Match4Sport - Software Documentation

מגישים: ירון סופר, אהרון, אייל.

**רעיון האפליקציה:**

**הקדמה:** תאר לעצמך כדורסלן או כדורגלן חובבן אשר מעוניין לצאת למשחק בספורטק השכונתי עם חבריו ביום שישי בצהריים אך חסרים לו שלושה שחקנים על מנת להשלים למשחק של חמש נגד חמש, כאשר רמת השחקנים הינה בינונית פלוס. לשם מטרה זו פיתחנו את **Match4Sport**.

**המטרה המרכזית של האפליקציה** הינה למצוא התאמות למשחקי ספורט ציבוריים בין שחקנים ברמות שונות בעולם כולו. האפליקציה מאפשרת לשחקנים/למשתמשים ליצור משחק חדש תחת ענף ספורט מסויים, כאשר לכל משחק יש שם משלו עם תאריך יצירה שונה, שם משתמש שונה, כמות שחקנים שונה וכן רמת השחקנים הנדרשת שונה.

בנוסף למציאת התאמות, וליצירת משחקים, המשתמשים יכולים לעדכן את פרטי המשחקים אשר אותם הם יצרו ואף למחוק אותם במידת הצורך. לכל משתמש שמורה טבלה אשר מכילה את פרטי המשחקים אליהם הוא בחר להירשם.

בנוסף לכך, האפליקציה מציעה למשתמש למצוא משחקים מסויימים לפי שיטות סינון שונות וכן מציגה עבור המשתמש נתונים סטטיסטיים שונים הודות המשחקים הקיימים ברחבי העולם.

**עיצוב ה Database:**

**תיאור סכמה, טבלאות, עמודות, מפתחות זרים, אינדקסים, יחסים בין טבלאות.**

**טבלאות:**

1. Countries:

|  |  |
| --- | --- |
| country\_id | varchar(35) |
| country\_name | varchar(35) |

טבלה זו מייצגת את כל שמות המדינות ואת הקוד המזהה של כל מדינה כאשר לכל מדינה יש קוד מזהה ייחודי משלה. טבלה זו מתלמאת על ידי שימוש בדאטה סט הנטען במהלך ריצת האפליקציה (יפורט בהמשך).

**המפתח הראשי** של טבלה זו הינו: country\_id.

1. **Cities:**

|  |  |
| --- | --- |
| City\_name | varchar(35) |
| Country\_id | varchar(35) |

**טבלה זו** מייצגת את שמות כל הערים השונית בעולם כאשר היא מקושרת למדינה המתאימה לה על ידי השדה country\_id . נשים לב שעמודה זו נמצאת בטבלה countries.

**המפתח הראשי** של הטבלה cities הינו: (city\_name, country\_id).

**המפתח הזר** של הטבלה cities הינו country\_id אשר מקושר לטבלה countries בהתאם.

1. **Game\_details:**

|  |  |
| --- | --- |
| User\_game | **VARCHAR(15)** |
| **Game\_name** | **VARCHAR(35)** |
| **Game\_date** | **DATETIME (NOT NULL)** |
| **Creation\_date** | **DATETIME (NOT NULL)** |
| **City** | **VARCHAR(70)** |
| **Sport\_category** | **VARCHAR(20)** |
| **Players** | **Int (NOT NULL)** |
| **Level** | **Int (NOT NULL)** |

**טבלה זו** מייצגת את כל פרטי המשחקים הקיימים בעולם והיא מוצגת בחלון ניהוחל המשחקים הראשי של האפליקציה עבור כל יוזר.

**המפתח הראשי** של הטבלה הינו: (user\_game, game\_name, creation\_date)

**המפתח הזר** של הטבלה: city אשר מקושר לעמודה city\_name שבטבלה cities.

1. **Match\_games:**

|  |  |
| --- | --- |
| User\_name | VARCHAR(35) |
| Game\_name | VARCHAR(15) |
| Creation\_date | DATETIME (NOT NULL) |
| Participant | VARCHAR(35) |

**טבלה זו** היא טבלה אשר קיימת עבור כל יוזר שונה במשחק. היא מחזיקה את הפרטים הודות המשחקים אשר אליהם אותו יוזר בחר להירשם.בנוסף לכך, כאשר יוזר יצר משחק חדש, אותו משחק מתווסף אוטומטית לטבלה המשחקים אליהם היוזר רשום. חשוב להדגיש שהעמודה user\_name מייצגת את השם של היוזר אשר יצר את המשחק ואילו העמודה participant מייצגת את היוזר הנוכחי אשר נרשם למשחק.

**המפתח הראשי** של הטבלה: (user\_name, game\_name, creation\_date, participant)

**המפתח הזר** של הטבלה הינו: (user\_name, game\_name, creation\_date)

1. **Users:**

|  |  |
| --- | --- |
| First\_name | **VARCHAR(?)** |
| Last\_name | **VARCHAR(?)** |
| **User\_name** | **VARCHAR(?)** |
| **Password** | **VARCHAR(?)** |
| **Email\_id** | **VARCHAR(?)** |
| **Mobile\_number** | **VARCHAR(?)** |

טבלה זו אחראית להחזיק את המידע הודות כל היוזרים אשר בוחרים להירשם לאפליקציה. הטבלה מכילה את כל הפרטים הבסיסיים הנדרשים עבור כל יוזר.

המפתח של הטבלה הינו: user\_name .

1. **Game\_details\_archives:**

|  |  |
| --- | --- |
| ID | INT (AUTO INCREMENT) |
| User\_name | VARCHAR |
| Game\_name | VARCHAR |
| Game\_date | DATETIME |
| Creation\_date | DATETIME |
| city | VARCHAR |
| Sport\_category | VARCHAR |
| players | INT |
| level | int |
| Deleted\_time | TIMESTAMP |

נעשה שימוש בטבלה זו כאשר האפליקציה מטריגה במקרה מסויים טריגר בשפת SQL שנקרא ‘before\_delete\_game’. טבלה זו מאחסנת את המשחקים מהעולם כולו אשר נכון לרגע נתון מאויישים לפי כמות השחקנים שהוגדרו להם מראש. כלומר, ברגע שכמות השחקנים עבור משחק מסויים היא אפס בטבלה game details המקורית אותו משחק נכנס כרשומה בטבלת הארכיון על ידי הטריגר לשם שחזור מאוחר יותר במידה ואחד השחקנים שנרשם למשחק יחליט שלא ישתתף במשחק זה, משמע יהיה חסר שחקן אחד למשחק זה על מנת שיהיה מאוייש לחלוטין.

**אינדקסים:** לא ראינו צורך באינדקסים, משום שהאפליקציה טוענת את המידע באופן מהיר מספיק.

**הקשרים בין הטבלאות:**

**Dataset: עם איזה dataset עבדנו. איזה תיקונים/שינויים/הרחבות עשינו לdata.**

הדאטה הסט שבו נעשה שימוש הוא... והוא מכיל 130,000 רשומות.

מקור הדאטה סט הוא...

הפורמט של הדאטה סט הוא קובץ אקסל, הדאטה סט המקורי מכיל מספר רב של עמודות שהן...

אנחנו בחרנו להשתמש בעמודות הבאות בלבד... כאשר התיקונים שעשינו הם...

על מנת לפרסר את הדאטה סט ולטעון אותו לטבלאות המתאימות במסד הנתונים שלנו כתבנו סקריפט בשפת פייתון שידאג לעשות זאת. הסקריפט נמצא תחת החבילה DataBase והוא מוטרג על ידי הקוד ג'באי בעת תהליך הקינפוג של מסד הנתונים.

**חבילות חיצוניות חבילות חיצוניות שהקוד עושה בהן שימוש:**

נעשה שימוש בjar על מנת להתחבר למסד הנתונים מסוג mysql: mysql-connector-java-8.0.27.jar.

(עבדנו מול התוכנה MySQL Workbench כקליינט).

בנוסף לכך, על מנת לטעון את הדאטה סט נעשה שימוש בסקריפט פייתוני ולכן הוספנו לסביבת העבודה python interpreter plugin.

נעשה שימוש נרחב באפליקציה בספריה swing של java: import java.swing.\* וכן על ידי java.awt:

Import java.awt.\*.

יתר על כן, עשינו שימוש בתמונות רקע שונות לאפליקציה אשר נמצאות תחת החבילה GUI.

**מבנה הקוד: (לפרט יותר אולי)**

האפליקציה שלנו הינה אפליקצית Desktop אשר כתובה בשפה .Java

את הגישות ל-Database מימשנו על ידי שימוש ב-JDBC API בצד השרת של האפליקציה.

בצד הלקוח (GUI) נעשה שימוש בספריות עיצוב שמציעה הספריה Java Swing.

סביבת הפיתוח שבה השתמשנו היא IntelliJ IDE.

על מנת לטעון את הדאטה הסט לטבלאות הרלוונטיות באפליקציה כתבנו סקריפט בפייתון אשר מוטרג על ידי הקוד הג'באי בתחילת הריצה של האפליקציה **כפי שהוסבר הנ"ל.**

**האפליקציה שלנו מחולקת לחבילות שונות כאשר לכל אחת מהן תפקיד ייעודי:**

1. **GUI** – חבילה זו מכילה את המחלקות השונות אשר אחראיות על החלונות השונים באפילקציה, על המעבר בין החלונות ואת התקשרות בניהם. חבילה זו מייצגת את צד הלקוח של האפליקציה והיא עושה שימוש נרחב ב Java Swing.

**WelcomePage** – מחלקה זו מציגה את חלון הפתיחה של האפליקציה ומעבירה את המשתמש לחלון ההתחברות/הרשמה.

**UserLogin** – במסך זה היוזר יכול להכנס לאפילקציה בעזרת שם משתמש וסיסמא במידה והינו כבר רשום או שעליו להכנס לחלון ההרשמה לאפליקציה.

**UserRegistration** – בחלון זה המשתמש ממלא את פרטיו האישיים על מנת שיוכל להתחבר לאפליקציה. לאחר שהמשתמש מקיש את כל פרטיו הם נשמרים בטבלה users במסד הנתונים של האפליקציה.

**UserHome** – בחלון זה מוצגות בפני המשתמש (לאחר התחברות לאפליקציה) שלושה מקשים כאשר כל אחד מהם פותח חלון אחר בפני עצמו עם ייעוד אחר.

החלון **FindGame** מאפשר למשתמש למצוא משחקים לפי סינונים מסויימים וכן לראות סטטיסטיקות שונות הודות המשחקים הקיימים ברחבי העולם (בעת לחיצה על הכפתורים מוטרגים שאילתות שונות למסד הנתונים דרך **הControllers**, שאילתות אלו אינם ממושות במחלקות הללו של הGUI אלא בחבילה DAO אשר נקראת מתוך **הControllers**).

הכפתור **GameManagement** מוביל למחלקה **JoinGame** אשר מאפשרת למשתמש לראות את כל המשחקים הקיימים (אשר נוצרו על ידי יוזרים אחרים), ליצור משחק חדש, לעדכן פרטים של משחק קיים (או למחוק משחק קיים) אשר נוצר על ידי אותו יוזר, להצטרף למשחק קיים (המשחק יוצג בטבלת הmatch games של אותו יוזר).

בנוסף לכך קיים הכפתור **ChangePassword** אשר מאפשר למשתמש לשנות את הסיסמא שלו (עדכון במסד הנתונים בטבלה users).

בנוסף לכך, תחת התיקייה GUI הוספנו מספר תמונות שמהוות תמונות רקע לחלונות השונים באפליקציה.

1. **DataBase**: חבילה זו מכילה את הממשק **IDataBase** ואת המחלקה **DataBase** אשר ממשת את ממשק זה. הממשק מגדיר תכונות בסיסיות שכל מסד נתונים צריך: - פונקציה אשר נותנת גישה לחיבור למסד הנתונים, סגירת החיבור למסד הנתונים ופונקציה אשר מאפשרת להריץ קובץ SQL על ידי קוד ג'באי אשר מקבל את קובץ ה-sql כקלט.

המחלקה DataBase משתמש בתבנית העיצוב singleton על מנת להבטיח יצירת עצם אחד מסוג מסד הנתונים בעת טעינת האפליקציה. בנוסף לכך, מחלקה זו טוענת את פרטי מסד הנתונים מקבצי קונפיגורציה אשר מוגדרים תחת החבילה Config.

זאת ועוד, מחלקה זו זו מריצה פונקציה בשם dataBaseConfig אשר מקבלת כקלט נתיב של קובץ sql ומריצה אותו אל מוץ מסד הנתונים של האפליקציה בעזרת קריאה לפונקציה runSqlScript.

יתר על כן, המחלקה DataBase מטריגה את תהליך טעינת הדאטה סט על ידי קריאה לסקריפט פייתוני כאשר היא עושה שימוש באובייקט ProcessBuilder.

הטעינה של הדאטה סט מתבצעת רק בעלייה הראשונה של השרת (בעת יצירת הסכמה), קיימת פונקציה במחלקה Database אשר בודקת האם הטבלאות country ן cities ריקות או לא.

1. **Models:** חבילה זו מכילה שתי מחלקות אשר מגדירות את שתי האובייקטים העיקריים באפליקציה: **game** and **user**. לכל מחלקה יש בנאי מתאים, שדות מתאימים לעמודות בטבלאות שהוגדרו במסד הנתונים ופונקציות נחוצות כמו **getters & setters** לשדות של מחלקות אלו.
2. **Dao:** בחבילה זו נעשה שימוש בתבנית עיצוב **Data Access Object** עבור שתי המחלקות שהגדרנו תחת החבילה Models. בחבילה זו מוגדרים שני ממשקים ושתי מחלקות אשר יורשות את הממשק המתאים להם. בממשקים מוגדרים החתימות של הפונקציות השונות אשר מבצעות גישות שונות למסד הנתונים בעזרת **JDBC API** הן עבור האובייקט **Game** והן עבור האובייקט **User**. חבילת הGUI יוצרת אינסטס מתאים של אובייקטים מסוג **Controller** אשר ממפים את הקריאות לאובייקטים מסוג Dao בהתאם. המחלקה **GameDaoImpl** מכילה את כל השאילתות אשר מופנות למסד הנתונים כאשר האובייקט הוא Game (מחלקה זו ממשת את הממשק **GameDao**). באותו אופן, המחלקה **UserDaoImpl** מכילה את כל השאילתות אשר מפונות למסד הנתונים כאשר מדובר באובייקט user (מחלקה זו ממשת את הממשק **UserDao**).
3. **Controllers:** בחבילה הגדרנו שתי מחלקות **UserController** ו- **GameController** אשר מאותחלים במחלקות הGUI והן אחראיות למפות ולתקשר בקשות ותשובות בין צד הלקוח לבין השרת ומסד הנתונים של האפליקציה. האובייקט GameController עוטף את כל הפניות הקשורות לאובייקט Game ואילו האובייקט UserController עוטף את כל הפניות הקשורות לאובייקט User. השימוש בחבילה זו חשוב ברמת ארכיטקטורת האפילקציה שכן הוא מהווה כגוף המקשר בין צד הלקוח לבין השרת ומסד הנתונים. כמוכן, הוא יוצר הפרדה מלאה בין שתי צדדים אלו.
4. **SqlUtils:** תחת חבילה זו נמצאים קבצי sql אשר ההרצה שלהם נעשית במחלקה DataBase אשר נמצאת בחבילה **DataBase.** קבצי sql אלו רצים על ידי קוד ג'אבאי אשר מפרסר ומריץ אותם מול מסד הנתונים של האפליקציה.

**createdDB.sql** - קובץ שיוצר את מסד הנתונים.

**createUsersTable.sql** – קובץ אשר יוצר את הטבלה users.

**createGamesTables.sql** – קובץ אשר יוצר את הטבלאות: countries, cities, game\_details, match\_games.

1. **Config:** תחת חבילה זו הגדרנו שני קבצי קונפיגורציה, אחד עבור ההתחברות למסד הנתונים ושמו **jdbc.properties** והשני עבור ה-Thread Pool אשר נעשה בו שימוש כחלק מארכיטקורת client server ושמו **transport.properties**(ניתן להגדיר בו את מספר הקליינטים שהשרת יוכל לטפל בהם בוזמנית, את ההוסט ואת הפורט של השרת).
2. **Utils:** תחת חבילה זו מוגדרת מחלקה אשר מכילה פונקציות יוטיליטי שונות אשר העיקריות בה הן פונקציות אשר קוראות ושומרות את הנתונים מקבצי הקונפיגורציה אשר נמצאים בחבילה Config.
3. **ClientServer:** תחת חבילה זו מוגדרים מס' מחלקות אשר מאפשרות לנו לממש את ארכיטקטורת הclient server באפליקציה שלנו. מוגדרת מחלקה **ThreadPooledServer** אשר מייצגת את השרת של האפליקציה אשר אחראי לטפל במספר לקוחות במקביל (מספר הלקוחות שניתן לטפל בהם במקביל מוגדר בקובץ transport.config) כאשר עבור כל לקוח/יוזר אשר מתחבר לאפליקציה מוקצה ת'רד עבורו. המחלקה **WorkerRunnable** מייצגת את היוזרים השונים אשר מתחברים לאפליקציה. שתי מחלקות אלו ממשות את הממשק runnable וזאת על מנת שכל אינסטנס מסוגם ירוץ על תרד משלו.
4. **ApplicationStartUp:** בחבילה זו מוגדר קלאס אחד **InitApplication** אשר מהווה כשרת של האפליקציה. הוא מאתחל את ה-Thread Pool, יוצר, מקנפג ומשתמש באינסטס יחיד של מסד הנתונים ולבסוף מחכה בלולאה אינסופית לקבלת לקוחות בhostname ובport המתאימים.

**תרשים זרימה של האפליקציה:**

כפי שהוסבר לעיל, האפליקציה עושה שימוש בתרד פול כחלק ממימוש ארכטיקטורת client-server.

המחלקה שמתחילה לרוץ הינה InitApplication (כשרת האפליקציה) וממתינה ללקוחות מסוג WorkerRunnable.

בצד הלקוח, המידע מוצג ליוזר ובהתאם לפעולותיו האובייקטים UserController ו- GameController אחראים למפות בקשות ותשובות בין הלקוח לבין שרת האפליקציה ומסד הנתונים.

Register was failed

Find Game

Register was succeeded

Register Window

Game Management

User Home

Login

Register

Welcome Window

Yes

Is registered user?

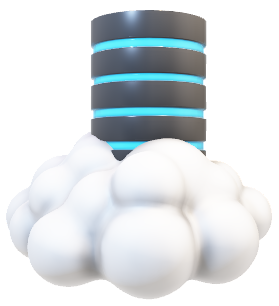
No

Change Password

Controllers



Queries, Updates & response

****

Daos

**שאילתות ועדכונים:** נתחיל בלהציג ולהסביר את השאילתות של האובייקט user כאשר החתימות של פונקציות אלו נמצאות בממשק UserDao והמימוש נמצא במחלקה UserDaoImpl.

**הוספת משתמש/שחקן חדש לטבלת users:**

**INSERT INTO users(first\_name, last\_name, user\_name, password, email\_id, mobile\_number)**

**VALUES(?, ?, ?, ?, ?, ?);**

הערכים אשר נכנסים לטבלת users בבסיס הנתונים הינם הערכים אשר המשתמש מילא בתהליך ההרשמה לאפליקציה.

**התחברות לאפליקציה:**

**Select user\_name, password from users where user\_name=user\_name and password=password;**

האפליקציה מוודאת שהשם משתמש והסיסמא שהתקבלו כקלט אכן נמצאת בטבלת users.

**ולידציה בעת החלפת סיסמא של היוזר:**

**Select password from users where user\_name = user\_name;**

האפליציה מוודאת שהסיסמא החדשה שהוכנסה שונה מהסיסמא הקודמת.

**שינוי סיסמא של היוזר:**

**UPDATE users set password = password where user\_name = user\_name;**

במידה ותהליך הולידציה בשאילתא הקודמת הצליח, האפליקציה מעדכנת את הסיסמא של אותו יוזר לסיסמא החדשה בטבלת users שבבסיס הנתונים.

כעת נעבור לשאילתות אשר מתייחסות לאובייקט game, החתימות שלהן נמצאות במחלקה GameDao והמימוש שלהן נמצא בתיקייה GameDaoImpl.

**לחיצה על הכפתור retrieve:**

**SELECT distinct t1.user\_name, t1.game\_name, t1.sport\_category, t3.country\_name, t1.city, t1.game\_date, t1.players, t1.level, t1.creation\_date FROM game\_details as t1 JOIN cities as t2**

**ON t1.city = t2.city\_name**

**JOIN countries as t3**

**ON t2.country\_id = t3.country\_id**

**ORDER BY unix\_timestamp(t1.game\_date);**

שאילתא זו מחזירה את כל המשחקים אשר נוצרו אי פעם על ידי כל היוזרים הרשומים לאפליקציה. התוכן של טבלה זו מוצג בחלון GameManagement בטבלה השמאלית למעלה. ברגע השיוזר לוחץ על הכפתור retrieve הוא מקבל את כל המידע אשר נמצא בטבלה זו.

**לחיצה על הכפתור Your Games:**

**SELECT user\_name, game\_name, creation\_date, participant**

**FROM match\_games**

**WHERE participant = ?;**

שאילתא זו מספקת ליוזר את כל המשחקים אשר הוא רשום אליהם (המשחקים מוצגים בטבלה Your Matches).

**לחיצה על הכפתור add:**

ראשית האפילקציה מוודאת שהמדינה והעיר אשר נכנסו כקלט על ידי המשתמש נמצאים בטבלאות countries & cities אשר נטענו עם המידע מהדאטה סט בעת עליית האפליקציה.

**SELECT country\_name FROM countries WHERE country\_name = ?;**

**SELECT city\_name FROM cities WHERE city\_name = ?;**

כעת מתבצעת טרנזקציית המורכבת משתי שאילתות הכנסה לשתי טבלאות שונות:

שאילתת ההכנסה הראשונה הינה לטבלה game\_details:

**INSERT IGNORE INTO game\_details(user\_name, game\_name, game\_date, creation\_date, city, sport\_category, players, level) VALUES(?, ?, ?, ?, ?, ?, ?, ?);**

שאילתת ההכנסה השנייה הינה לטבלה match\_games אשר מייצגת את המשחקים אשר רשום אליהם היוזר הנוכחי:

**INSERT IGNORE INTO match\_games(user\_name, game\_name, creation\_date, participant)**

**VALUES(?, ?, ?, ?);**

שתי השאילתות הנ"ל מתבצעות תחת אותה טרנזקציה כך שכישלון של אחת מהן יבטל קומיט לבסיס הנתונים של השאילתא האחרת שכן הצליחה.

**לחיצה על כפתור update:**

כמו קודם, האפילקציה מוודאת שהמדינה והעיר אשר נכנסו כקלט על ידי המשתמש נמצאים בטבלאות countries & cities אשר נטענו עם המידע מהדאטה סט בעת עליית האפליקציה (ראה שאילתא בתחילת העמוד).

במידה והיוזר אשר מנסה לעדכן את המשחק הוא אותו יוזר אשר יצר אותו מתבצעות השאילתות הבאות על מנת לבצע עדכון לערכי המשחק:

מציאת שם המשחק שאותו היוזר מעוניין לעדכן על ידי שימוש בסממן שמצביע על השורה שהיוזר בחר להחליף:

**SELECT game\_name FROM game\_details**

**ORDER BY unix\_timestamp(game\_date)**

**LIMIT ?, 1;**

* הערך שמקבל סימן השאלה הוא מספר השורה של המשחק שהיוזר מעוניין לעדכן את ערכיו.

כעת נעבור לשאילתת העדכון אשר מקבלת כקלט אובייקט מסוג game עם כל הערכים החדשים שבחר היוזר לעדכן ואת שם המשחק הישן:

**UPDATE game\_details SET game\_name = ?, game\_date = ?, city = ? ,**

**sport\_category = ?,players = ?, level=?**

**WHERE game\_name = ?;**

**לחיצה על הכפתורgame delete לשם מחיקת משחק מטבלת המשחקים העולמית:**

במידה והיוזר אשר מנסה למחוק את המשחק הוא אותו יוזר אשר יצר אותו מתבצעות השאילתות הבאות בטרנזקציה:

מחיקה מהטבלה game\_details:

**DELETE FROM game\_details WHERE game\_name = ? AND creation\_date = ?;**

מחיקה מהטבלה match\_games:

**DELETE FROM match\_games WHERE game\_name = ? AND creation\_date = ?;**

**לחיצה על הכפתור delete match על מנת למחוק משחק שיוזר מסויים רשום אליו מטבלת המשחקים האישיים:**

בעת לחיצה על כפתור זה מתבצעת טרנזקציה של השאילתות הבאות:

מחיקה מהטבלה match\_games.

**DELETE FROM match\_games WHERE participant=? AND game\_name=?;**

עדכון כמות השחקנים בטבלה game\_details בפלוס שחקן אחד לאותו משחק:

**UPDATE game\_details SET players = +1 WHERE game\_name = ?;**

**\*If the current game does not appear in table game\_details because it was matched the application code retrieves the game details back from the archive table to the orig game\_details table.**

**INSERT IGNORE INTO game\_details(user\_name, game\_name, game\_date, creation\_date, city, sport\_category, players, level)**

**SELECT user\_name, game\_name, game\_date, creation\_date, city, sport\_category, players + 1, level**

**FROM game\_details\_archives WHERE game\_name=?;**

**לחיצה על כפתור (Trigger in edge case) Join A Game:**

ראשית, האפליקציה מוודא שהמשחק המבוקש אינו קיים כבר בטבלת המשחקים האישיים של היוזר ע"י השאילתה:

**SELECT \* FROM match\_games WHERE user\_name = ? and game\_name = ? and creation\_date = ? and participant = ?;**

במידה והשאילתא מחזירה ערך כלשהו תהליך הוספת המשחק מפסיק ומוצגת הודעה מתאימה ליוזר על כך שהוא כבר רשום למשחק.

במידה והשאילתא החזירה false (כלומר, המשחק לא קיים בטבלה האישית של אותו יוזר) מתבעת טרנזקציה אשר מורכבת מהשאילתות הבאות:

הכנסת פרטי המשחק לטבלה האישית של היוזר:

**INSERT IGNORE INTO match\_games(user\_name, game\_name, creation\_date, participant)**

**VALUES(?, ?, ?, ?);**

עדכון כמות השחקנים בטבלה game\_details כאשר במקרה זה כמות השחקנים יורדת ב1 עבור אותו משחק:

**UPDATE game\_details SET players = -1 WHERE game\_name = ?;**

**\*If the number of players of the current game is equal to zero the application code invokes SQL trigger called: “before\_delete\_game” that stores the current game details inside archive table.**

**CREATE TRIGGER before\_delete\_game BEFORE DELETE ON game\_details**

**FOR EACH ROW Begin**

**INSERT INTO game\_details\_archives(user\_name , game\_name , game\_date , creation\_date , city , sport\_category , players , level )**

**VALUES ( OLD.user\_name , OLD.game\_name , OLD.game\_date , OLD.creation\_date , OLD.city , OLD.sport\_category , OLD.players , OLD.level );**

**END**

**After the trigger complete successfully, the application code deletes the game from table game\_details:**

**DELETE FROM game\_details WHERE game\_name = ? AND creation\_date = ?;**

**לחיצה על הכפתור clear: לחיצה על כפתור זה אינה מפעילה שאילתת sql כלשהי, היא מחזירה טבלה ריקה ברמת הGUI של Java Swing.**

**כעת נציג את השאילתות המוטרגות תחת החלון FindGame:**

לחיצה על הכפתור Find game אשר מאפשר למצוא את פרטי משחק מסויים לפי שמו:

**SELECT distinct t1.user\_name, t1.game\_name, t1.sport\_category, t3.country\_name, t1.city, t1.game\_date, t1.players, t1.level, t1.creation\_date**

**FROM game\_details as t1**

**JOIN cities as t2**

**ON t1.city = t2.city\_name**

**JOIN countries as t3**

**ON t2.country\_id = t3.country\_id**

**WHERE t1.game\_name=?**

**ORDER BY unix\_timestamp(t1.game\_date);**

לחיצה על הכפתור Find all games in city למציאת כל המשחקים בעיר מסויימת:

**SELECT distinct t1.user\_name, t1.game\_name, t1.sport\_category, t3.country\_name, t1.city, t1.game\_date, t1.players, t1.level, t1.creation\_date**

**FROM game\_details as t1**

**JOIN cities as t2**

**ON t1.city = t2.city\_name**

**JOIN countries as t3**

**ON t2.country\_id = t3.country\_id**

**WHERE t2.city\_name=?**

**ORDER BY unix\_timestamp(t1.game\_date);**

לחיצה על הכפתור Find all games in country למציאת כל המשחקים במדינה מסויימת:

**SELECT distinct t1.user\_name, t1.game\_name, t1.sport\_category, t3.country\_name, t1.city, t1.game\_date, t1.players, t1.level, t1.creation\_date**

**FROM game\_details as t1**

**JOIN cities as t2**

**ON t1.city = t2.city\_name**

**JOIN countries as t3**

**ON t2.country\_id = t3.country\_id**

**WHERE t2.country\_name=?**

**ORDER BY unix\_timestamp(t1.game\_date);**

מציאת כל המשחקים ברמה הגבוהה יותר לפי ענף ספורט (כאשר היוזר מכניס כקלט את ענף הספורט):

SELECT distinct t1.user\_name, t1.game\_name, t1.sport\_category, t3.country\_name, t1.city, **t1.game\_date, t1.players, t1.level, t1.creation\_date**

**FROM game\_details as t1**

**JOIN cities as t2**

**ON t1.city = t2.city\_name**

**JOIN countries as t3**

**ON t2.country\_id = t3.country\_id**

**WHERE t1.sport\_category = ?**

**GROUP BY t1.game\_name, t1.sport\_category, t1.level**

**ORDER BY max(t1.level) DESC**

**LIMIT 10;**

מציאת המדינה שבה ענף ספורט מסויים משוחק הכי הרבה (שאילתא שמוצגת בטבלת נתוני הסטטיסטיקה):

**SELECT t3.country\_nameFROM game\_details as t1**

**JOIN cities as t2 ON t1.city = t2.city\_name**

**JOIN countries as t3 ON t2.country\_id = t3.country\_id**

**WHERE t1.sport\_category = ?**

**GROUP BY t3.country\_name**

**ORDER BY COUNT(\*) DESC**

**LIMIT 1;**

מציאת תחום הספורט אשר משוחק הכי הרבה בחודש הנוכחי (שאילתא שמוצגת בטבלת נתוני הסטטיסטיקה):

**SELECT distinct t1.sport\_category**

**FROM game\_details as t1**

**WHERE month(t1.game\_date) = (SELECT MONTH(CURDATE()))**

**GROUP BY t1.sport\_category**

**ORDER BY COUNT(\*) DESC LIMIT 1;**

תחום הספורט שבו כמות השחקנים המבוקשת היא הנמוכה ביותר בממוצע (שאילתא שמוצגת בטבלת נתוני הסטטיסטיקה):

**SELECT distinct t1.sport\_category, avg(t1.players) as avg\_players**

**FROM game\_details as t1 JOIN cities as t2**

**ON t1.city = t2.city\_name**

**JOIN countries as t3**

**ON t2.country\_id = t3.country\_id**

**where t3.country\_name = ?**

**group by t1.sport\_category**

**order by avg\_players asc**

**limit 1;**