שם: צליל ויצמן שם: טופז סופר
ת"ז: 322590340 ת"ז: 208279513

System Documentation:

Database schema structure:

1. **Name of the table**: Movies
   Scheme: Movies(movie_id, title, release_year, runtime, tagline, overview, popularity, budget, revenue)
   Types and constraints:
   movie_id- int, not null and unique
   title -varchar(255)
   release_year- int
   runtime- int
   tagline- text
   overview- text
   popularity- float
   budget- bigint
   revenue- bigint

2. **Name of the table:** Actors
   Scheme: Actors(actor_id, actor_name)
   Types and constraints:
   actor_id- int, not null and unique
   actor_name- varchar(255)

3. **Name of the table:** MovieActors
   Scheme: MovieActors(movie_id (FK to Movies.movie_id), actor_id (FK to Actors.actor_id))
   Types and constraints:
   actor_id- int
   movie_id- int
   the combination of actor_id and movie_id is unique and not null

4. **Name of the table:** Genres
   Scheme: Genres(genre_id, genre_name)
   Types and constraints:
   genre_id- int, not null and unique
   genre_name- varchar(100)

5. **Name of the table:** MovieGenres
   Scheme: MovieGenres(movie_id (FK to Movies.movie_id), genre_id (FK to Genres.genre_id))
   Types and constraints:
   movie_id- int
   genre_id- int
   the combination of actor_id and movie_id is unique and not null

שם: צליל ויצמן שם: טופז סופר
ת"ז: 322590340 ת"ז: 208279513

6. **Name of the table:** Directors
Scheme: Directors(director_id, director_name)
Types and constraints:
director_id- int, not null and unique
director_name- varchar(255)

7. **Name of the table:** MovieDirectors
Scheme: MovieDirectors(movie_id (FK to Movies.movie_id), director_id (FK to Directors.director_id))
Types and constraints:
movie_id- int
director_id- int
the combination of actor_id and movie_id is unique and not null

8. **Name of the table:** ProductionCompanies
Scheme: ProductionCompanies(company_id, company_name)
Types and constraints:
company_id- int, not null and unique
company_name- varchar(255)

9. **Name of the table:** MovieProductionCompanies
Scheme: MovieProductionCompanies(movie_id (FK to Movies.movie_id), company_id (FK to ProductionCompanies.company_id))
Types and constraints:
movie_id- int
company_id- int
the combination of actor_id and movie_id is unique and not null

10. **Name of the table**: MoviesImdb
Scheme: MoviesImdb(movie_id (FK to Movies.movie_id), imdb_id)
Types and constraints:
movie_id- int, not null and unique
imdb_id- varchar(15)

שם: צליל ויצמן שם: טופז סופר
ת"ז: 322590340 ת"ז: 208279513

Reasoning for the chosen database design

תחילה, לצורך בניית הטבלאות ביצענו פענוח ואבחון של העמודות בקובץ ה-csv ובהתאם לכך בחרנו את העמודות שסיפקו מענה על צרכי התוכנית. טרם השימוש בדאטה, הפכנו NAN ל-NULL ומחקנו שורות כפולות שהתבססו על מזהה (ID) כפול, שכן הוא משמש כמפתח ראשי.

גזרנו את הנתונים מתוך ה-csv ל- DataFrame (df) ראשי בשם Movies וממנו יצרנו תתי df's.
באמצעות צורת בניית הטבלאות, אנו משמרים את עקרונות ה-normalization וזאת כדי לשמר לוגיקה ומניעת עודף מידע.
כמו כן, בקובץ ה-csv , עמודות כמו director ו-genres בעלות מבנה מהצורה " XXXX|XXXX".
כלומר, כל פרט מופרד ב-"|". לצורך הרצת שאילתות full_text וניתוח הנתונים "פירקנו" את הפרטים של כל עמודות אלו עבור כל סרט והכנסנו לטבלאות רלוונטיות בהתאם ליחסים בין הישויות שיצרנו תוך שמירה על כללי ה-normalization (כפי שתואר מטה עבור כל טבלה).
כל טבלה למעשה מייצגת ישות בודדת והקשר בין הישויות בא ליידי ביטוי ב-FK כפי שצוין בסכמה.
נציין כעת את הלך המחשבה עבור בניית כל טבלה תחת עין ביקורתית:

- <u>בניית טבלת Movies</u>
טבלה זו מכילה את המידע הספציפי עבור כל סרט. לדוגמה, title, release_year ועוד. כל סרט מוגדר באופן ייחודי ע"י PK שהוא movie_id כדי למנוע רשומות כפולות. אם נבחן בעין ביקורתית את טבלה זו, עולה שייתכן והיה צריך לשקול את פיצול טבלת Movies . זאת משום שהיא מכילה מידע רב לכל רשומה ולכן שאילתות שנריץ עם טבלה זו עלולות להיות לא יעילות.
עם זאת, מרבית השאילתות שכתבנו ביצעו חיפוש בעיקר על תתי הטבלאות, טרם חיפוש ב-Movies. לכן, נמנעה פגיעה משמעותית ביעילות השאילתות.
כמו כן, פיצול הטבלה המרכזית לטבלה ראשית ומשנית אמנם מאפשר את אחסון 'הנתונים העיקריים' של כל אחד מהסרטים, לדוגמה: מזהה, כותרת וסקירה כללית בטבלה הראשית.
בטבלה המשנית, נוכל לאחסן את המאפיינים הנותרים: לדוגמה, popularity, תקציב. גישה זו יכולה להביא ליתרונות במונחים של עלות I/O. זאת באמצעות הפרדה בין 'הנתונים העיקריים', שאליהם ניתן לגשת לעתים קרובות, לבין שאר המאפיינים. בכך נוכל להפחית את שכיחות שליפת רשומות גדולות יותר בכל פעם, מה שעלול להפחית את תיקרת ה- I/O.
אך, אלטרנטיבה זו פחות טובה עבור הצגת המידע שכן אם נפצל את הטבלה המרכזית, תיתכן פגיעה בבהירות ובהבנת הדאטה. בנוסף, פיצול הטבלה המרכזית עלול להוביל לתחזוקה כפולה עבור כל הוספה/ מחיקה של רשומה. על כן, הבחירה ביצירת טבלה מרכזית משפרת את הבהירות והקריאה ונכונה יותר עבור תחזוקת מבנה הנתונים לטווח הארוך.
יתר על כן, כדאי לשקול הוספת שדות ביקורת כמו created_at, updated_at.

- <u>בניית טבלת Actors וטבלת MovieActors:</u>
טבלת Actors- מכילה את שמות כלל השחקנים שמופיעים בכלל הסרטים, כאשר עבור כל שחקן ישנו actor_id ייחודי (=PK) שיצרנו. הטבלה מאפשרת טיפול בעודף מידע, שכן חלק מהשחקנים מופיעים במספר סרטים. מבחינה בעין ביקורתית עולה כי כדאי היה לאסוף מידע נוסף על השחקנים כגון, מין, גיל ועוד.

טבלת MovieActors- משמרת את היחס של many to many בין שחקנים לסרטים. בטבלה זו הקומבינציה של actor_id ו-movie_id מהווה PK ובכך מאפשרת למנוע מצב בו שחקן יהיה שייך לאותו סרט מספר פעמים וכן שחקן לא יכול להיות שייך לסרט שלא קיים.

הרעיון מאחורי יצירת הטבלאות הללו נובע מכך שלסרטים רבים יש שחקנים רבים וכן שחקנים רבים משותפים לסרטים רבים. מבחינה בעין ביקורתית עולה כי לכאורה הטבלה אינה לוקחת בחשבון את האפשרות ששחקן משחק ב-2 תפקידים שונים באותו הסרט. יחד עם זאת, לא מסופק ב-csv מידע על תפקידי השחקנים, אלא רק על שחקני הסרט שהשתתפו בו. לכן, נמליץ לפתור סוגיה זו באמצעות איסוף מידע נוסף בנושא והתאמת הטבלה בהתאם לדאטה החדש.

- <u>בניית טבלת Genres וטבלת MovieGenres:</u>
טבלת Genres- מכילה את שמות כלל הז'אנרים של הסרטים, כאשר עבור כל ז'אנר ישנו genre_id ייחודי (=PK). הטבלה מאפשרת טיפול בעודף מידע, שכן ישנם ז'אנרים שמופיעים

שם: צליל ויצמן שם: טופז סופר
ת"ז: 322590340 ת"ז: 208279513

בכמה סרטים שונים. ניתן היה לשקול חלוקה של ז'אנרים ותתי ז'אנרים- לדוגמה עבור קומדיה, תת הז'אנר שלה יהיה קומדיה רומנטית.

טבלת MovieGenres- משמרת את היחס של many to many בין ז'אנרים לסרטים. בטבלה זו הקומבינציה של genre_id ו-movie_id מהווה PK ובכך מאפשרת למנוע מצב בו ז'אנר יהיה משויך לאותו סרט מספר פעמים וכן ז'אנר לא יכול להיות שייך לסרט שלא קיים.
הרעיון מאחורי יצירת הטבלאות הללו נובע מכך שלסרטים רבים יש ז'אנרים רבים וכן ז'אנרים רבים משותפים לסרטים רבים.

- <u>בניית טבלת Directors וטבלת MovieDirectors:</u>
טבלת Directors- מכילה את שמות כלל הבמאים של הסרטים, כאשר עבור כל במאי ישנו director_id ייחודי (=PK). הטבלה מאפשרת טיפול בעודף מידע, שכן ישנם במאים שמופיעים בכמה סרטים שונים. גם כאן, כדאי היה לאסוף מידע נוסף על הבמאים כגון, מין, גיל ועוד.

טבלת MovieDirectors- משמרת את היחס של many to many בין במאים לסרטים. בטבלה זו הקומבינציה של director_id ו-movie_id מהווה PK ובכך מאפשרת למנוע מצב בו במאי יהיה שייך לאותו סרט מספר פעמים וכן במאי לא יכול להיות שייך לסרט שלא קיים.

הרעיון מאחורי יצירת הטבלאות הללו נובע מכך שלסרטים רבים יש במאים רבים וכן במאים רבים משותפים לסרטים רבים. צורת בניית טבלה זו לוקחת בחשבון שעבור סרט כלשהו ייתכן והיה יותר מבמאי אחד וכן ייתכן שבמאי היה משותף לכמה סרטים.

- <u>בניית טבלת ProductionCompanies וטבלת MoviesProductionCompanies:</u>
טבלת ProductionCompanies- מכילה את שמות כלל חברות ההפקה של הסרטים, כאשר עבור כל חברה ישנו company_id ייחודי (=PK). הטבלה מאפשרת טיפול בעודף מידע, שכן ישנם חברות הפקה שמופיעות בכמה סרטים שונים.

טבלת MoviesProductionCompanies- משמרת את היחס של many to many בין חברות הפקה לסרטים. בטבלה זו הקומבינציה של company_id ו-movie_id מהווה PK ובכך מאפשרת למנוע מצב בו חברת הפקה תהיה שייכת לאותו סרט מספר פעמים וכן חברת הפקה לא יכולה להיות שייכת לסרט שלא קיים.

הרעיון מאחורי יצירת הטבלאות הללו נובע מכך שלסרטים רבים יש חברות הפקה רבות וכן חברות הפקה רבות משותפות לסרטים רבים. צורת בניית טבלה זו לוקחת בחשבון שעבור סרט כלשהו ייתכן והייתה יותר מחברת הפקה אחת וכן ייתכן שחברת הפקה הייתה משותפת לכמה סרטים.

- <u>בניית טבלת MoviesImdb</u>
ממפה כל סרט למזהה ב-IMDB לצורך הפנייה חיצונית. מאפשר למשתמש להשיג מידע חיצוני על הסרט באמצעות שימוש במזהה הייחודי של ה-IMDB. כמו כן, בטבלה זו יופיעו בהכרח סרטים שנמצאים בבסיס הנתונים שכן movie_id מהווה PK.
על כן, עיצוב זה מונע כפילויות, מוודא עקביות ומקטין את הסיכוי לאנומליות בדאטה.
כל סכמה תואמת לעקרונות הנורמליזציה שכן כל עמודה מכילה ערכים אטומיים, אין קבוצות שחוזרות על עצמן וכל המשתנים שאינם PK תלויים ב-PK.

מעבר ליתרונות שציינו במהלך סקירת הטבלאות, במבט על היתרונות בעיצוב DB זה הם: ראשית, הימנעות ממידע מיותר. באמצעות נרמול הנתונים, שכיחות הכפילויות הינה מינימלית. לדוגמה, שמות שחקנים, ז'אנרים או פרטי חברת הפקה מופיעים פעם אחת בלבד.
שנית, תוקף ושלמות הנתונים. קשרים בין טבלאות מוגדרים באמצעות מפתחות זרים ואילוצים, מה שמבטיח שיוכיים תקפים.
שלישית, scalability. הסכימה נועדה לטפל במערכות יחסים many to many, ובכך מאפשר התאמת קנה מידה ככל שמסד הנתונים גדל. יתרון נוסף הינו, שימוש יעיל עבור שאילתות. מערכות יחסים שבאות ליידי ביטוי באמצעות "טבלאות צומת" (למשל, MovieActors, MovieGenres) מאפשרות ביצוע יעיל של שאילתת יחסים.

שם: צליל ויצמן שם: טופז סופר
ת"ז: 322590340 ת"ז: 208279513

בעוד שהעיצוב המנורמל הנוכחי שלנו הוא אופטימלי עבור שלמות נתונים ומזעור מידע עודף/ מיותר, גישות חלופיות שניתן היה לשקול (ביניהן דה-נורמליזציה סלקטיבית או שימוש ב-view) היו עשויות לפשט שאילתות ולשפר את הביצועים עבור מקרי שימוש ספציפיים.
לסיכום, עיצוב זה מתאים היטב לטיפול בנתונים הקשורים לסרטים, תומך בגמישות ומבטיח עקביות בכל הישויות והקשרים.

Database optimization

כדי לשפר אופטימיזציה, נעשה שימוש באינדקסים על מנת לייעל את ריצת השאילתות:

ב-query_1, התוצאות מסודרות בסדר יורד לפי ה-popularity. ע"י יצירת אינדקס של עמודת popularity, השאילתה יכולה לרוץ ביעילות, כך שהיא מסדרת את הרשומות בהתאם לנדרש במהירות רבה יותר. כמו כן, כדי לאפשר חיפוש באמצעות מילת מפתח, ב-name יישמנו full-text אינדקס. האינדקס משפר את הסיבוכיות והיעילות של פעולת החיפוש ומאפשר החזרה מהירה של מידע רלוונטי. על כן, האינדקסים משפרים את ביצועי השאילתה ע"י זיהוי מהיר של סדר הצגת התוצאות הנדרש.

ב-query_2, התוצאות מסודרות בסדר יורד לפי ה-release_year, כאשר הסרטים העדכניים מופיעים קודם לכן. ע"י יצירת אינדקס של עמודת release_year, השאילתה יכולה לרוץ ביעילות, כך שהיא מסדרת את הרשומות בהתאם לנדרש במהירות רבה יותר. כמו כן, כדי לאפשר חיפוש באמצעות מילת מפתח, ב-overview יישמנו full-text אינדקס. האינדקס משפר את הסיבוכיות והיעילות של פעולת החיפוש ומאפשר החזרה מהירה של מידע רלוונטי. על כן, האינדקס משפר את ביצועי השאילתה ע"י זיהוי מהיר של סדר הצגת התוצאות הנדרש.

ב-query_3, התוצאות מסודרות בסדר יורד לפי ה-budget. ע"י יצירת אינדקס של עמודת budget, השאילתה יכולה לרוץ ביעילות, כך שהיא מסדרת את הרשומות בהתאם לנדרש במהירות רבה יותר. הסידור המהיר מאפשר את הצגת 5 הרשומות המובילות במהירות (שכן, בשאילתה הגבלנו ל-5 הרשומות הראשונות). שימוש באינדקס נוסף הינו ביצירת האינדקס genre_id. האינדקס מאפשר גישה מהירה לחיבור הטבלאות Genres ו-MovieGenres ובכך התאמת שם הז'אנר לסרט שאליו משויך במהירות. על כן, האינדקס משפר את ביצועי השאילתה ע"י זיהוי מהיר של סדר הצגת התוצאות הנדרש.

ב-query_5, נעשה שימוש באינדקס באמצעות יצירת actor_id. האינדקס מאפשר גישה מהירה לחיבור הטבלאות Actors ו-MovieActors ובכך התאמת שם השחקן לסרט שאליו משויך במהירות. בשאילתה זו, החיפוש נעשה על ידי שם ספציפי באמצעות '=' ולא מילת מפתח ולכן לא נעשה שימוש בfull-text אינדקס אך החיפוש יעיל. על מנת למצוא את הז'אנר הנפוץ ביותר עבור השחקן שמצאנו, השתמשנו באותה שאילתה ללא שכפול קוד (שימוש ב WITH) וחישבנו מספר המופעים של כל ז'אנר. השארנו רק את הרשומה עם הז'אנר הנפוץ ביותר.

ב- query_7, התוצאות מסודרות לפי ספירת הסרטים (company_counts=) של כל חברת הפקה בסדר יורד. ()ROW_NUMBER יוצר דירוג רציף (=מעין אינדקס) עבור החברות, החל מ-1 עבור החברה עם הספירה הגבוהה ביותר. בכך, מתאפשר מציאת 5 החברות המובילות ביותר בצורה מהירה ויעילה. כמו כן, שימוש באינדקס נוסף הינו ביצירת האינדקס company_id. האינדקס מאפשר גישה מהירה לחיבור הטבלאות ProductionCompanies ו-
MoveiProductionCompanies ובכך מייעל את פעולת COUNT(mpc.movie_id) ועוזר במיון (ORDER BY movie_count DESC) בפונקציית החלון.
על כן, השימוש באינדקסים משפר את ביצועי השאילתה ע"י זיהוי מהיר של סדר הצגת התוצאות הנדרש. (ניתן למצוא עוד בנושא זה ב-"main queries 5" תחת query_7).
שאילתה זו גם עושה שימוש בפונקציית WINDOW שמובילה גם לאופטימיזציה משמעותית- פירוט על כך ניתן למצוא ב-"main queries 5" תחת query_7.

שם: צליל ויצמן שם: טופז סופר
ת"ז: 322590340 ת"ז: 208279513

Normalization:

בנינו את מסד הנתונים שלנו באמצעות מודל הנורמליזציה כפי שנלמד בכיתה - הסבר נוסף ניתן למצוא ב- "Reasoning for the chosen database design" במסמך.

Window func:

ב-query_7 נעשה שימוש ב-window func כפי שנלמד בכיתה למטרות ייעול. ניתן לראות הסבר מפורט על היתרונות ועל השימוש בו ב-"5 main queries", תחת query_7.

5 main queries

1. query_1- שאילתה זו מחפשת סרטים שהכותרים שלהם מכילים מילת מפתח ספציפית שסופקה על ידי המשתמש. השאילתה נוצרה מתוך מחשבה שמשתמש שמכיר את הסרט או מעוניין לצפות בסרט בנושא מסוים יספק את שמו או מילת מפתח הקשורה לשמו.
השתמשנו ב MATCH(title) AGAINST (%s) על מנת לבדוק אם ה-title מכיל את מילת המפתח. כדי לאפשר חיפוש באמצעות מילת מפתח, ב-name יישמנו full-text אינדקס. האינדקס משפר את הסיבוכיות והיעילות של פעולת החיפוש ומאפשר החזרה מהירה של מידע רלוונטי.התוצאות מסודרות לפי פופולריות בסדר יורד כך שהכותרים הפופולריים ביותר מופיעים ראשונים. כדי לאפשר הצגת תוצאות בצורה מהירה ויעילה הוספנו גם אינדקס עבור popularity (כפי שתואר ב-database optimization).

2. query_2- שאילתה זו מחפשת סרטים לפי מילות מפתח ב-overview שלהם.
שאילתה זה נוצרה בעקבות המחשבה שמשתמש שמחפש סרט העוסק בנושא מסוים או לחלופין מכיר את תקציר הסרט או את תוכנו ואינו זוכר את שמו, יוכל למצוא את שם הסרט בקלות. כדי לאפשר חיפוש באמצעות מילת מפתח, ב-overview יישמנו full-text אינדקס. האינדקס משפר את הסיבוכיות והיעילות של פעולת החיפוש ומאפשר החזרה מהירה של מידע רלוונטי.
בדומה לשאילתה 1, בדקנו האם הקלט הינו חלק ממילת מפתח בעמודת ה-overview.
התוצאות מסודרות לפי שנת יציאה בסדר יורד, ומציגות תחילה את הסרטים העדכניים ביותר.
כדי לאפשר הצגת תוצאות בצורה מהירה ויעילה הוספנו גם אינדקס עבור release_year (כפי שתואר ב-database optimization).

3. query_3- שאילתה זו מוצאת את 5 הסרטים המובילים עם התקציבים הגבוהים ביותר בז'אנר ספציפי שנבחר על ידי המשתמש. הרעיון מאחורי שאילתה זו היה שמשתמשים אשר יחפשו סרטים עם ז'אנר מסוים יעדיפו סרט שנעשה בתקציב גבוהה יותר מאשר סרט שנעשה בתקציב נמוך (כי הדבר עלול להעיד על איכות הסרט כמו תפאורה, שחקנים וכו'), או שאילתה זו עושה שימוש בטבלאות Movies (מכיל פרטי סרט כמו תקציב), MovieGenres (מקשר סרטים לז'אנרים שלהם) ו- Genres (מכיל שמות ז'אנרים ומזהים). סינון הסרטים מתבצע לפי שם הז'אנר שצוין (למשל, "אקשן"). שקלנו את הוספת האינדקס עבור genre_name, אך משום שיש מספר יחסית קטן של ז'אנרים (=כ-20), הוחלט כי אין צורך במקרה זה. במידה ויוסיפו ז'אנרים כדאי לשקול נושא זה שוב.
התוצאות ממוינות לפי עמודת התקציב בסדר יורד. כדי לאפשר הצגת תוצאות בצורה מהירה ויעילה הוספנו גם אינדקס עבור budget (כפי שתואר ב-Database optimization).
השאילתה מגבילה את הפלט ל-5 שורות.

4. query_5- שאילתה זו מוצאת את כל הסרטים שבהם מופיע שחקן שסופק ע"י המשתמש. השאילתה נוצרה מתוך מחשבה שייתכן ומשתמש ירצה לראות סרט שמופיע בו שחקן שהוא אוהד. בנוסף, משום שהמשתמש אוהד את השחקן הזה הוא יתעניין בפרטים נוספים לגביו ולכן השאילתה מחזירה גם את הז'אנר הנפוץ עבור

שם: צליל ויצמן שם: טופז סופר
ת"ז: 322590340 ת"ז: 208279513

השחקן הספציפי ומספר הסרטים שלו בז'אנר זה. השאילתה מבצעת join בטבלת Movies (מכילה את פרטי סרטים) עם טבלת MovieActors (מקשר בין סרטים לחברי שחקנים) ובין MovieActors ל-Actors (המכילה את שמות השחקנים).
לאחר מכן, סופרת עבור כל סרט שקיבלנו לפי הז'אנרים מתאימים, כמה סרטים מצאנו מכל סוג ז'אנר. החיפוש הינו עבור שמו המלא של השחקן ולא באמצעות מילת מפתח.

5. query_7- השאילתה מוצאת את חמשת חברות ההפקה המובילות, המופיעות הכי הרבה בטבלת Movies, באמצעות פונקציית חלון. הרעיון מאחורי השאילתה נובע בעיקר מנקודת מבט של ניתוח נתונים ויאפשר למצוא את חמשת חברות ההפקה המובילות. פלט השאילתה הוא רשימת tuples של company_name ו-movie_count.
הביטוי company_counts, מקבץ את ספירת הסרטים עבור כל חברת הפקה באמצעות COUNT(mpc.movie_id) ומבצע group by לפי company_name.
זהו שלב קדם לחישוב סך הסרטים המשויכים לכל חברת הפקה.
הביטוי ROW_NUMBER() OVER, מקצה דירוג ייחודי לכל חברת הפקה בהתבסס על ספירת הסרטים שלה בסדר יורד.
ROW_NUMBER() יוצר דירוג רציף עבור החברות, החל מ-1 עבור החברה עם הספירה הגבוהה ביותר.
לצורך מציאת חמשת החברות המובילות, השאילתה החיצונית מסננת עבור שורות שבהן דירוגן <= 5, ומבטיחה שרק 5 החברות המובילות מופיעות.
יתרון בולט העולה משימוש בפונקציית WINDOW הינו שבפונקציית חלון בניגוד לפעולת aggregation, חישובים כמו ROW_NUMBER() מבוצעים על פני כלל הנתונים, אך מבלי לכווץ שורות לתוצאות מצטברות. לכן, פונקציות WINDOW גמישות יותר, מה שמאפשר לשמור נתונים ברמת השורה תוך כדי ביצוע חישובים מדורגים או פעולות אחרות.
כדי להפוך את השאילתה ליעילה: יצרנו אינדקס על company_id (ב-ProductionCompanies ו-MovieProductionCompanies). בזכות כך, הדבר מאפשר גישה מהירה לחיבור בין הטבלאות הללו. יצירת אינדקס על movie_id (ב-MovieProductionCompanies) מייעל את פעולת COUNT(mpc.movie_id).
יצירת אינדקס על company_name (ב-ProductionCompanies) עוזר במיון (ORDER BY movie_count DESC) בפונקציית החלון.

- <u>הערה:</u> ישנן שאילתות נוספות שיצרנו לצורך הנגשת המידע בצורה פשוטה ויעילה, הרחבת יכולות המשתמש ומתן כלים אינפורמטיביים עבור המשתמש. השאילתות נועדו על מנת לנתח את המידע בטבלאות בצורה אינפורמטיבית ולהוביל לתובנות עסקיות/ לתובנות אצל המשתמש.

<u>Outline and code structure</u>

create_db_script.py- קובץ זה מכיל את הפונקציה create_database(conn, cursor, db_name) אשר אחראית על יצירת בסיס הנתונים אם הוא לא קיים. הפונקציה מבצעת בדיקה, האם מסד הנתונים כבר קיים. פונקציה נוספת היא drop_database(conn, cursor, db_name) שמוחקת את בסיס הנתונים אם הוא כבר קיים. הפונקציה האחרונה בקובץ זה היא create_db(conn, cursor) שיוצרת את טבלאות ה-SQL לפי הסכמה ומדווחת במקרה של שגיאה בעת יצירתן.
<u>הערה:</u> קובץ הקוד מכיל הסברים והערות עבור כל פונקציה ומשתניה.

api_data_retrieve.py- קובץ זה מכיל משתנים גלובליים שנועדו לצורך הליך הכנסת הדאטה. כמו כן, כדי לתחזק את המידע וביצוע מעקב על הכנסת הדאטה, נעשה שימוש במילונים tables ו-data.
בקובץ זה יש את הפונקציה insert_table(conn, cursor, table) שמכניסה את הנתונים מ-df של פנדה באמצעות הכנסות SQL. הפונקציה מטפלת בשגיאות של הכנסה של רשומה כפולה באמצעות דילוג על רשומות אלה וכן מעדכנת על הכנסת דאטה מוצלחת.

שם: צליל ויצמן שם: טופז סופר
ת"ז: 322590340 ת"ז: 208279513

פונקציה נוספת בקובץ זה הינה process_text(col_name) שנועדה לטפל בעמודות הטקסט שמקורן ב-csv. לדוגמה, עמודת genre, production_company ועוד. פונקציה זו למעשה מפרידה בין כל פרט בטקסט ומכניסה אותו בנפרד לשורה הרלוונטית (היא למעשה מפרקת את תוכן הטקסט לחלקיו).
הפונקציה האחרונה בקובץ זה היא פונקציית insert_data(conn, cursor) שקוראת למעשה לשתי הפונקציות שתיארנו מעל עם המשתנים הרלוונטיים לצורך הרצתן.
<u>הערה:</u> קובץ הקוד מכיל הסברים והערות עבור כל פונקציה ומשתניה.

queries_db_script.py- קובץ זה מכיל את הפונקציות query_x שמכילה את מחרוזת השאילתה שאותה נוכל להריץ בהמשך. ישנן בסה"כ 7 שאילתות כאשר עבור כל אחת ישנן הערות והסברים של השאילתה.

queries_execution- קובץ זה מכיל פונקציות running_query_x, אשר אחראיות על ביצוע מחרוזת השאילתה המוחזרת מ-query_x, והדפסת התוצאות. ה-main של הקובץ מכיל התחברות לשרת, אפשרות לבנייה מחדש של הDB והנתונים והרצות לדוגמה של השאילתות (עם inputs ספציפיים לשאילתה הכוללים קלטים ריקים או שאינם מניבים תוצאות מהמסד נתונים על מנת להדגים שהמקרים האלו מטופלים)
<u>הערה:</u> ניתן לבטל את ההערה של בלוק האתחול מחדש בחלק העליון של ה-main על מנת לאתחל מחדש את ה-DB והנתונים.

API usage:

החיבור לקובץ ה CSV הינו באמצעות api_data_retrieve.py שמבצע טיפול בנתונים והכנסתם לDATABASE שיצרנו. בחרנו את העמודות הרלוונטיות עבור מטרות האפליקציה.
הקובץ queries_db_script.py מכיל את כל השאילתות ואופן השימוש בנתונים, מתועד ומכיל את כל המידע הרלוונטי לגביהן. השימוש בשאילתות אלו מודגם בפונקציית ה- main שלנו ב-queries_execution.py.

General flow of the application:

ניתן לחקור את ממשק המשתמש של היישום שלנו ב-user manual, בו ניתן באמצעות לחיצת כפתור לראות כיצד האפליקציה שלנו פועלת. באמצעות לחיצות כפתור ניתן לנווט לשקופית המתאימה. בסף הקובץ הוספנו הסבר נוסף קצר לגבי הממשק והפונקציות.