**חלק יבש**

**מגישים:** אייל טולצ'ינסקי, מרבי צ'יקושווילי

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).

2. תחילה דאגנו להפרדה הבסיסית ביותר בין המודולים הגדולים ביותר, מסד הנתונים, הממשק הגרפי והלוגיקה. מסד הנתונים הוא שרת mysql 5.6 שאליו אנו מתחברים בעזרת מאגר חיבורים ומבצעים עליו שאילתות שתומכות בכל הלוגיקה של האתר, הממשק הגרפי הם דפי ה- html, jsp (the rendered part) שמספקים את דרך הצגת הנתונים והלוגיקה מאחוריי כל פעולה מתבצעת ב- servlets, jsp (the code snippets). ע"י כך דאגנו כי המודולים הגדולים לא יסתבכו בינהם ויצרו תלויות שאנו לא רוצים. בתוך כל מודול דאגנו להפרדות נוספות, למשל, לכל טבלה יש מחלקה משלה המטפלת בפעולות לה תזדקק הטבלה. בנוסף למחלקות הקיימות לכל טבלה שיש בסכמה שלנו יש לנו מחלקה שמכירה בכל המחלקות הנ"ל על מנת לבצע טרנזקציות, היא מכירה בכל המחלקות והפונקציות שלה המקבלות חיבור ודואגת לקרוא להן לפי הסדר על מנת לבצע את הטרנזקציות.

בנוסף ברמת מסד הנתונים אנו דאגנו לדאוג להפרדה המירבית ביותר בין טבלאות כך שלא תהיינה איזשהן תלויות פונקציונליות בין הטבלאות ודאגנו שלא נשמור מידע מיותר שיכול לגרום לחוסר קונסיסטנטיות במסד הנתונים.

ברמת דפי ה-html אין יותר מדי מה לפרט, דאגנו כי לכל מידע שסביר על הדעת שיהיה באתר יהיה דף html שיציגו.

ברמת הservlets דאגנו תחילה כי לכל פונקציונליות אפשרית בדפי ה- html יהיה לה servlet שיקרא וכך נשמרת המודולריות כך שבעצם כל servlet משרת מטרה אחת בלבד ולא משותף לכמה דפים נוספים. אך ברגע כי שמנו לב כי קיימות חזרה על קוד, למשל כשעוברים לכל עמוד לוודא כי המשתמש אכן מחזיק session חוקי או בעל cookies חוקיים החלטנו לבצע בדיקות אלה בעזרת פילטרים שיבוצעו לכל בקשה וכך בעצם נבטיח חסך בשיכפול קוד ואפשרות להשתמש בפילטר בעוד מקומות.

לסיכום, תמיד היה בראש סדר העדיפויות שלנו לשמור על תכן נקי ומודולרי על מנת להפוך את חיינו לקלים יותר בזמן הניפוי שגיאות וכו'.. עמדנו במשימה היטב ויצרנו קוד בעל תכן מודולרי ומספק.

3. את הקלט שלנו אנו מעבירים בסופו של דבר למחלקות המתאימות המטפלות בטבלה הרלוונטית לשאילתה. תחילה כל קלט אשר אנו מקבלים מהמשתמש מועבר למסד הנתונים אך ורק דרך preparedStatement על מנת למנוע כל אפשרות של Sql Injection, בכך אנו מבטיחים כי הקלט שהמסד נתונים שלנו יקבל לא יגרום לאיזשהי התנהגות לא צפויה ולא רצויה מצידו וחשיפה של מידע לא נחוץ למשתמש.

הקלט עובר כל הדרך אל המחלקה המטפלת במסד הנתונים הרלוונטי ללא כל בדיקה בדרך, הקלט נבדק אך ורק בפונקציה הרלוונטית במחלקה, תחילה אנו בודקים ערכי null ואם קיימים כאלה אנו זורקים חריגה שתתפס יותר מאוחר ב- servlet המתאים, ולאחר מכן אנו מבצעים את הפעולה המבוקשת ואם במקרה נזרקת חריגה הקשורה לפרמטרים שהמסד קיבל גם אותה אנו זורקים הלאה עד ה- servlet הרלוונטי.

על מנת שהאפליקציה שלנו תתאים ללקוחות השונים המשתמשים בה בכל הדפים אנו לא משתמשים בגדלים הנקבעים בפיקסלים או מדדים כאלה ואחרים, אלא רק במידות המסך, זאת אומרת אחוזים ממנו.

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.

6. כל השגיאות שיתכנו נזרקות מהמסד הנתונים שלנו לכל פעולה שמתבצעת בכל דף. יצרנו לכל טבלה שיש לנו את כל החריגות היתכנו ממנה ודאגה לכך שה-stackTraceTree של החריגה המקורית יודפס גם בחריגה שלנו על מנת שיהיה נוח לבצע ניפוי שגיאות. בסופו של דבר כל טבלה זורקת חריגות שאנו הגדרנו לפי הצורך שראינו לנכון ומה שקריטי ליצירת חיווי מתאים למשתמש של המערכת. ע"י כך שאנו זורקים מהטבלאות הלאה את החריגות שאנו הגדרנו ויצרנו אנו יודעים כיצד להתמודד עם כל חריגה שיכולה להווצר בכל המודולים המשתמשים במסד הנתונים. ע"י כך אנו בעצם זורקים את החריגות מהמסדים ותופסים אותם ב- servlets הרלוונטיים ובהתאם לכל חריגה אנו יודעים להוציא את הודעת השגיאה המתאימה. אנו גם לא רוצים שחיווי שגיאות תהיה פעולה יקרה שפוגעת בסקלביליות המערכת ומוודאים שכל עמוד מרנדר רק חלק קטן מאוד ממנו ע"י שימוש ב-ajax.