

## Raport

Celem projektu było porównanie czasu przykładowych zapytań do bazy relacyjnej i obiektowej, oraz sprawdzenie ich skalowalności względem liczby rekordów.

Do testów użyliśmy bazy danych z filmami w liczbie 27 278, oraz ocenami w liczbie 20 000 263.

Dane pobraliśmy ze strony <https://grouplens.org/datasets/movielens/> (MovieLens 20m Dataset).

Aby testować skalowalność przeprowadzaliśmy zapytania dla 5, 10, 15, 20 milionów ocen.

### MySQL

W testach bazy relacyjnej łączyliśmy się z MySQL poprzez SQLAlchemy. Zapytania formułowane były poprzez SQL. Model bazy został skonstruowany w następujący sposób:

movieId	title	genres
30894	White Noise (2005)	Drama Horror Mystery Sci-Fi Thriller

Przykładowy rekord z tabeli „movies”

W pierwszej tabeli („movies”) rekord posiada 3 kolumny:

- movieId – klucz główny tabeli, identyfikator filmu
- title – tytuł filmu
- genres – gatunki filmu

movieId	rating
30894	3.0

Przykładowy rekord z tabeli „ratings”

W drugiej tabeli („ratings”) rekord posiada 2 kolumny:

- movieId – klucz obcy odpowiadający filmowi z pierwszej tabeli
- rating – ocena filmu

Tabele zostały połączone relacją jeden do wielu.

Import danych do bazy relacyjnej wyglądał następująco:

```
-----CREATE DATABASE filmy_oceny
-----USE filmy_oceny
-----CREATE TABLE movies (movieId INT NOT NULL,
                             title VARCHAR(255) NOT NULL,
                             genres VARCHAR(255) NOT NULL,
                             PRIMARY KEY (movieId));
-----CREATE TABLE ratings (movieId INT NOT NULL,
                              rating VARCHAR(255) NOT NULL,
                              FOREIGN KEY (movieId) REFERENCES movies(movieId));
-----SET NAMES 'utf8';
-----SET CHARACTER SET utf8;

-----LOAD DATA LOCAL INFILE '/home/piotr/Desktop/BDII/projekt/movies.csv'
      INTO TABLE movies
      FIELDS TERMINATED BY ','
      ENCLOSED BY '"'
      LINES TERMINATED BY '\r\n'
      IGNORE 1 ROWS;

-----LOAD DATA LOCAL INFILE '/home/piotr/Desktop/BDII/projekt/5msql.csv'
      INTO TABLE ratings
      FIELDS TERMINATED BY ','
      LINES TERMINATED BY '\n'
      IGNORE 1 ROWS;
```

## ZODB

W testach bazy obiektowej łączyliśmy się z ZODB poprzez ZEO. Rekordy w bazie danych przechowywaliśmy w B-drzewie jako obiekty klasy Movie.

```
class Movie(persistent.Persistent):
    def __init__(self, title, genres):
        self.title = title
        self.genres = genres.split("|")
        self.ratings = []

    def set_title(self, title):
        self.title = title

    def set_genres(self, genres):
        self.genres = genres

    def set_ratings(self, ratings):
        self.ratings = ratings

    def addRating(self, r):
        self.ratings.append(r)
```

*Klasa Movie*

Oceny filmów przechowywane są w tym samym obiekcie w polu ratings (lista).

# TESTY

Dla każdej z baz przeprowadziliśmy po 3 testy z pomiarami czasu:

- Test nr 1 – zapytanie o najlepszy film z gatunku Thriller posiadający co najmniej 20 ocen
- Test nr 2 – 250000 zapytań o losowy film
- Test nr 3 – dopisanie 10000 rekordów do bazy danych.

Funkcje testujące znajdują się w załączniku pod nazwami:

- Baza relacyjna (mysql.py)
  - Test nr 1 – bestThriller()

```
def bestThriller():
    result = session.execute('SELECT title, AVG(ratings.rating) as average, COUNT(ratings.rating) as numb \
                              FROM (SELECT * FROM movies where genres like "%Thriller%") as tharray \
                              JOIN ratings on ratings.movieId = tharray.movieId \
                              GROUP BY title HAVING numb > 20 ORDER BY average DESC LIMIT| 1;').fetchall()
    print("Best -- " + str(result[0][0]) + " " + str(result[0][1]))
```

- Test nr 2 – random1Mmovies(m, movieList = [])

```
def random1Mmovies(m, movieList = []):
    if(m==1):
        idList = movieList
        #1
        for i in range (0,250000):
            randomMovie = random.choice(idList)
            resultsql2 = session.execute('SELECT title from movies \
                                         WHERE movieId = :randomMovie;', \
                                         {'randomMovie':randomMovie}).fetchone()
            print(str(resultsql2[0]))
```

- Powyższa funkcja wykonuje 3 różne zapytania w zależności od parametru „m”. W teście użyliśmy najbardziej optymalnego zapytania(m = 1) polegającego na założeniu, że serwer posiada listę id filmów i na jej podstawie odpytuje bazę danych

- Test nr 3 – saveRandomData()

```
def saveRandomData():
    startIndex=150000
    for i in range(startIndex, 160000):
        sqlAdd = session.execute('INSERT INTO movies(movieId,title,genres) \
                                  VALUES(:movieId,:title,:genres);', \
                                  {'movieId':i, \
                                  'title':"Random Title " +str(i), \
                                  'genres':"Thriller|Action|Comedy|Horror"})
        for j in range(0,3):
            for k in range(1,6):
                sqlAddRating = session.execute('INSERT INTO ratings(movieId,rating) \
                                                VALUES(:movieId,:rating);', \
                                                {'movieId':i,'rating':k})
    session.commit()
    sqltitle = session.execute('SELECT title FROM movies \
                               WHERE movieId = :id;', \
                               {'id':i}).fetchone()
    print(str(sqltitle[0]))
```

- Baza obiektowa (testyobektowe.py)

- Test nr 1 – bestThriller(tree) wraz z funkcją pomocniczą ave(rList)

```
def ave(rList):
    if(len(rList) == 0):
        return 0
    s=0.0
    for i in rList:
        s+=float(i)
    return s/len(rList)

def bestThriller(tree):
    bestScore = 0.0
    bestT = ""
    for i in tree.keys():
        if("Thriller" in tree[i].genres and len(tree[i].ratings) > 20):
            tmp = ave(tree[i].ratings)
            if(tmp > bestScore):
                bestScore = tmp
                bestT = tree[i].title
    print("Best -- " + bestT + " " + str(bestScore))
```

- Test nr 2 – randomMovies(tree)

```
def randomMovies(tree):
    for i in range(0,250000):
        randomMovie = random.choice(tree.keys())
        print(str(tree[randomMovie].title))
```

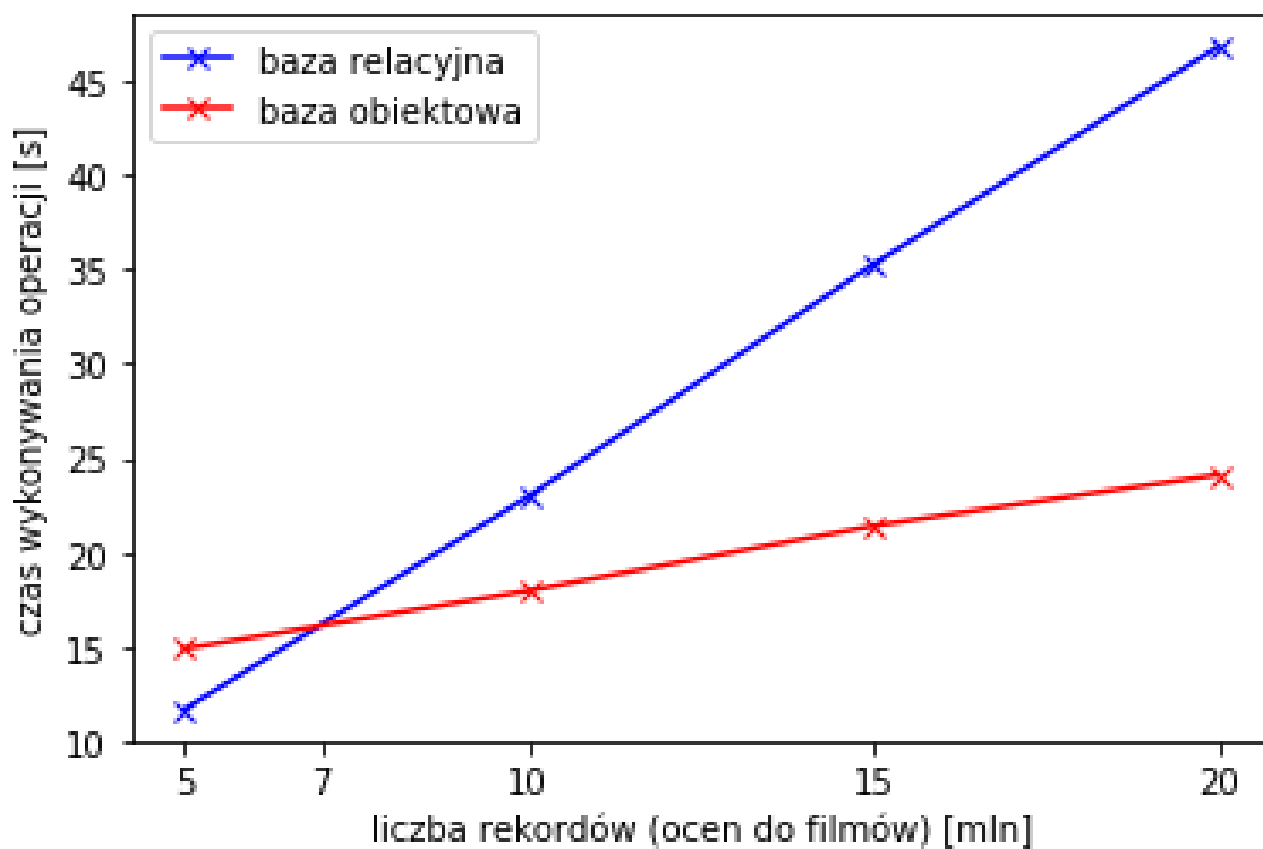
- Test nr 3 – saveRandomData(tree)

```
def saveRandomData(tree):
    startIndex=150000
    for i in range(startIndex, 160000):
        movieId = str(i)
        title = "Random Title " + movieId
        genre = "Thriller|Action|Comedy|Horror"
        rating = [1,2,3,4,5,1,2,3,4,5,1,2,3,4,5,1,2,3,4,5]
        tree[movieId] = Movie(title, genre)
        tree[movieId].addRating(rating)
        print(str(tree[movieId].title))
    transaction.commit()
```

## WYNIKI

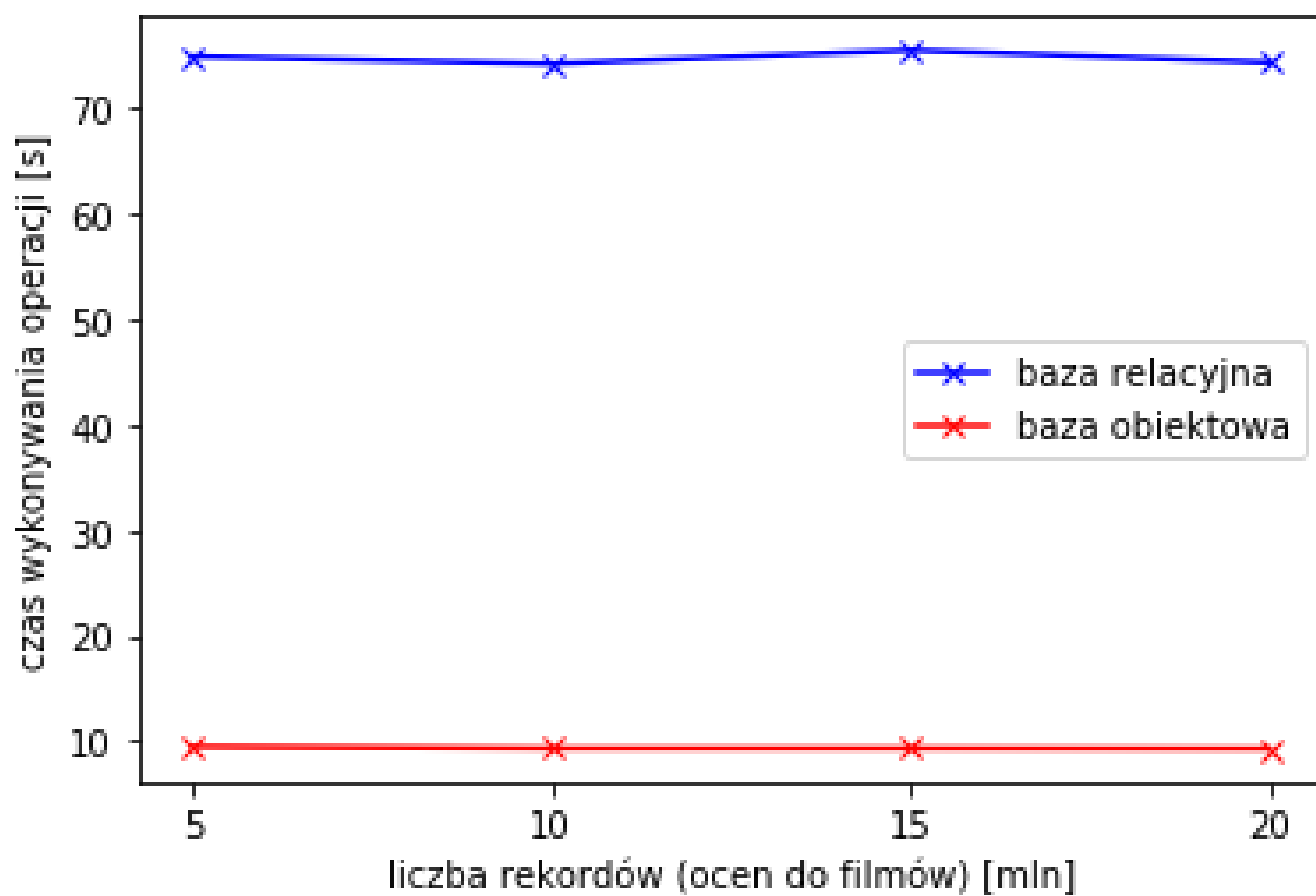
Test nr 1

	Baza relacyjna	Baza obiektowa
Ilość ocen	Czas wykonania [s]	
5 mln	11.7366361618042	14.94456171989441
10 mln	23.00187373161316	18.013745307922363
15 mln	35.28084588050842	21.451774835586548
20 mln	46.77485370635986	24.14399027824402



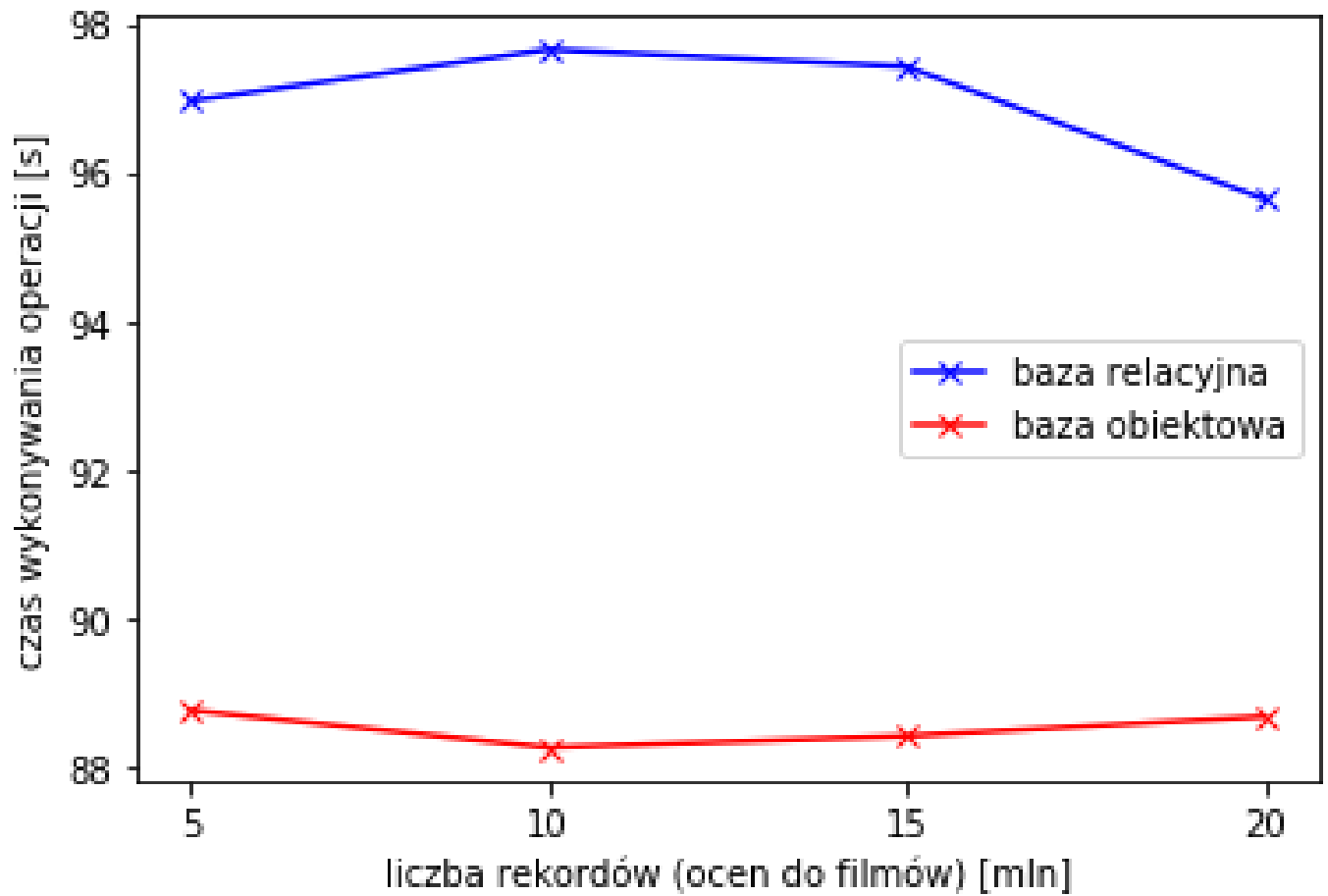
Test nr 2

	Baza relacyjna	Baza obiektowa
Ilość ocen	Czas wykonania [s]	
5 mln	74.95877528190613	9.511908292770386
10 mln	74.25568914413452	9.352771282196045
15 mln	75.54806280136108	9.350659608840942
20 mln	74.40225386619568	9.325289011001587



Test nr 3

	Baza relacyjna	Baza obiektowa
Ilość ocen	Czas wykonania [s]	
5 mln	96.9652726650238	88.76216793060303
10 mln	97.65681028366089	88.26180005073547
15 mln	97.42846298217773	88.41565036773682
20 mln	95.64041709899902	88.67719268798828



## WNIOSKI

W naszych testach obiektowa baza danych wypadła lepiej względem wydajności. W pierwszym teście możemy zauważyć, że obiektowa baza danych skaluje się lepiej niż baza relacyjna i już dla około 7 milionów ocen była bardziej wydajna. Spowodowane jest to korzystniejszym modelem danych polegającym na przechowywaniu listy ocen w tym samym obiekcie co pozostałe dane filmu. W pozostałych testach nie ma zmian w wydajności względem liczby ocen. Zapytanie o losowy film jest znacznie bardziej wydajne w obiektowej bazie danych. Średni czas zapytania o 1 film wyniósł 0.000299s w MySQL i 0.000038s w ZODB. W przybliżeniu jest to około 8 razy szybciej. W trzecim teście polegającym na zapisie filmów wraz z jego ocenami obie bazy osiągnęły podobne wyniki. Średni czas zapisu jednego filmu z ocenami wyniósł 0.009692 w MySQL i 0.008852s w ZODB.

## DODATKOWE TESTY

Zrobiliśmy również dodatkowe testy na 20mln bazach. Sprawdziliśmy szybkość wyszukiwania najlepszego Thrillera w obiektowej bazie danych w zmienionym modelu używając dodatkowej struktury B-drzewa:

```
root.thriller = BTrees.OOBTree.BTree()

for i in root.movies.keys():
    if("Thriller" in root.movies[i].genres and len(root.movies[i].ratings) > 20):
        root.thriller[i] = Movie(root.movies[i].title,"1|2")
        root.thriller[i].set_genres(root.movies[i].genres)
        for r in root.movies[i].ratings:
            root.thriller[i].addRating(r)

for i in root.thriller.keys():
    print(root.thriller[i].title)
```

Drzewo to zawierało wszystkie Thrillery z co najmniej dwudziestoma ocenami. Czas wyszukania najlepszego Thrillera wyniósł wtedy 5.025348663330078s. Zapytanie z oryginalnego modelu wyniosło dla porównania 24.14399027824402s.

Sprawdziliśmy również dla drugiego testu wylosowanie 250000 filmów w relacyjnej bazie danych poprzez (bardzo wolne) zapytanie SQL:

```
'SELECT title FROM movies ORDER BY Rand() LIMIT 1;'
```

Przy takim zapytaniu otrzymaliśmy czas równy 2571.9706058502197s (~43 min). Czas wylosowania jednego filmu wyniósł około 0.010287s.

Załączniki:

- plik importu csv do MySQL (importSQL)
- plik importu csv do ZODB (obiektowa.py)
- plik testów do MySQL (mysql.py)
- plik testów do ZODB (testyobietkowe.py)



