# Cinematch

**Our people, your cast.**

Documentations

## מידע כללי

האפליקציה שלנו מנהלת מידע עבור חברות לליהוק שחקנים. האפליקציה מאפשרת גישה לדוחות המסייעים לחברות ליהוק לקשר בין שחקנים לבין חברות הפקה:

- שחקנים יוכלו למצוא חברות הפקה לעבוד עבורן לפי פרמטרים מועדפים.
- חברות ההפקה יוכלו לקבל מידע על שחקנים מתאימים.

## תוכן עניינים

# מבנה הפרויקט

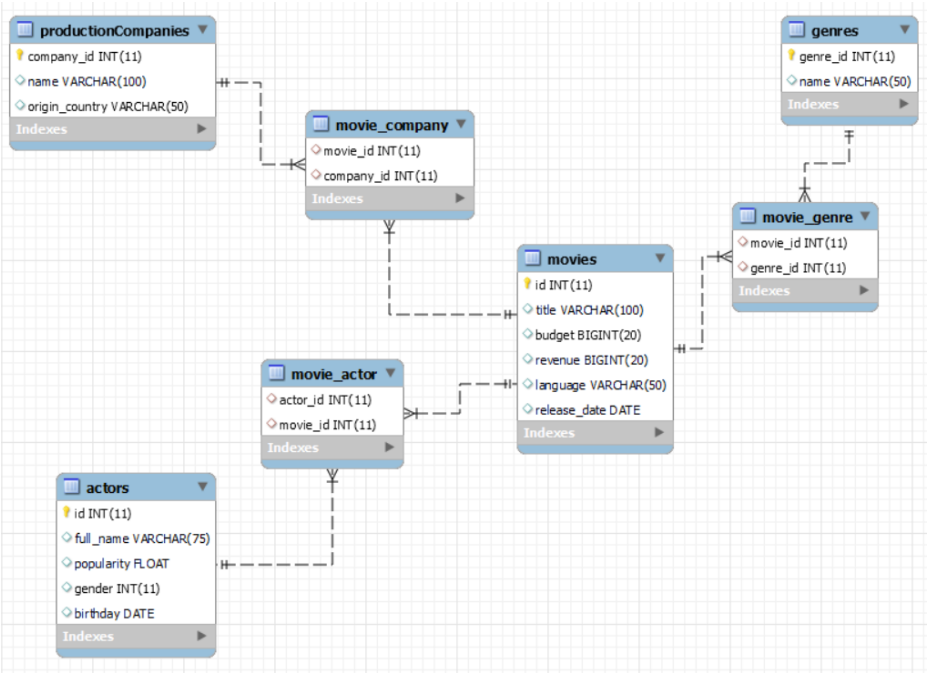
הקבצים בפרויקט הינם :

- /SRC
  - /API-DATA-RETRIEVE.py
  - /CREATE-DB-SCRIPT.py
  - /QUERIES.py
- / DOCUMENTATION
  - USER-MANUAL.pdf
  - SOFTWARE-DOCS.pdf
  - NAMES-AND-IDS.txt
  - MYSQL-USER-AND-PASSWORD.txt
- /queries
  - 1.sql
  - 2.sql
  - 3.sql
  - 4.sql
  - 5.sql
  - 6.sql
  - 7.sql

**הערה:** כדי שהסקריפט של QUERIES.py ירוץ, אין לשנות את היררכית הקבצים.

# מבנה בסיס הנתונים

## סכמה




**Relations:**

- **Actors**(id, full_name, popularity, gender, birthday)
- **Movies**(id, title, budget, revenue, language, release_date)
- **productionCompanies**(company_id, name, origin_country)
- **genre**(genre_id, name)
- **movie_actor**(actor_id, movie_id)
- **movie_company**(movie_id, company_id)
- **movie_genre**(movie_id, genre_id)

## בניית בסיס הנתונים

רצינו לאפשר לחברות השמה לבצע התאמה בין שחקנים לבין חברות הפקה. עבור חברות הפקה בחרנו פרמטרים לפיהם יוכלו לסנן שחקנים עבור תפקידים מסוימים, למשל: מין, גיל, רמת פופולאריות. מצד שני, חברת ההשמה תרצה לתת לשחקנים הזדמנות גבוהה להתקבל לתפקיד. לשם כך, לחברת ההפקה דרוש מאגר חברות הפקה ושחקנים. שני אלו מחוברים על ידי מאגר מידע של סרטים, בעזרתו נוכל לדעת על עבודות קודמות של שחקנים, ולמדוד את ההצלחה שלהם. לאחר שהגדרנו את המידע הדרוש לנו, הגדרנו טבלת ביניים המקשרת בין שחקנים לסרטים וטבלה נוספת המקשרת בין חברות הפקה לסרטים. טבלאות אילו "טריוויאליות" מאחר ומדובר ביחס many-to-many.

כאשר הכנו את ה-DB המטרה העיקרית שלנו הייתה לפרק את המידע כך שנימנע מיתירות של מידע ועדכון של מידע יתבצע במהירות, ולכן כל הטבלאות שלנו מנורמלות.

השאילתות שלנו מתמקדות בקשר בין שחקנים ובין הסרטים שהופיעו בהם, ובין חברות הפקה והסרטים אשר לקחו בהן חלק, ובשל כך זה היה הגיוני ליצור Relation בשם movies אשר ישמש כטבלה מקשרת בין ה-Relations האחרים

(יחד עם טבלאות נוספות). כמו כן, יצרנו את הטבלאות של productionCompanies ו-actors אשר מאגדות מידע חד-חד ערכי לגבי שחקנים וחברות הפקה בהתאמה. טבלה נוספת שיצרנו הינה genres אשר מכילה מזהה ייחודי לז׳אנר ואת השם שלו.

נשקלה האפשרות להכין טבלה אשר מקשרת בין שחקנים לבין חברות הפקה אשר השתתפו בהם, אך התועלת לא הייתה כדאית לעומת הבזבוז בזיכרון – שחקנים וחברות הפקה נמדדים על פי פרמטרים שונים הנוגעים לסרטים שלקחו בהם חלק, ובפרט שעניין זה בא לידי ביטוי בשאילתות שהגדרנו. טבלה מקשרת שכזאת הייתה הופכת להיות הטבלה הגדולה ביותר ב-DB שלנו כאשר טבלת actors ו-productionCompanies הן הגדולות ביותר. מה גם, ואת הקשר הנ״ל אנחנו מקבלים דרך טבלת movies.

כאמור, כל הטבלאות הינן מנורמלות ולכן נעשה שימוש בטבלאות מקשרות בין ה-entities השונים- relationships between entities. כפי שנלמד בכיתה, כאשר קיים יחס של Many-to-many, הטבלאות המקשרות מכילות את המפתחות של ה-entities שהן מקשרות. בין ה-entities השונים קיים רק יחס של many-to-many (לדוגמה, שחקן אחד יכול להופיע בכמה סרטים ובסרט יכולים לשחק מספר שחקנים) וכן קיימות טבלאות מקשרות בין ה-entities השונים באופן שהוזכר.

## <u>שיפור הביצועים</u>

### בניית DB

בניית הטבלאות התבצעה באופן שבו יובטח לנו DB Optimization, כפי שנלמד בכיתה – נעשה וידוא כי כל עמודה מחזיקה data מהסוג הנכון, ונעשה שימוש בטבלאות מקשרות עם מעט עמודות בין ה-entities המרכזיים ב-DB כך שכל שינוי שיתבצע עבור entity – יגרור שינויים קלים יותר ב-DB.

### אינדקסים

האופטימיזציות שנעשו בפרויקט זה באו לידי ביטוי ב-indices ששמנו בטבלאות השונות. בחירת ה-indices התבססה על ההבנה של כיצד השאילתות שהוכנו עבור פרויקט זה יוכלו להתבצע בסיבוכיות I/O הטובה, יחד עם הבנה של מה סביר שיהיו הפרמטרים שמשתמשים באפליקציה ירצו לשאול עליהם הכי הרבה. להלן פירוט אודות כל אינדקס:

| טבלה(שדה) | אינדקס | יתרונות |
|---|---|---|
| Actors(full_name) | Full-text index (reversed list) | אינדקס זה משרת אותנו עבור שאילתת full-text בה החיפוש מתבסס על שאילתב מסוג Match()… AGAINST על השמות של השחקנים (עמודה מסוג string) |
| Movies(release_date) | Clustered Btree | עבור שאילתה 2 מתבצע selection לפי טווח תאריכים של הסרטים. כפי שנלמד בהרצאה, שמירת אינדקס כזה מבטיחה שה-data של הסרטים מסודר לפי התאריך שלהם ובכך ה-Cost קטן משמעותית. |
| productionCompanies (origin_country) | clustered hash | רוב חברות ההפקה ב-DB מתבססות ב-US. עבור פרויקט זה הוכנה שאילתה שמתמקדת דווקא בחברות מחוץ למדינה זאת. האינדקס יעזור לו לסנן אחר חברות אלו. |

**הערה:** בפונקציה createIndex בקובץ CREATE-DB-SCRIPT אנו מוסיפים את האינדקסים לטבלה. לא הצלחנו לגרום לכך שהאינדקסים יהיו מסוג Clustered על פני עמודות שאינן Primary Key. יחד עם זאת, עבור אופטימיזציות התייחסנו לאינדקסים כאילו היו כך.

## שאילתות

עבור השאילתות עקבנו אחר ההנחיות בתרגול 11. ווידאנו כי הפרמטרים השונים שהיו יכולים לגרור cost גבוה ביותר בשאילתה לא יתקיימו. לדוגמה, וידאנו כי אנחנו מנפים ערכים שלא יהיה בהם צורך בעקבות פעולת join שעומדת להתבצע.

**שאילתה 1 – Full-Text**

**פירוט:** חברות השמה לפעמים ממליצות לשחקנים לשנות את שמם בשביל נוכחות מדיה יותר מרשימה. כששחקן מציע שם, החברה תרצה לדעת כמה הוא כבר נמצא בשימוש בתעשייה. לכן, נוכל להנפיק לו דו"ח עם כל השחקנים ששמם מכיל את שמו, או חלק ממנו. למשל, שחקן שרוצה להחליף את שמו כך שיתחיל ב-"Zac" יוכל להריץ את השאילתה הבאה ולקבל רשימה של שחקנים ששמם הינו או מכיל את השם שהוא שוקל, כאשר הרשימה ממוינת לפי הפופולאריות של השחקנים.

```
SELECT actors.full_name, actors.popularity
FROM DbMysql28.actors
where match (full_name) against ('Zac*' IN BOOLEAN MODE)
ORDER BY popularity DESC
```

**אופטימיזציה:** בכדי לאפשר חיפוש full text יש ליצור reverse list index, ולכן יצרנו אינדקס על העמודה של שמות השחקנים.

**שאילתה 2**

**פירוט:** סרט נחשב "מכובד" אם לפחות 3 חברות הפקה השקיעו בו. רמת הניסיון של שחקן נמדדת בכמות הסרטים ה"מכובדים" שהופיע בהם. חברת הפקה מעוניינת ללהק שחקנים מבין השחקנים המנוסים ביותר ב-5 שנים האחרונות (2016-2021). החברה תרצה דו"ח עם שמות השחקנים ומספר הסרטים שהשתתפו בהם, ממויינים לפי רמת הניסיון של השחקנים בסדר יורד.

**שאילתת SQL:**

```
SELECT actors.full_name, Count(movie_filter.id) AS counter
FROM  (SELECT movies.id as id
        FROM movies, movie_company
        WHERE movies.id = movie_company.movie_id
        AND (movies.release_date >= 20160101 and movies.release_date <= 20211227)
        GROUP BY movies.id
        HAVING Count(DISTINCT movie_company.company_id) > 3) AS movie_filter, actors,
        movie_actor
WHERE movie_filter.id = movie_actor.movie_id AND movie_actor.actor_id = actors.id
GROUP BY actors.id
ORDER BY counter DESC;
```

**אופטימיזציה:** עבור שאילתה זו נעשה שימוש ב-sub-query בה לסנן עבור סרטים ב-5 שנים האחרונות. בשאילתה הפנימית אנחנו צריכים רק את ה-id של הסרטים, ולכן אנחנו נבצע projection עליו בלבד, כדי לא להחזיר מידע מיותר. בנוסף, בכדי לתמול במציאת טווח של תאריכי יציאה של סרטים, הוכן אינדקס מסוג clustered Btree לפי העמודות של release_date בטבלת הסרטים. אופטימיזציה זו מאפשרת לשאילתת select הזאת להתבצע ב-cost הכי טוב שאפשר.

**שאילתה 3**

**פירוט:** בכדי ללהק לתפקידים ביעילות, חברות הפקה צריכות מידע על שחקנים, לפי פרמטרים שונים. למשל, בכדי ללהק לתפקיד של אישה צרפתיה בסרט קומי, חברת הפקה תרצה רשימה של נשים בטווח הגילאים 40-50 ששחקו בקומדיה צרפתית. נחזיר את הרשימה של השחקניות ואת הפופולאריות שלהן, כאשר השורות ממוינות לפי הפופולאריות של השחקניות.

```
SELECT actors.full_name, actors.popularity
FROM actors
WHERE EXISTS (
        SELECT actors.id
        FROM movies, movie_actor, movie_genre, genres
        WHERE movies.id = movie_actor.movie_id AND
        actors.id = movie_actor.actor_id AND
        movies.id = movie_genre.movie_id AND
        movie_genre.genre_id = genres.genre_id AND
        genres.name = 'comedy' AND movies.language = 'fr'AND
        actors.gender  = 1 AND
        actors.birthday > 19710101 AND actors.birthday < 19810101)
ORDER BY actors.popularity
```

**אופטימיזציה:** בשאילתה הפנימית אנחנו צריכים רק את ה-id של השחקנים, ולכן אנחנו נבצע projection עליו בלבד, כדי לא להחזיר מידע מיותר.

**שאילתה 4**

**פירוט:** עבור סרט מדע בדיוני מבטיח חברת הפקה רוצה הזדמנות ללהק שחקנית פחות מוכרת. לשחקנית צריכה להיות רמת פופולאריות של 0.7 ומטה, ושלא השתתפה ביותר מ-2 סרטים בשנה האחרונה. עבור כל שחקנית, החזר את השם של השחקנית והסרט האחרון שהשתתפה בו, ממוין לפי התאריך בסדר יורד.

```
SELECT actors.full_name, Max(movies.release_date) AS latest_date
FROM movies, actors, movie_actor
WHERE movies.id = movie_actor.movie_id AND
        actors.id = movie_actor.actor_id AND
        actors.popularity < 0.7 AND
        actors.gender = 1
GROUP BY actors.id
HAVING Count(DISTINCT movies.id) <= 2
ORDER BY latest_date DESC
```

**שאילתה 5**

**פירוט:** חברת הפקה נחשבת מוצלחת אם הצליחה להוציא לפחות 4 סרטים עם רווח גדול מ-$10,000,000. לאחרונה הרבה שחקנים אמריקאיים מעוניינים לפרוץ לתודעה של מדינות אחרות ולעבוד רק עבור חברות הפקה מוצלחות שאינן מבוססות ב-US. נרצה להנפיק דו"ח עם שמות של חברות הפקה מצליחות מחוץ ל-US, יחד עם הסרט הרווחי ביותר שהוציאו. הדו"ח יהיה ממויין לפי הרווחים של הסרטים הכי מצליחים בסדר יורד.

```
SELECT productionCompanies.name, movies.title, non_us_based.earnings
FROM(
        SELECT productionCompanies.company_id, Max(movies.revenue - movies.budget) as
        earnings
        FROM productionCompanies, movies, movie_company
        WHERE movies.id = movie_company.movie_id
                and movie_company.company_id = productionCompanies.company_id
                and productionCompanies.origin_country <> "US"
                and movies.revenue - movies.budget > 10000000
        GROUP BY productionCompanies.company_id
        HAVING count(DISTINCT movies.id) > 3 ) as non_us_based, movies, movie_company,
        productionCompanies
WHERE productionCompanies.company_id = non_us_based.company_id and
        movies.id = movie_company.movie_id and
        movie_company.company_id = productionCompanies.company_id and
        non_us_based.earnings = (movies.revenue - movies.budget)
ORDER BY earnings DESC
```

**אופטימיזציה:** שיפור ראשון היה לבצע selection ו-projection מוקדם ככל האפשר. בשאילתה הפנימית, סיננו לפי התקציבים, בכדי לעבוד בשאילתה החיצונית עם מידע קטן יותר. באופן דומה, סיננו את העמודות. השיפור השני היה הוספת index מסוג hash clustered, בכדי שנוכל לענות ביעילות על השאלה אם מדינה היא לא ב-"US".

**שאילתה 6**

**פירוט:** על מנת להתאים שחקנים שרוצים לעבוד בז׳אנר מסוים, חברת ההשמה מנפיקה דו״ח המדרג לכל ז׳אנר את 3 חברות ההפקה הטובות ביותר , כאשר הדירוג של חברות ההפקה נקבע לפי מספר השחקנים שהעסיקה עבור אותו ז׳אנר. הדו״ח יחזיר לכל ז׳אנר את 3 חברות ההפקה הטובות ביותר, מספר השחקנים שהחברות העסיקו בז׳אנר זה ואת הדירוג שלהן עבור הז׳אנר. משום שיש מספר סופי של ז׳אנרים ישנה רק חשיבות לסדר של הדירוג של כל חברה בכל ז׳אנר.

```
SELECT*
FROM(
        SELECT  genres.name as genre, productionCompanies.name, COUNT(DISTINCT actors.id)
        AS employees,
        RANK() OVER (PARTITION BY genres.name ORDER BY count(DISTINCT actors.id)
        DESC) AS ranking
        FROM productionCompanies, movie_company, movies, movie_actor, actors, genres,
        movie_genre
        WHERE productionCompanies.company_id = movie_company.movie_id
                AND movie_company.movie_id = movies.id
                AND movies.id = movie_actor.movie_id
                AND movie_actor.actor_id = actors.id
                AND movies.id = movie_genre.movie_id
        AND movie_genre.genre_id = genres.genre_id
        Group By productionCompanies.company_id, genres.genre_id) as sub_query
WHERE sub_query.ranking <=3
```

**שאילתה 7**

**פירוט:** שחקנים שפונים לחברת הפקה יודעים שכדאי להם להיות מועסקים על ידי חברות שיש להן הרבה מאוד ״קשרים״. כאשר שתי חברות הפקה עבדו במשותף על סרט ניתן להגיד כי החברות יצרו קשר. חברת ההשמה רוצה לחלץ דו״ח שמדרג את החברות לפי מספר הקשרים שהן יצרו. הדו״ח יהיה ממוין לפי מספר הקשרים של כל חברה בסדר יורד.

```
SELECT p1.name, COUNT(p2.company_id) as connections
FROM productionCompanies as p1, productionCompanies as p2,
        movie_company as mp1, movie_company as mp2
WHERE mp1.company_id = p1.company_id AND
        mp2.company_id = p2.company_id AND
        p1.company_id != p2.company_id AND
        mp1.movie_id = mp2.movie_id
GROUP BY p1.company_id
ORDER BY Count(p2.company_id) DESC
```

# מבנה הקוד

## 1. CREATE-DB-SCRIPT

זוהי תוכנית PYTHON היוצרת את בסיס הנתונים. ראשית, היא יוצרת את הטבלאות, ולאחר מכן מוסיפה את האינדקסים.

**תיאור הפונקציות**:

```
def main()
```

הפונקציה מריצה את התוכנית.

```
def init_connection():
# This function is connecting to our server and returns the cursor
```

הפונקציה יוצרת קשר עם בסיס הנתונים שלנו ומחזירה את ה-cnx וה-cursor שמאפשרים לנו לעבוד עם בסיס הנתונים.

```
def createDB(cnx, cursor):
# This function creates our database
```

הפונקציה יוצרת את הטבלאות של בסיס הנתונים, כולל הגדרת מפתחות ומפתחות זרים.

```
def createIndexes(cursor):
# This function adds the indexes to the table.
```

הפונקציה מוסיפה את האינדקסים הרלוונטיים לטבלאות.

```
def deleteDB(cursor):
# This function deletes all tables from our database.
```

פונקציה המוחקת את כל הטבלאות מבסיס הנתונים.

```
def clearDB(cursor):
# This function clears all rows from our database.
```

פונקציה המסירה את כל השורות בכל הטבלאות בבסיס הנתונים.

## 2. API-DATA-RETRIEVE

```
def main()
```

הפונקציה מריצה את התוכנית.

```
def init_connection():
# This function is connecting to our server and returns the cursor
```

הפונקציה יוצרת קשר עם בסיס הנתונים שלנו ומחזירה את ה-cnx וה-cursor שמאפשרים לנו לעבוד עם בסיס הנתונים.

```
def getMoviesIDs():
# This function gets all the movies IDs from the API and
# returns it as a set.
```

הפונקציה מבצעת קריאות ל-API ומקבלת רשימה של סרטים, שומרת את ה-ID של כל סרט ומחזירה סט המכיל את כל ה-IDS של הסרטים.

*הערת צד: לשם קריאות הגבלנו את כמות הסרטים, שכן הם משליכים באופן ישיר על מספר השחקנים במאגר הנתונים שחצה את 50,000. ניתן לשנות זאת על ידי עדכון משתנה limit בתחילת התוכנית ל-1000, למשל.

```
def insertMovies(cnx, cursor, movies_ids):
```

```
# This function is getting cnx, cursor, movies_ids
# and inserts movies to movies table in our database.
# In addition, it returns two dictonaries and a set:
# D1. movie_company: key - movie_id, value - a set of its production
companies
# D2. movie_genre: key - movie_id, value - a set of its genres
# S1. companies_ids
```

הפונקציה מקבלת סט המכיל את ה-Ids לכל הסרטים במאגר. בעזרת קריאה ל-API, היא מקבלת מידע על כל סרט, אותו אנחנו נשמור ב-DB. בנוסף, אנחנו מייצרים מאותו המידע מילונים המקשרים בין סרט לחברות ההפקה שלו ולז'אנרים שלו וסט המכיל את ה-IDs של כל חברות ההפקה הקיימות במאגר.

```
def insertCompanies(cnx, cursor, companies_ids):
# This function is getting cnx, cursor, companies_ids
# and inserts companies to productionCompanies table in our database.
```

הפונקציה מקבל את ה-Ids של כל חברות ההפקה ומכניסה אותן ל-DB.

```
def insertMovieCompany(cnx, cursor, movie_company):
# This function is getting cnx, cursor, a dictonary movie_company
# where key - movie_id, value - a set of its production companies
# and inserts movie_company relations into movie_company table.
```

הפונקציה מקבלת מילון המקשר בין סרט לחברות ההפקה שלו ומכניסה את הקשרים הללו לטבלה המקשרת ב-DB בשם movie_company.

```
def insertGenres(cnx, cursor):
# This function is getting cnx and cursor
# and inserts genres to genres table in our database.
```

הפונקציה מקבלת מה-API רשימה של ז'אנרים ומוסיפה אותם ל-DB.

```
def insertMovieGenre(cnx, cursor, movie_genre):
# This function is getting cnx, cursor, a dictonary movie_genre
# where key - movie_id, value - a set of its genres
# and inserts movie_genre relations into movie_genre table.
```

הפונקציה מקבלת מילון המקשר בין סרט לז'אנרים שלו ומכניסה את הקשרים הללו לטבלה המקשרת ב-DB בשם movie_genre.

```
def getActors(movies_ids):
# This function is getting movies_ids and returns a dictonary
# and a set:
# D1. movie_actors: key - movie_id, value - a set of its personnel
# S1. actors_ids
```

הפונקציה מקבלת סט המכיל את ה-Ids של כל הסרטים במאגר. לכל סרט מבצעת קריאה ל-API ומקבלת מידע לגביו, אותו שומרת ל-DB. בנוסף, בעזרת מידע זה מחזירה סט המכיל את ה-Ids של כל השחקנים ומילון המקשר בין סרט לכל השחקנים שהשתתפו בו.

```
def insertActors(cnx, cursor, actors_ids):
# This function is getting cnx, cursor, people_ids
# and inserts people to people table in our database.
```

הפונקציה מקבל את ה-Ids של כל השחקנים ומכניסה אותן ל-DB.

```
def insertMovieActors(cnx, cursor, movie_actors):
# This function is getting cnx, cursor, a dictonary movie_actors
# where key - movie_id, value - a set of its people
# and inserts movie_people relations into movie_people table.
```

הפונקציה מקבלת מילון המקשר בין סרט לשחקנים שלו ומכניסה את הקשרים הללו לטבלה המקשרת ב-DB בשם movie_actor.

```
def insertData(cnx, cursor):
# This function inserts our information into the DB
```

זו הפונקציה הראשית שנקראת מה-main ומנהלת את הכנסת כל המידע ל-DB.

```
def print_progress(current, total):
```

פונקציית עזר זו מדפיסה את "קצב ההורדה". כלומר, כמה פניות ביצענו ל-API מסך הפניות שאנחנו צריכים לעשות בסך הכול.

## 3. QUERIES

```
1. def main():
2. # This function asks for input of the query number and prints its
   result
```

פונקציה זו מקבלת קלט בין 1 ל-7 ומחזירה את הפלט לפי השאילתה הנבחרת.

```
def init_connection():
# This function is connecting to our server and returns the cursor
```

באופן דומה לקודם.

```
def getQuery(queryNumber):
```

הפונקציה הזו מקבלת את מספר השאילתה וקוראת את הפקודה עליה לבצע מקובץ הSQL המתאים.

לכל שאילתה, מוגדרת פונקציה מתאימה אשר מחזירה את המידע המתאים מהצורה הנ"ל. למשל עבור השאילתה הראשונה הפונקציה הבאה תחזיר את המידע המבוקש:

```
def query1(cursor):
```

# The movie Database API

לתרגיל זה השתמשנו ב-The movie Database API בכתובת:

API Overview — The Movie Database (TMDB) (themoviedb.org)

.

**אופן השימוש:**

תחילה, בעזרת מעבר על הסרטים המדורגים ביותר (בלולאה על כל הדפים) באופן הבא:

```
link = "https://api.themoviedb.org/3/movie/top_rated?api_key=" + \
     api_key + "&language=en-US&page={}"
```

קיבלנו את ה-Ids לכל הסרטים במאגר.

בעזרת ה-Ids הללו, קיבלנו מה-API מידע לגבי כל סרט:

```
link = "https://api.themoviedb.org/3/movie/{}?api_key=" +\
     api_key + "&language=en-US"
```

מהשלב הקודם, קיבלנו את ה-Ids של כל החברות לכל סרט, ובעזרתם שלפנו את המידע הרצוי לגבי כל חברת הפקה:

```
link = "https://api.themoviedb.org/3/company/{}?api_key=" + api_key
```

בעזרת השאילתה הבאה קיבלנו רשימה של כל הז׳אנרים ב-API:

```
link = "https://api.themoviedb.org/3/genre/movie/list?api_key=" + \
    api_key + "&language=en-US"
```

נעזרנו פעם נוספת ב-Ids של הסרטים כדי לקבל מה-API את ה-credits של כל סרט, וכך השגנו את ה-Ids של כל השחקנים והקשר שלהם לכל סרט:

```
link = "https://api.themoviedb.org/3/movie/{}/credits?api_key=" + \
    api_key + "&language=en-US"
```

לבסוף, קיבלנו את המידע על השחקנים בעזרת ה-ID שלהם על ידי:

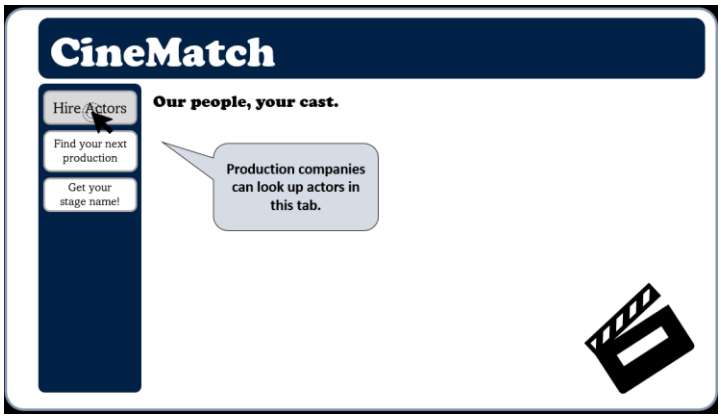
```
link = "https://api.themoviedb.org/3/person/{}?api_key=" + \
     api_key + "&language=en-US"
```

את המידע הנ״ל ארגנו והכנסנו לטבלאות של ה-DB שלנו.

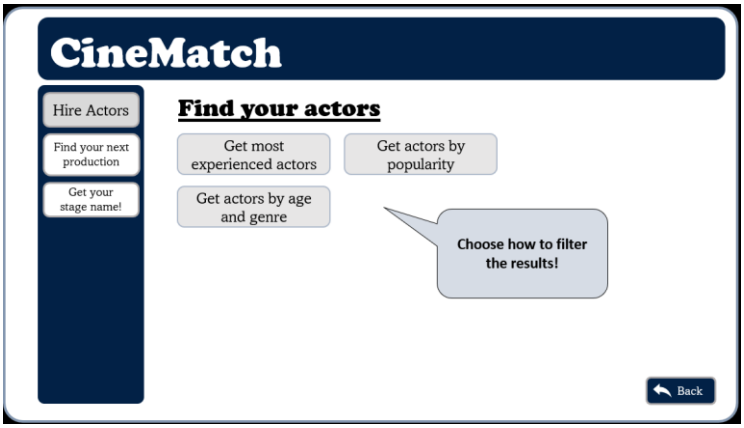
# General Flow

האפליקציה תתחיל בדף הבית :

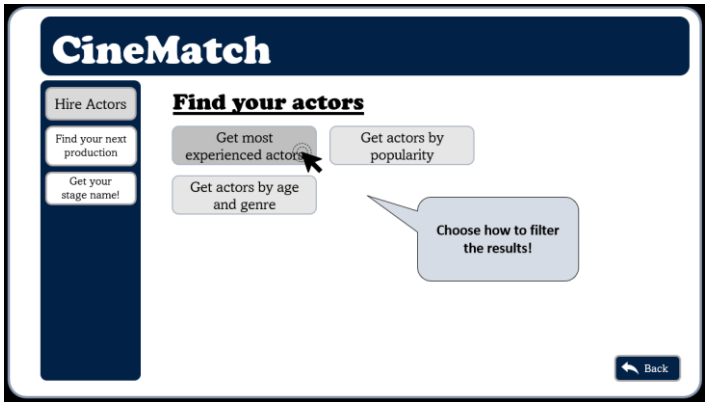

חברות הפקה המחפשות שחקנים יוכלו בלחיצה לחפש שחקנים לפי פרמטרים מועדפים :



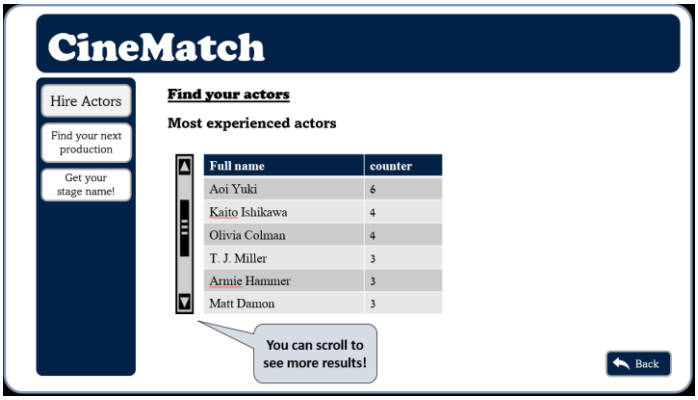

כל כפתור בתפריט שנפתח מייצג שאילתה שהגדרנו :



למשל, על ידי לחיצה על הכפתור הבא:




נעבור לדו"ח הנוכחי, שמתאים לשאילתה 2:




"מאחורי הקלעים" – בלחיצה על הכפתור, נריץ את הסקריפט QUERIES.py נגדיר שאנו מעוניינים בשאילתה מספר 2. את התוצאות המתקבלות נכניס לטבלה.