MMORPG ולהתאים את המערכת לעומסים משתנים.

ככל שמספר השחקנים והפעולות במשחק גדל, שיטת השארדים מאפשרת להוסיף שרתים נוספים בקלות, ובכך להתמודד עם דרישות הולכות וגדלות של אחסון וניתוח נתונים. זה מספק גמישות המאפשרת לנו להמשיך ולנהל את המשחק בצורה יעילה גם כשהתעבורה גדלה באופן משמעותי.

השיטה מאפשרת לחלק את העומס בין מספר שרתים, כך שכל שרת מתמקד בחלק מהנתונים בלבד.
פיזור זה תורם לשיפור הביצועים, ומבטיח זמן תגובה מהיר יותר לשחקנים, תוך שמירה על חוויית משחק רציפה.

למרות שהנתונים מפוזרים בין השארדים, MongoDB מסוגל לנהל את כולם באופן שמאפשר למפתחים ולשחקנים להרגיש כאילו הם עובדים עם מסד נתונים יחיד.
כך אנו נהנים מביצועים מיטביים תוך שמירה על ניהול פשוט יחסית של הנתונים, מבלי לוותר על הגמישות הנדרשת.

MongoDB מספק כלים לניהול עקבי בין השארדים, מה שמקטין את הסיכון לבעיות תאימות או אי-אחידות בעבודה עם הנתונים.
זהו יתרון משמעותי עבורנו כשאנו מבקשים לשמור על שלמות הנתונים בין כל השרתים, גם כשהמערכת מתרחבת ומתפתחת.

בשורה התחתונה מבחינתנו, שיטת השארדים היא הבחירה הטבעית למחקר במשחקים בסדר גודל כזה, מכיוון שהיא משלבת גמישות, ביצועים גבוהים ויכולת להתרחב בהתאם לצרכים המשתנים.

חלוקת הנתונים בשיטת **Sharded Cluster** מתבצעת על פי "שארדים" (Shards), כאשר כל שארד מכיל חלק מהנתונים של מאגר המידע הכולל. זו השיטה הנפוצה במערכות הדורשות יכולת הרחבה (scalability) גבוהה, כמו מערכות MMORPG עם מספר גדול של שחקנים ואירועים.

דוגמה לחלוקה

נניח שאנחנו מחלקים את נתוני השחקנים והפעולות שלהם למספר שארדים על פי join_date.
החלוקה הזו מאפשרת לכל שארד להתמקד בקבוצת שחקנים מסוימת.

נניח שיש לנו שלושה שארדים, וכל אחד מטפל בטווח של תאריכי הצטרפות:

שארד 1: יכיל שחקנים שהצטרפו בין ה-1 לינואר 2022 ועד ה-31 בדצמבר 2022.
שארד 2: יכיל שחקנים שהצטרפו בין ה-1 בינואר 2023 ועד ה-31 בדצמבר 2023.
שארד 3: יכיל שחקנים שהצטרפו החל מה-1 בינואר 2024 ואילך.

כיצד זה מייעל את המערכת:

חלוקת עומס:
כל שארד מנהל טווח ספציפי של תאריכי הצטרפות (למשל שנה מסוימת), ולכן הוא מטפל רק בנתונים מאותו טווח. הדבר מפחית את העומס על כל שארד, ומונע מצב שבו שארד אחד נדרש לטפל בנתונים עצומים.

יכולת הרחבה (Scalability):
במקרה שמספר השחקנים גדל משמעותית, נוכל להוסיף שארדים חדשים בקלות. לדוגמה, אם שחקנים חדשים מצטרפים מעבר לשנת 2022, אפשר להוסיף שארד נוסף שיטפל בהם.
MongoDB תבצע rebalance באופן אוטומטי כדי לפזר את הנתונים בצורה מאוזנת בין השארדים.

שיפור ביצועים:
בדרך כלל, ש**חקנים שהתחילו באותו פרק זמן יהיו בשלבים צמודים** במשחק **ובעלי דפוסי משחק דומים.**
חלוקה לפי `join_date` תוכל להקל על יצירת קבוצות ושיתופי פעולה בין שחקנים מאותה תקופה.

חלוקה זו מייעלת את המעקב אחר שיתופי הפעולה במשחק ומספקת תמונה ברורה יותר לגבי האינטראקציות בין שחקנים, מאחר ושחקנים מאותו שארד סביר להניח שיהיו בעלי דפוסי פעולה דומים.

ד. מימוש המערכת

מימשנו את המערכת באמצעות MongoDB, מסד נתונים מסוג documentDB
כל השאילתות הנדרשות לפעולות המערכת, כולל שליפות העונות לשאלות המחקר, מתועדות ומצורפות במסמכים נפרדים.
כאמור, הבחירה ב-MongoDB מאפשרת לנו לבצע עיבוד מהיר של נתונים בזמן אמת, תוך שמירה על התאמה לאופי הדינמי והגמיש של המשחק.

1. שאילתה על שחקנים לפי רמה והצגת פרטים מסוימים

```
db.player.find(
    { level: { $gte: 3 } }, // all players with level 3 and above
    { _id: 0, player_id: 1, username: 1, level: 1 } // display
)
```

שאילתה זו מוצאת את כל השחקנים (players) עם רמת ניסיון (level) השווה ל-3 ומעלה, ומציגה רק את מספר מזהה השחקן (player_id), שם המשתמש (username), ורמת השחקן (level).

שלב 1: תנאי החיפוש (find)

- השאילתה מחפשת באוסף השחקנים (player) את כל המסמכים בהם השדה level גדול או שווה ל-3.
- **$gte**: אופרטור של MongoDB שמשמעותו **"גדול או שווה"**. זה שקול ל-=< ב-SQL.

שלב 2: בחירת שדות להצגה (projection)

- בשלב זה, אנו קובעים אילו שדות ייכללו בתוצאה ואילו לא.
- **_id: 0**: מבטלים את הצגת השדה _id שהוא ברירת המחדל של MongoDB. כברירת מחדל, _id מוצג אלא אם מבטלים אותו במפורש.
- **player_id: 1, username: 1, level: 1**: בוחרים להציג את השדות player_id, username, ו-level במסמכים שנמצאו.

אינדקסים שמייעלים את השאילתה

- { level: 1 } על אוסף player:
  - האינדקס הזה מאפשר למנוע לבצע חיפוש מהיר לפי רמת השחקנים. כאשר יש אינדקס על level, MongoDB יכול לגשת ישירות לשחקנים שרמתם שווה או גבוהה מ-3 מבלי לסרוק את כל האוסף.

2. אגירה וקיבוץ שחקנים לפי מדינה וחישוב השחקן עם הרמה הגבוהה ביותר בכל מדינה

```
db.player.aggregate([
    {
        $group: { // quest_id לפי documents מאגד
            _id: "$country",  // .כל קבוצה היא מדינה - country-קיבוץ לפי שדה ה
            highest_level_player: { $first: "$$ROOT" },// .שמירה על המסמך הראשון בכל קבוצה - זה יהיה המסמך המייצג של המדינה
            max_level: { $max: "$level" }  // .חישוב הערך המקסימלי של רמת השחקן באותה מדינה
        }
    },
    {
        $project: {
            _id: 0,  // .לא להציג את שדה ה-_id בתוצאה
            country: "$_id",  // .group$-מקבוצת ה-_id שהתבסס על ה-country הצגת שדה ה
            username: "$highest_level_player.username",  // .הצגת שם המשתמש של השחקן עם הרמה הגבוהה ביותר
            level: "$max_level"  // .group$-הצגת הרמה המקסימלית כפי שנחוש ב
        }
    }
])
```

השאילתה לוקחת את כל המסמכים באוסף ה-player, מחלקת אותם לקבוצות לפי המדינות.
עבור כל מדינה, היא מוצאת את הרמה הגבוהה ביותר בקרב השחקנים באותה מדינה ומציגה את השחקן הראשון בכל מדינה.
בסוף מתקבלות התוצאות: שם המדינה, שם המשתמש של השחקן המוביל באותה מדינה, והרמה הגבוהה ביותר באותה מדינה.

שלב 1: group$

- **"id: "$country_**: קיבוץ כל המסמכים באוסף השחקנים לפי שדה המדינה שלהם (country).
כלומר, כל מדינה מהווה קבוצה נפרדת.
- **"highest_level_player: { $first: "$$ROOT } {**: בכל קבוצה שנוצרת לפי מדינה, נשמר המסמך הראשון של כל קבוצה כולה (שימו לב שאין כאן מיון ולכן המסמך הראשון יהיה סתם אקראי מתוך הקבוצה).
- **"max_level: { $max: "$level } {**: בכל קבוצה, מחושב הערך המקסימלי של שדה הרמה (level),
כלומר רמת השחקן הגבוהה ביותר באותה מדינה.

שלב 2: project$

- **id: 0_**: הסתרת השדה _id מהתוצאה הסופית (שדה זה כולל את ערך המדינה אחרי ה-group$).
- **"country: "$_id**: הצגת השדה country, כאשר הוא נלקח מהערך של ה-_id מהשלב הקודם (שהוא ערך המדינה).
- **"username: "$highest_level_player.username**: הצגת שם המשתמש של השחקן המוביל (הראשון בכל קבוצה, כפי שנשמר ב-highest_level_player).
- **"level: "$max_level**: הצגת רמת השחקן הגבוהה ביותר שנמצאה בקבוצה (כפי שחושב ב-max$ בשלב הקודם)

אינדקסים שמייעלים את השאילתה

- { country: 1 } על אוסף player:

- האינדקס הזה היה משפר את הביצועים על ידי סידור שחקנים לפי מדינה ובכך יאפשר לקבוצת ה-$group לאסוף את כל השחקנים במדינה מסוימת במהירות. אם נבצע חיפוש או קיבוץ לפי country, האינדקס יפחית את הזמן הנדרש לביצוע הקיבוץ.

- { level: 1 } על אוסף player:
  - האינדקס על level עוזר במציאת הרמה הגבוהה ביותר בכל מדינה בשלב ה-$group שבו מתבצע חישוב ה-$max

3. שאילתה על משימות משותפות ואיתור מדינות השחקנים

```
db.shared_quests.aggregate([
    { $unwind: "$player_ids" }, // פותח את המערך של 'player_ids', כך שכל ערך במערך הופך לדוקומנט נפרד, מה שמאפשר עיבוד נפרד של כל שחקן במשימה.

    {
        $lookup: {
            from: "player", // מציין את האוסף בו נבצע את החיפוש.
            localField: "player_ids", // השדה באוסף הנוכחי שישמש לצימוד.
            foreignField: "player_id", // השדה באוסף המטרה שלעומתו יתבצע הצימוד.
            as: "player_info" // השם של השדה שיכיל את התוצאות של החיפוש.
        }
    },

    { $unwind: "$player_info" }, // פותח את המערך שנוצר מה-$lookup כדי להמשיך ולעבד כל שחקן כדוקומנט נפרד.

    {
        $group: {
            _id: "$quest_id", // מקבץ את הנתונים לפי מזהה המשימה.
            countries: { $addToSet: "$player_info.country" } // מוסיף לקבוצה ייחודית של מדינות מהשחקנים במשימה זו.
        }
    },

    {
        $group: {
            _id: null, // קבוצה גלובלית לכל הדוקומנטים.
            total_shared_quest_combinations_different_county: { $sum: 1 }, // סופר את כל המשימות המשותפות ללא תלות במדינה.
            shared_quest_combinations_with_same_country: {
                $sum: {
                    $cond: [{ $eq: [{ $size: "$countries" }, 1] }, 1, 0] // בודק אם כל השחקנים מאותה משימה הם מאותה מדינה, ואם כן מחזיר 1, אחרת 0.
                }
            }
        }
    },

    {
        $project: {
            _id: 0, // מסתיר את שדה ה-_id מהתוצאה הסופית.
            total_shared_quest_combinations_different_county: 1, // מציג את מספר המשימות המשותפות עם שחקנים ממדינות שונות.
            shared_quest_combinations_with_same_country: 1 // מציג את מספר המשימות המשותפות עם שחקנים מאותה מדינה.
        }
    }
```

השאילתה פותחת את מערך השחקנים במשימות המשותפות כדי לנתח כל שחקן בנפרד.
היא מצרפת מידע על המדינה של כל שחקן, ומקבצת את הנתונים לפי משימות.
לאחר מכן מחשבים כמה משימות כוללות שחקנים ממדינות שונות וכמה משימות כוללות שחקנים מאותה מדינה.
בסוף התוצאה מספקת מספר כולל של משימות עבור שתי הקטגוריות.

שלב 1: $unwind

- פותח את המערך של השחקנים (player_ids) כדי לטפל בכל שחקן במשימה כדוקומנט נפרד.
- כלומר, אם יש משימה עם מספר שחקנים, כל שחקן ייחשב לדוקומנט נפרד בשאילתה.

שלב 2: $lookup

מבצע צימוד (join) בין אוסף המשימות (shared_quests) לבין אוסף השחקנים (player).

- **localField**: השדה באוסף הנוכחי (shared_quests) המשמש לחיבור עם השדה באוסף היעד (player).
- **foreignField**: השדה באוסף player (במקרה זה player_id) לפיו מתבצע החיבור.
- **as**: זהו השם של השדה שיכיל את תוצאות ה-lookup (במקרה זה player_info).

שלב 3: $unwind

- אחרי חיפוש השחקנים, יתקבל מערך בשם player_info עבור כל שחקן. שלב זה פותח את המערך ומאפשר להמשיך לעבד כל שחקן כדוקומנט נפרד.

שלב 4: $group

קיבוץ לפי מזהה המשימה (quest_id): כל השחקנים באותה משימה מקובצים יחד תחת אותו מזהה משימה.

- **$addToSet: "$player_info.country"**: לכל משימה, נאספות המדינות של השחקנים לתוך קבוצה ייחודית, כך שאין כפילויות במדינות בתוך אותה משימה.

שלב 5: $group נוסף

קיבוץ גלובלי (ללא _id): התוצאות מקובצות יחד ללא חלוקה נוספת, כלומר השאילתה עוסקת במספר כולל של משימות.

- **total_shared_quest_combinations_different_county**: סופר את מספר המשימות המשותפות, ללא קשר לשאלה אם השחקנים מאותה מדינה או לא.
- **shared_quest_combinations_with_same_country**: סופר את המשימות שבהן כל השחקנים מגיעים מאותה מדינה.
  - זה מתבצע על ידי בדיקה אם גודל קבוצת המדינות שווה ל-1 באמצעות $cond (אם כל השחקנים מאותה מדינה, קבוצה זו תכיל רק ערך אחד).

שלב 6: $project

מציג את התוצאה הסופית תוך הסתרת השדה _id.

- **total_shared_quest_combinations_different_county**: מציג את מספר המשימות בהן יש שחקנים ממדינות שונות.
- **shared_quest_combinations_with_same_country**: מציג את מספר המשימות בהן כל השחקנים באים מאותה מדינה.

אינדקסים שמייעלים את השאילתה

- { player_ids": 1" } על אוסף shared_quests:
  - האינדקס הזה מאפשר לגשת מהר יותר לשחקנים שנמצאים בתוך המערך player_ids לאחר פעולת ה-unwind. אם יש הרבה שחקנים שמבצעים משימות, האינדקס יעזור לסרוק מהר יותר את הרשומות עם המזהה הספציפי של השחקנים.
- { player_id: 1 } על אוסף player:
  - אינדקס זה משפר את הביצועים ב-$lookup, בכך שהוא מאפשר חיפוש מהיר של שחקנים לפי המזהה שלהם מתוך האוסף player כאשר מתבצע החיבור בין השחקנים לבין הנתונים שלהם.

הכנסת נתונים

```mongo
db.player.insertMany(
    [
        {
            "player_id": "p1",
            "username": "DragonSlayer",
            "join_date": "2024-01-01",
            "level": 5,
            "total_play_time": 7200,
            "country": "Israel"
        },
        {
            "player_id": "p2",
            "username": "MageMaster",
            "join_date": "2024-01-05",
            "level": 7,
            "total_play_time": 9000,
            "country": "USA"
        },
```

```mongo
db.quest.insertMany(
    [
    {
        "quest_id": "q101",
        "quest_name": "Clear the Dungeon",
        "difficulty": 3,
        "average_completion_time": 150,
        "failure_rate": 0.14,
        "rewards": ["Gold", "XP"]
    },
    {
        "quest_id": "q102",
        "quest_name": "Defend the Village",
        "difficulty": 4,
        "average_completion_time": 180,
        "failure_rate": 0.12,
        "rewards": ["Gold", "XP", "Item"]
    },
```

```mongo
db.player_action.insertMany([
    {
        "player_id": "p1",
        "action_type": "Quest Completion",
        "quest_id": "q101",
        "difficulty": 3,
        "completion_date": "2024-01-10"
    },
    {
        "player_id": "p1",
        "action_type": "Login",
        "login_date": "2024-01-11"
    },
    {
        "player_id": "p2",
        "action_type": "Quest Completion",
        "quest_id": "q102",
        "difficulty": 4,
        "completion_date": "2024-01-12"
    },
```

```mongo
db.shared_quests.insertMany([
    {
        "quest_id": "q101",
        "player_ids": ["p1", "p2", "p3"]
    },
    {
        "quest_id": "q102",
        "player_ids": ["p1", "p4"]
    },
    {
        "quest_id": "q103",
        "player_ids": ["p2", "p5"]
    },
    {
        "quest_id": "q104",
        "player_ids": ["p3", "p6"]
    },
```

יצירת ה - indexes

```
// player
db.player.createIndex( name: { player_id: 1 }); // Index on player_id (ascending)
db.player.createIndex( name: { level: 1 }); // Index on level (ascending)

// quest
db.quest.createIndex( name: { quest_id: 1 })  // Index on quest_id (ascending)

// player_action
db.player_action.createIndex( name: { player_id: 1, action_type: 1 }); // Compound index on player_id and action_type
db.player_action.createIndex( name: { quest_id: 1 }); // Index on quest_id (ascending)

//shared_quest
db.shared_quests.createIndex( name: { "player_ids": 1 }); // Index on player_ids (ascending)
```