הסבר ל-Database Design ול-implementation של ה-API

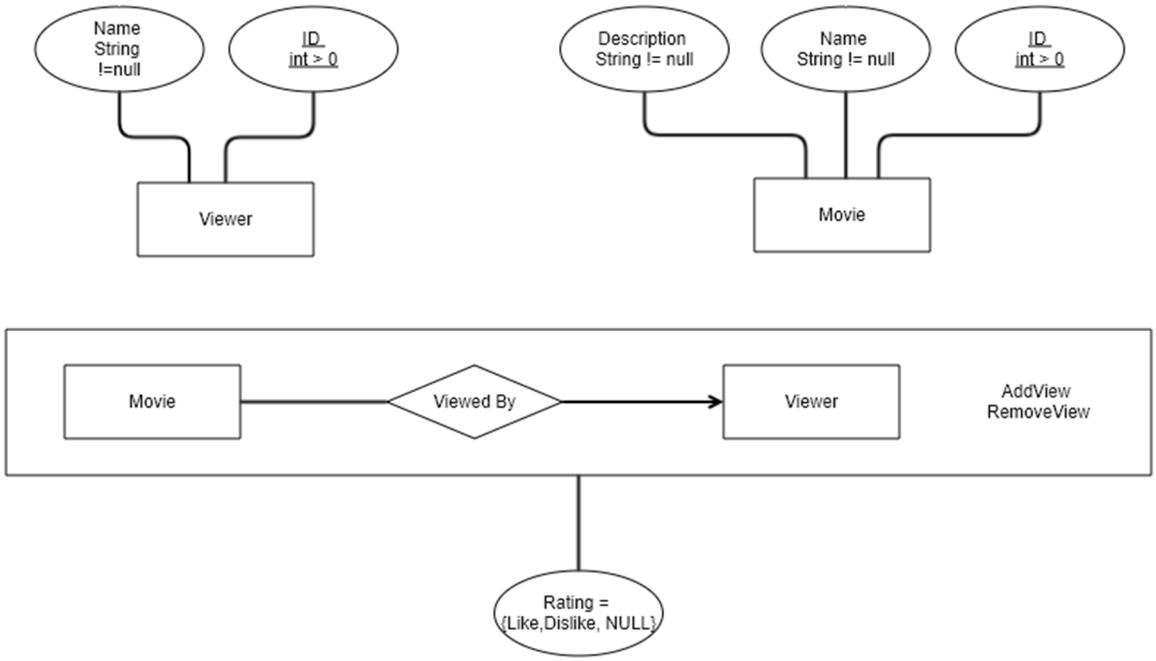
הסבר ל-design הכללי:

המערכת שלנו מורכבת מ-3 טבלאות.

1) הטבלה הראשונה היא עבור הצופים, ומורכבת ממזהה הצופה ומשמו. מזהה הצופה הינו Primary Key, ושם הצופה אינו יכול להיות ריק.
2) הטבלה השנייה היא עבור הסרטים, ומורכבת ממזהה הסרט, משמו ומתיאורו. מזהה הסרט הינו Primary Key, ושם הצופה ותיאורו אינם יכולים להיות ריקים.
3) הטבלה השלישית נקראת viewedBy והיא מורכבת משלוש עמודות, מזהה של צופה, מזהה של סרט ודירוג. לטבלה הזאת נכניס צפיות. כלומר, אם צופה 1 צפה בסרט 1, נכניס זאת לטבלה כרשומה חדשה עם דירוג null בתור ברירת מחדל. עמודת הדירוג יכולה להכיל LIKE, null או DISLIKE. רק לאחר שנוסף דירוג באמצעות פונקציה יעודית תשתנה עמודת ה-null ל-LIKE/DISLIKE בהתאם לדירוג - <u>עבור רשומה קיימת</u>. בנוסף, גם מזהה הצופה וגם מזהה הסרט ביחד מהווים Primary Key. כל אחד מהם בנפרד הוא Foreign Key.

הערה: לאחר מחיקת צופה או סרט, נמחקים באופן אוטומטי גם כל הצפיות המכילות את הצופה הנ"ל ו/או את הסרט הנ"ל

תרשים ERD של המסד:



הסבר לפונקציות מתוך Basic API:

- addView: אנו מוסיפים רשומה לטבלה viewedBy. כל רשומה מורכבת מ-viewerID, movieId ו-rating. בתור ברירת מחדל אנו קובעים את ה-rating להיות null. שדה זה משתנה רק כאשר נקראת פונקציה אשר נותנת דירוג. בהתאם להגדרת הטבלה לעיל, במידה ולא קיים movieID או viewerID מתאים בטבלת הסרטים או הצופים בהתאמה, נחזיר ערך שגיאה.

- removeView: מוריד רשומה מהטבלה. בהתאם להגדרת הטבלה לעיל, במידה ולא קיימים movieID או viewerID מתאים באותה הרשומה בטבלה viewedBy, נחזיר ערך שגיאה.

- getMovieViewCount: אנו מחזירים את מספר הרשומות בטבלה viewedBy שבה מזהה הסרט הוא המזהה שנשלח כארגומנט. זה יתן לנו את מספר הצפיות הכולל בסרט.

- addMovieRating: זו פונקציה המשנה את הדירוג כפי שהסברנו קודם. במקום הדירוג שהיה קודם, הדירוג משתנה להיות LIKE/DISLIKE. אם לא קיימת צפייה כזאת במערכת, אנו מחזירים NOT_EXISTS.

- removeMovieRating: אנו מסירים דירוג מסוים ומחזירים אותו להיות null (כלומר חסר-דירוג מהצופה הנ"ל). בדומה לaddMovieRating, אם לא קיימת צפייה כזאת במערכת, אנו מחזירים NOT_EXISTS.

- getMovieLikesCount: אנו מחזירים את מספר הרשומות ב-viewedBy שבהן הסרט מופיע עם דירוג LIKE.

- getMovieDislikesCount: באופן דומה לפונקציה הקודמת, רק עם DISLIKE.

הסבר לפונקציות נבחרות מתוך Advanced API:

- getSimilarViewers: נסמן את הצופה עליו שואלים בתור צופה א'. אנו בונים טבלה חדשה שנקראת t1 המכילה מזהה של צופה ומספר הסרטים שהוא צפה בהם (מתוך הטבלה viewedBy). אנו מוודאים שהצופים המוכנסים לטבלה הם לא הצופה א' וגם שלסכום הסרטים נכנסים אך ורק סרטים שצופה א' צפה בהם. לבסוף באמצעות מילת המפתח HAVING אנו ממיינים לפי צופים שמספר הסרטים שהם צפו בהם גדול משלושת רבעי (כפל ב-0.75) ממספר הסרטים שבהם צפה צופה א'. לבסוף מתוך הטבלה t1 אנו לוקחים רק את העמודה שהיא מזהה הצופה, וממיינים את התוצאה לפי מזהה זה.

- mostInfluencing: אנו לוקחים את כל מזהי הצופים מתוך הטבלה viewedBy מקובצים לפי מזהי הצופים, ואז ממוינים לפי - תחילה מספר הפעמים שמופיע צופה מסוים, כלומר מספר הצפיות שלו - בסדר יורד, לאחר מכן לפי מספר הדירוגים שלו בסדר יורד, ורק אז לפי מזהה הצופה בסדר עולה. כל השאילתה מוגבלת ל-10 התוצאות הראשונות. כך נקבל את 10 הצופים המשפיעים ביותר.

- getMoviesRecommendation: ראשית, נסמן את הצופה עליו שואלים בתור צופה א'. נקרא לשאילתה מgetSimilarViewers על מנת לקבל את רשימת הצופים הדומים.
ניצור טבלה חדשה ומכניסים אליה את כל הצופים הנ"ל על-מנת שיהיה לנו יותר קל לעבוד איתם. כעת אנו מרכיבים שתי טבלאות שונות, טבלה א' וטבלה ב', ולאחר מכן מבצעים להן UNION:
טבלה א' מורכבת מרשומות מתוך הטבלה viewedBy - מזהי הסרטים, מספר הלייקים ואת המספר 1 כעמודת filter, כאשר אנו מכניסים אליה רק סרטים שנצפו על ידי צופים דומים שגם לא נצפו ע"י צופה א' וגם סומנו עם הדירוג לייק (כלומר, כל סרט שקיבל דירוג לייק יסומן ב1).
לטבלה השנייה אנו מכניסים מתוך הטבלה viewedBy גם כן סרטים שנצפו על ידי צופים דומים, ושצופה א' לא צפה בהם, אלא שכאן מספר הלייקים יהיה 0 קבוע לכל הסרטים, והמספר 0 יהיה המספר הקבוע לעמודת ה-filter (כלומר, כל סרט שלא דורג יסומן ב0).
כאמור, נבצע UNION לשתי הטבלאות א' ו-ב', ונקרא לטבלה המאוחדת t1.
לסיום, בוחרים מתוך t1 הזאת את העמודות הבאות: מזהה הסרט, סכום דירוגי הלייק וסכום ה-filter, ואז מקבצים לפי מזהה הסרט, וממיינים ראשית לפי הדירוג בסדר יורד, לאחר מכן לפי כמות הלייקים בסדר יורד, ואז לפי מזהה הסרט בסדר עולה.

הסבר לבחירת העמודות ולמיון: מזהה הסרט היא העמודה אותה בסופו של דבר נרצה להחזיר בשאילתה, קיבוץ לפי מזהה הסרט + סכום הפילטר יסירו כפילויות של סרטים מהמערכת, שכן סרט עם filter=1 הוא סרט שנרצה שיופיע ראשון, ואם הסרט מופיע גם כאשר filter=0 הוא ישאר עם ערך filter=1, ובגלל הקיבוץ יוסרו כפילויות. סכום דירוגי הלייק למעשה יתן את הסכום האמיתי עבור הסרטים מהטבלה הראשונה, אותם נרצה שיופיעו ראשונים, וסכום שהוא 0 עבור סרטים אחרים. כלומר, תוספת של 0 לא תשפיע על סרטים מהטבלה הראשונה, וכך נוודא שהסרטים אכן ימוינו לפי מספר הלייקים (כאמור - קודם לפי הדירוג עם ה-filter ורק אחר כך לפי מספר הלייקים). לבסוף אנו מבצעים מיון בסדר עולה של מזהי הסרטים - וכך למעשה ביצענו את המיון כפי הנדרש בשאלה.
הערה: אם אין מספיק סרטים שדורגו עם לייק מתוך הסרטים של צופים דומים שצופה א׳ אינו ראה, נכניס סרטים ללא דירוג לייק של צופים דומים וצופה א׳ אינו ראה - לפי סדר עולה של מזהה הסרט.

- getConditionalRecommendations: נסמן את הצופה עליו שואלים בתור צופה א׳, ואת הסרט עליו שואלים בתור סרט א׳.
השאילתה הזאת זהה כמעט לחלוטין לשאילתה הקודמת, למעט שינוי אחד: אנו רוצים לעבוד עם similarRankers במקום עם similarViewers, ולכן לאחר שנקבל את הרשימה של similarViewers לאחר קריאה לשאילתה המחזירה את הצופים הדומים, ניצור טבלה שנקראת similarRankers באופן הבא:
נקבל את הדירוג של צופה א׳ את סרט א׳. אם צופה א׳ לא צפה בסרט א׳ או שלא קיים דירוג עדיין של צופה א׳ את סרט א׳, נסיים את השאילתה ונחזיר תוצאה ריקה.
אחרת, נסנן מתוך similarViewers את הצופים שדירגו בלייק את סרט א': עבור כל צופה מהרשימה similarViewers נבדוק אם הוא אכן צפה בסרט. אם לא, לא נכניס אותו לטבלת similarRankers. אם כן, נבדוק שהדירוג אינו null וגם שהדירוג זהה לדירוג שנתן צופה א׳ לסרט א׳. רק אם שני התנאים מתקיימים - יכנס הצופה לרשימה similarRankers. מעתה והלאה - ההמשך זהה לגמרי לשאילתה הקודמת, עם יצירת שתי הטבלאות וביצוע ה-UNION ביניהן.