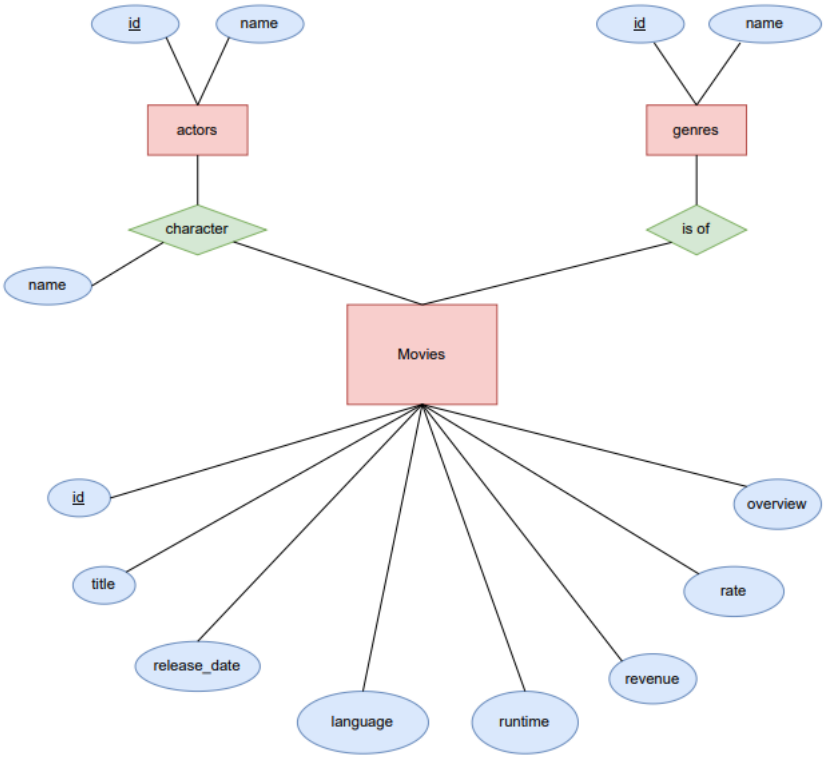
Software Docs

- ER diagram



- מבנה ה DB:
  - טבלאות
  - Primary key
  - Foreign keys
  - אינדקסים: BTREE, FULLTEXT

- Movies (id, title, rate, revenue, release_date, overview, runtime, spoken_language)
- Genres (id, name)
- Actors (id, name)
- Characters (id, name, movie_id, actor_id)
  FKs: movie_id → movies.id , actor_id → actors.id
- Movies_genres (movie_id, genre_id)
  FKs: movie_id → movies.id , genre_id → genres.id

- <u>עקרונות מנחים בבניית ה DB:</u>
  - טבלאות קטנות ככל הניתן - בהתאם למודל הרלציוני.
    טבלאות קטנות תורמות הן מבחינת קריאות והן מבחינת יעילות, וחוסכות בזיכרון ובפעולות IO.
  - לכל טבלה הגדרנו PK מטיפוס int, ערך מספרי מאפשר לחסוך בזיכרון. בנוסף הגדרנו אינדקס מסוג BTREE על כל PK כדי לאפשר חיפושים יעילים ולבדוק ביעילות key violations בהכנסות לטבלה.
  - בין טבלאות קשורות הוגדרו FKs על שדות שהם PK, שדות אלה כבר מאונדקסים וכך יתאפשר ביצוע פעולות על הDB ביעילות.
  - את ערכי ה runtime, spoken_language בחרנו לייצג כשדות בטבלת הסרטים, במקום בטבלה נפרדת לכל אחד מהם (את משך הסרט היינו מייצגים בתור קטגוריות של קצר, בינוני, ארוך). עשינו זאת כדי לחסוך בזיכרון שהיה דרוש כדי לתחזק שתי טבלאות נוספות, וגם כי מדובר בערכים "קטנים", השפה מיוצגת כמחרוזת של מילה אחת ומשך הסרט מיוצג כint (~ 50-200 דקות).
    במצב כזה הFKs שהיינו צריכים לשמור בטבלת הסרטים היו תופסים כמות זיכרון דומה לערכי השדות עצמם.
  - בטבלת movies_genres בחרנו שלא להוסיף עמודת PK חדשה אלא להשתמש בשתי העמודות הקיימות שכבר מאונדקסות ובכך לחסוך בזיכרון.

- <u>אופטימיזציה:</u>
  - אינדקס BTREE - הוספנו עבור כל PK ועבור כל FK המקשרים בין טבלאות, לצורך ייעול חיפושים ופעולות JOIN.
    בנוסף, אינדקסנו את שדה הדירוג של הסרטים כי השתמשנו בו בשאילתות רבות (לצורך סידור הפלט למשל).
  - אינדקס FULLTEXT - הוספנו עבור השדות title בטבלת הסרטים ו name בטבלת השחקנים, משום שמתבצעים עליהן חיפושים בשאילתות הטקסט.
    - עבור השדה name בטבלת הז'אנרים (למרות שמבצעים עליו חיפוש) לא הוספנו אינדקס כי מדובר בטבלה קטנה בעלת 19 שורות וייתכן כי העלות של האינדקס עצמו תהיה גדולה מהתרומה שלו בעת החיפוש של הז'אנר.

- <u>תיאור השאילתות:</u>

1. Pick_by_movie_title
   - קלט: שם של סרט.
   - פירוט:
     - השאילתה מוצאת עד 10 סרטים דומים לסרט שהזין המשתמש מבחינת דירוג, משך הסרט וז'אנר.
     - הדמיון עבור דירוג: טווח של (rating of input movie - 1 , rating of input movie + 1).
     - הדמיון עבור משך הסרט: טווח של (runtime of input movie - 15 , runtime of input movie + 15).
     - הדמיון עבור ז'אנר: חפיפה של לפחות ז'אנר אחד עם הז'אנרים של סרט הקלט.
   - פלט: שמות הסרטים, יחד עם תקציר, דירוג, שפה, תאריך יציאה לאקרנים, ומשך הסרט.
   - מטרת השאילתה ותרומה לאפליקציה: לאפשר למשתמש שאוהב סרט מסוים לבחור סרט לפי מבחר סרטים הדומים לאותו הסרט, בנוסף השאילתה מחזירה שדות רלוונטיים שיתנו למשתמש מידע על הסרטים שעשוי לעניין אותו.
   - אופטימיזציה:
     - החיפושים (שימוש באופרטור IN) מתבצעים על מפתחות מטיפוס int ובעלי אינדקס BTREE.
     - אינדקס על הדירוג לצורך ביצוע מהיר של range search (על משך הסרט לא הוספנו אינדקס כי ביצענו פעולות על שדה זה בשתי שאילתות בלבד והתמורה של האינדקס לא בהכרח הייתה מספיקה).
     - שימוש באינדקס FULLTEXT על שדה שם הסרט ושימוש ב MATCH() AGAINST() כדי לייעל את חיפוש הסרט שניתן בקלט.
     - השימוש ב IN NATURAL LANGUAGE MODE מאפשר למצוא את הסרט שהמשתמש רצה גם אם הוא הזין שם שלא מופיע באותה צורה ב DB (עד רמה מסוימת): החיפוש מתבצע בצורת case-insensitive, מאפשר סימני פיסוק, מאפשר הופעה של רק חלק מהמילים בשם שהוזן, מתעלם מ stop words וממילים שאורכן קצר מ3 תווים.
     - (פחות אופטימיזציה ויותר למען קריאות הקוד) יצירת VIEW המחפש את הסרט שניתן בקלט בטבלת הסרטים ומחזיר את הרשומה שלו (משמש בהרבה חלקים בשאילתה).
   - עיצוב ה DB:
     - תורם ליעילות השאילתה בכך שהמידע הרלוונטי מחולק ל2 טבלאות שונות קטנות ככל הניתן, ובכך מביא למינימום את מספר פעולות ה IO.
       - הגדרת PKs ו FKs מתאימים לתיאור הקשר בין טבלאות הסרטים והז'אנרים דרך טבלת movies_genres, למעשה על אף שהשאילתה עוסקת בז'אנרים, לא נעשה שימוש בטבלת הז'אנרים בשאילתה זו אלא רק בטבלה movies_genres המקשרת.

2. pick_by_actor
   - קלט: שם של שחקן.
   - פלט: הסרטים שהשחקן שיחק בהם, יחד עם תקציר, דירוג, שפה, תאריך יציאה לאקרנים, ומשך הסרט.
   - מטרת השאילתה ותרומה לאפליקציה: לאפשר למשתמש שאוהב שחקן מסוים לבחור סרט לפי השחקן, בנוסף השאילתה מחזירה שדות רלוונטיים שיתנו למשתמש מידע על הסרטים שעשוי לעניין אותו.
   - אופטימיזציה:
     - JOIN מתבצע לפי מפתחות מטיפוס int ובעלי אינדקס BTREE.
     - שימוש באינדקס FULLTEXT על שדה שם השחקן ושימוש ב MATCH() AGAINST() כדי לייעל את חיפוש השחקן שניתן בקלט.
     - השימוש ב IN NATURAL LANGUAGE MODE מאפשר למצוא את השחקן שהמשתמש רצה גם אם הוא הזין שם שלא מופיע באותה צורה ב DB (עד רמה מסוימת): החיפוש

מתבצע בצורת case-insensitive, מאפשר סימני פיסוק, מאפשר הופעה של רק חלק מהמילים בשם שהוזן, מתעלם מ stop words וממילים שאורכן קצר מ3 תווים.

- עיצוב ה DB:
  - תורם ליעילות השאילתה בכך שהמידע הרלוונטי מחולק ל3 טבלאות שונות קטנות ככל הניתן, ובכך מביא למינימום את מספר פעולות ה IO.
  - הגדרת PKs ו FKs מתאימים לתיאור הקשר בין טבלאות הסרטים והשחקנים דרך טבלת הדמויות.
  -

3. Profitable_movies_by_genre
   - קלט: שם של ז'אנר.
   - פלט: 10 הסרטים הכי רווחיים עבור אותו ז'אנר. השאילתה מחזירה את שמות הסרטים יחד עם הרווחים שלהם, מסודרים בסדר יורד לפי הרווחים.
   - מטרת השאילתה ותרומה לאפליקציה: במידה והמשתמש הינו "חובב דירוגים" ומחפש לצפות בסרט מז'אנר מסוים, הוא יכול לראות אילו סרטים הכי רווחיים עבור אותו ז'אנר ולבחור סרט מתוכם. הסידור של הפלט נעשה לפי הרווחים בסדר יורד כדי שלמשתמש יהיה נוח לבחור את הסרטים הרווחיים ביותר שנמצאים בתחילת הפלט.
   - אופטימיזציה:
     - JOIN מתבצע לפי מפתחות מטיפוס int ובעלי אינדקס BTREE.
     - לא נעשה שימוש באינדקס על שדה שם הז'אנר בטבלת הז'אנרים כי יש רק 19 ז'אנרים ולכן הוא לא יתרום הרבה ואפילו עשוי לעלות יותר ממה שהוא יתרום.
   - עיצוב ה DB:
     - תורם ליעילות השאילתה בכך שהמידע הרלוונטי מחולק ל3 טבלאות שונות קטנות ככל הניתן, ובכך מביא למינימום את מספר פעולות ה IO.
     - הגדרת PKs ו FKs מתאימים לתיאור הקשר בין טבלאות הסרטים והז'אנרים דרך טבלת movies_genres.

4. Top films by year
   - קלט: אין.
   - פירוט:
     - השאילתה מוצאת 10 סרטים בעלי הדירוג הכי גבוה מבחינת סכום של ממוצע הדירוגים של סרטים עבור כל שנה וממוצע הדירוגים של סרטים עבור כל ז'אנר.
   - פלט: 10 הסרטים המובילים מבחינת הקריטריון הנ"ל, הדירוג שלהם, ז'אנר, ממוצע הדירוגים של כל הסרטים שיצאו בשנת שבה יצא הסרט, ממוצע הדירוגים של כל הסרטים מאותו הז'אנר של הסרט.
   - מטרת השאילתה ותרומה לאפליקציה: להמליץ למשתמש על סרטים בצורה מעט יותר מתוחכמת המבוססת על שילוב בין ז'אנר לשנה בה הסרט יצא.
   - אופטימיזציה:
     - JOIN מתבצע לפי מפתחות מטיפוס int ובעלי אינדקס BTREE.
   - עיצוב ה DB:
     - תורם ליעילות השאילתה בכך שהמידע הרלוונטי מחולק ל3 טבלאות שונות קטנות ככל הניתן, ובכך מביא למינימום את מספר פעולות ה IO.
     - הגדרת PKs ו FKs מתאימים לתיאור הקשר בין טבלאות הסרטים והז'אנרים דרך טבלת movies_genres.

5. Pick by Preference
   - קלט: ז'אנר, טווח עבור משך הסרט (minimal runtime, maximum runtime), שפה.
   - פלט: עד 10 סרטים התואמים את הקריטריונים שהמשתמש הזין. השאילתה מחזירה את שמות הסרטים, יחד עם תקציר, דירוג, שפה, תאריך יציאה לאקרנים, ומשך הסרט. הסרטים מוחזרים בסדר יורד לפי דירוג הסרט (מהגבוה לנמוך).

- מטרת השאילתה ותרומה לאפליקציה: לאפשר למשתמש לבחור סרט לצפייה ע"פ קריטריונים שהוא בוחר בעצמו. השאילתה מחזירה שדות רלוונטיים שיתנו למשתמש מידע על הסרטים שעשוי לעניין אותו. הסרטים שמוצעים למשתמש מסודרים לפי דירוג מתוך הנחה שהוא יעדיף לצפות בסרטים בעלי דירוג גבוה קודם.
- אופטימיזציה:
  - החיפושים (שימוש באופרטור IN) מתבצעים על מפתחות מטיפוס int ובעלי אינדקס BTREE.
  - אינדקס על הדירוג לצורך ביצוע מהיר של מיון.
  - לא נעשה שימוש באינדקס על שדה שם הז'אנר בטבלת הז'אנרים כי יש רק 19 ז'אנרים ולכן הוא לא יתרום הרבה ואפילו עשוי לעלות יותר ממה שהוא יתרום.
- עיצוב ה DB:
  - תורם ליעילות השאילתה בכך שהמידע הרלוונטי מחולק ל3 טבלאות שונות קטנות ככל הניתן, ובכך מביא למינימום את מספר פעולות ה IO.
  - הגדרת PKs ו FKs מתאימים לתיאור הקשר בין טבלאות הסרטים והז'אנרים דרך טבלת movies_genres.

6. Mean_revenue_for_genre
- קלט: אין.
- פלט: לכל ז'אנר, השאילתה מחזירה את שם הז'אנר וממוצע הרווחים של הסרטים המשתייכים אליו, מסודרים בסדר יורד.
- מטרת השאילתה ותרומה לאפליקציה: בהמשך לשאילתה הקודמת, במידה והמשתמש הינו "חובב דירוגים" אבל הוא לא מחפש לצפות בסרט מז'אנר מסוים, הוא יכול לראות אילו ז'אנרים הכי רווחיים ולבחור סרט לאחר מכן עבור הז'אנר שבחר. הסידור של הפלט נעשה לפי ממוצע הרווחים בסדר יורד כדי שלמשתמש יהיה נוח לבחור את הז'אנר/ים הרווחיים ביותר שנמצאים בתחילת הפלט.
- אופטימיזציה:
  - JOIN מתבצע לפי מפתחות מטיפוס int ובעלי אינדקס BTREE.
- עיצוב ה DB:
  - תורם ליעילות השאילתה בכך שהמידע הרלוונטי מחולק ל3 טבלאות שונות קטנות ככל הניתן, ובכך מביא למינימום את מספר פעולות ה IO.
  - הגדרת PKs ו FKs מתאימים לתיאור הקשר בין טבלאות הסרטים והז'אנרים דרך טבלת movies_genres.

7. Top_characters_by_actor
- קלט: שם של שחקן.
- פלט: עד 5 דמויות שהשחקן גילם, יחד עם הסרטים הרלוונטיים מסודרים לפי דירוג הסרט המתאים בסדר יורד.
- מטרת השאילתה ותרומה לאפליקציה: לאפשר למשתמש שאוהב שחקן מסוים לראות אילו דמויות הוא גילם, ובהתאם לבחור סרט לצפייה.
- אופטימיזציה:
  - JOIN מתבצע לפי מפתחות מטיפוס int ובעלי אינדקס BTREE.
  - שימוש באינדקס FULLTEXT על שדה שם השחקן ושימוש ב MATCH() AGAINST() כדי לייעל את חיפוש השחקן שניתן בקלט.
  - השימוש ב IN NATURAL LANGUAGE MODE מאפשר למצוא את השחקן שהמשתמש רצה גם אם הוא הזין שם שלא מופיע באותה צורה ב DB (עד רמה מסוימת): החיפוש מתבצע בצורת case-insensitive, מאפשר סימני פיסוק, מאפשר הופעה של רק חלק מהמילים בשם שהוזן, מתעלם מ stop words וממילים שאורכן קצר מ3 תווים.
  - אינדקס על הדירוג מייעל את סידור הפלט.

- עיצוב ה DB:
  - תורם ליעילות השאילתה בכך שהמידע הרלוונטי מחולק ל3 טבלאות שונות קטנות ככל הניתן, ובכך מביא למינימום את מספר פעולות ה IO.
  - הגדרת PKs ו FKs מתאימים לתיאור הקשר בין טבלאות הסרטים והשחקנים דרך טבלת הדמויות.

8. Top_rated_movies_by_language
   - קלט: שפה.
   - פלט: עבור השפה הנתונה, השאילתה מחזירה את 10 הסרטים שקיבלו את הדירוג הגבוה ביותר, ואת הדירוג שלהם, מסודרים בסדר יורד לפי הדירוג.
   - מטרת השאילתה ותרומה לאפליקציה: לאפשר למשתמש לבחור סרט פופולרי בשפה שהוא דובר.
   - אופטימיזציה:
     - אינדקס על הדירוג לצורך ביצוע מהיר של מיון.
     - לא השתמשנו באינדקס על שדה השפה כי מדובר בשאילתה פשוטה.
   - עיצוב ה DB: כל המידע הדרוש לשאילתה זו נמצא בטבלה אחת, ובכך מביא למינימום את מספר פעולות ה IO.

- <u>מבנה הקוד:</u>

הקוד בנוי מ-4 קבצי פייתון מרכזיים שמבצעים 2 תהליכים נפרדים:

- בניית הdb וטעינת המידע:
  תהליך זה מורכב מ-2 קבצים:
  - Api-data-retrieve
  - Create_db_script

הקובץ api-data-retrieve תחילה קורא לפונקציה create_tables מתוך create_db_script אשר אחראית על בניית הdb.

הפונקציה create_tables קוראת קובץ sql אשר מכיל את השאילתות לבניית הdb, מפרידה ביניהן ומריצה כל שאילתא בנפרד, כאשר הבדיקה האם הטבלאות קיימות נעשית בsql על מנת למנוע כפל טבלאות/errors.

אחרי בניית הdb, api-data-retrieve קוראת לפונקציות אשר אחראיות על הכנסת המידע לdb – תחילה הכנסת סרטים וקטגוריות, לאחר מכן שחקנים ולבסוף דמויות. (ראו סעיף api-usage)

- הגדרת השאילתות וביצוען-
  תהליך זה מורכב מ-2 קבצים:
  - queries_db_script – הקובץ בו מוגדרות הפונקציות שמרכיבות את הapi
  - queries_execution- מחברת ג'ופיטר אשר מסבירה ומדגימה את השימוש בכל אחת מפונקציות הapi.

הקובץ queries_execution קורא לפונקציות מתוך queries_db_script בו מוגדרת פונקציה מתאימה לכל שאילתא, אשר קוראת את שאילתת הsql מתוך קובץ txt, עושה execution לשאילתא ומחזירה את התשובה.

- <u>שימוש בAPI</u>

על מנת לקבל את המידע המקיף על הציטוטים הזדקקנו למידע על סרטים, דמויות, שחקנים וציטוטים. לשם כך השתמשנו ב API של TMDB המבוסס על הנתונים של IMDB ובקובץ טקסט המכיל ציטוטים שחולצו גם הם מ IMDB .

- חילוץ הנתונים על הסרטים –

בוצע בפונקציה retrieve_movies.
ניגשנו לניתוב המתאים בAPI והנתונים המתאימים הוכנסו לטבלה movies.

גישה לנתוני הסרטים:

https://api.themoviedb.org/3/movie/{movie_id}?api_key={our_api_key}

וייצוא השדות:

id, title, release date, rate, revenue, genre, overview, spoken_language, runtime

- חילוץ הנתונים על השחקנים –

בוצע בפונקציה retrieve_actors

ניגשנו לניתוב המתאים בAPI והנתונים המתאימים הוכנסו לטבלה actors.

גישה לנתוני השחקנים:

https://api.themoviedb.org/3/person/{person_id}?api_key={our_api_key}

וייצוא השדות id, name

- חילוץ הנתונים על הדמויות –

בוצע בקובץ retrieve_ characters

ניגשנו לניתוב המתאים בAPI והנתונים המתאימים הוכנסו לטבלה characters.

עבור כל שחקן פנינו לניתוב בו מפורטות הדמויות ששיחק

: https://api.themoviedb.org/3/person/{person_id}/credits?api_key={our_api_key}

עבור כל שחקן הוכנסו כל הדמויות שהוא שיחק על סמך השדות movie_id, name שמכילים את ה id של הסרט ואת שם הדמות, שדות אלה הוכנסו לטבלת characters.

- <u>ספריות חיצוניות</u>

השתמשנו בספריית mysql-connector ו requests