1. המסד מכיל מספר טבלות:

Users: מכיל מידע אודות המשתמש – שם המשתמש (להתחברות וזיהוי), סיסמה, שם ויתרה

Friendships: מכיל מידע אודות החברויות. עבור זוג חברים u1,u2 אנו שומרים בטבלה זוג שורות u1,u2 ו-u2,u1. אמנם בזבוז זיכרון, אך מאפשר שאילתות יותר פשוטות ויותר מהירות.

Activities: מכיל מידע אודות הפעילויות: מספר זיהוי, כותרת, שם המשתמש שיצר אותה, מכסת משתמשים וסוג (שירות או משימה)

Activities\_registrations: מכיל מידע אודות הרשמות לפעילויות: מספר זיהוי הפעילות ושם המשתמש שנרשם אליה  
החל מתחילת הפרויקט תכננו כיצד נוכל ליישם מקביליות אשר תהיה מהירה אך עם זאת עקבית.

ראשית השתמשנו במאגר חיבורים (connection pool), אשר כל החיבורים נלקחים ממנו ובכך מונעים יצירת חיבורים למאגר בכל גישה (מה שמקצר את זמן הגישה).

החלטנו להשתמש ב-MySQL עם מנוע innoDB אשר מאפשר שימוש בטרנזקציות, ומראש הגדרנו חלק מהפונקציות לגישה למסד הנתונים (במחלקות אשר אמונות על כך) שיקבלו משתנה מטיפוס connection. בצורה זו יכולנו בסופו של יום להשתמש בחיבור בעל טרנזקציה, שישמש למספר שאילתות, ובכך נקבל טרנזקציה אטומית אשר תשאיר את המסד עקבי.

כמו כן השתמשנו ברמות שונות של בידוד טרנזקציות, החל מקריאות לא מחויבות (uncommitted) ועד לרמה של סריאליות.

השתדלנו שרוב הגישות למטרת שאילתות יהיו ברמה נמוכה (ללא נעילות ועם קריאות לא עקביות), ואילו הגישות לעדכון מסד הנתונים ברמה הגבוה למען שמירת עקביות המסד.

בצורה זו חסכנו זמן יקר על הבאת מידע על חשבון נכונותו (כאשר היינו מוכנים לשלם זאת), ואילו בעדכוני המסד לא חיפפנו ווידאנו את נכונותו גם על חשבון המקביליות.

לכן סה"כ, ע"י שימוש במאגר חיבורים אשר מקצר את זמן ההמתנה לחיבור למסד, ושימוש בטרנזקציות ברמות בידוד שונות, הגענו לאיזון האופטימאלי לדעתנו בין רמת המקביליות לרמת העקביות.

בצורה זו גם משרתים גישות מקבילות ע"י משתמשים רבים בצורה טובה, תוך איזון המקביליות והעקביות (כאשר ככלל העדיפות היא לעקביות).

1. לב פונקציית החיפוש (מציאת חברים במרחק 1 או 2 ממשתמש נתון) היא בשאילתה המורכבת שלה.

תחילה אנו מוצאים את שמות כל המשתמשים במרחק הרצוי, כולל המרחק המינימאלי מהם.

המציאה מבוצעת מאיחוד של כל החברים עם החברים של החברים.

מציאת החברים היא מהירה ע"י שימוש באינדקס על שם המשתמש הראשון.

במציאת החברים ממרחק 2 אנו מבצעים תחילה חיפוש של החברים המידיים, ומבצעים JOIN עם טבלת החברים הכוללת.

הטבלה הראשונה ב-JOIN היא של חברי המשתמש בלבד, והיא מגודל N בלבד (מספר חברי המשתמש), גודל מזערי לעומת הטבלה כולה. שוב בעזרת INDEX חכם ה-JOIN מבוצע באופן מהיר ואנו נותרים עם חברים ממרחק 2, טבלה קטנה בהרבה מהטבלה כולה.

לבסוף אנו ממיינים (עדיין ב-database) את החברים לפי מרחקם המינימאלי מהמשתמש (ייתכן מצב שמשתמש מסוים הוא גם חבר מידי וגם חבר ממרחק 2), ובעזרת JOIN פשוט על חברת המשתמשים אנו מוצאים גם את המידע על אותם המשתמשים.

בסופו של דבר, מה שעושה את הפעולה ליעילה כל כך היא שימוש בלעדי בשאילתת ה-SQL עם צמצום כמה שיותר מוקדם של הטבלות בשאילתה עצמה.

השימוש הבלעדי בשאילתת ה-SQL למען הבאת המידע מצמצמת הבאה של מידע מיותר, ומאפשרת לשרת ה-SQL לבצע את כל הפעולות בעצמו (קוד שנכתב ונבדק ע"י מומחים רבים, ומהיר בהרבה מכל קוד שנוכל לכתוב ב-JAVA). כמו כן צמצום מוקדם של הטבלות מאפשר לשרת לעבוד בכל שלב בשאילתה עם טבלאות קטנות.

נציין גם כי הפעולה עצמה מורכבת משאילתה אחת בלבד, ולא מטרנזקציה בעלת מספר שאילתות, וכן כי היא מבוצעת ברמת בידוד נמוכה. גם דברים אלו מאפשרים להאיץ מעט את השאילתה.

1. גם כאן, בדומה לשיקולי המקביליות, חשבנו עוד מתחילת הפרויקט על סקלביליות (scalability) האתר. הדבר טופל מעבר לפתרונות המקביליות הרגילים אשר הובאו בשאלה 1.

כך למשל הבנו כי מס המשתמשים הכולל, וכן מספר הפעילויות המוצעות וכדומה, יכולים להיות גדולים. הבאה של כל המשתמשים לצורך הצגה יהיו לא יעילים הן מבחינת הבאת מידע עצום מהמסד, כן ביצירת קובץ html גדול לשליחה לדפדפן וכן באי יכולת המשתמש להתבונן במידע כה רב.

לכן השתמשנו בשאילתות אשר מביאות רק חלק מן המסד, והדפים אשר מציגים מידע זה מותאמים להצגת חלקים בודדים שלו.

נציין כי לא תמיד טרחנו לעשות זאת. כך למשל הנחנו כי מספר החברים של משתמש יהיה קטן דיו כדי לא להתעסק עם הבאת חלקים. כמובן שבאתר אמיתי, שבו היה לנו יותר זמן ומשאבים, היינו מטפלים גם במקרה זה ואחרים.