Box Office App

Overview

There is one dependent variable (sales) and three independent variables (date, theater, movie) in the problem. For any single API call, we can vary exactly one of the independent variables, and either sum over or fix each of the remaining two. We will first describe the schema used in the database. Then we will document each API call that can be made, along with a discussion of the indices necessary to insure fast lookup for each call. Finally, we will give instructions on how to run the code.

Schema

There are three tables: movies, theaters, sales.

Movies columns:

Id: primary key auto increment
Title: varchar(128) unique
release_date: date not null
end_date: date default null

Explanation: Each movie is uniquely identified by its title. However, it is bad practice (and slows down lookups) to use varchar as a primary key. Therefore, we use an id that is auto incremented as the primary key. There is a unique index on title that enables one to easily retrieve a movies id from its title. Finally, release_date can't be null since every movie has a definite release date, while end_date can be, since we may not know when a movie currently showing in theaters will be taken out.

Theaters columns:

Id: primary key auto increment
Name: varchar(128) unique
City: varchar(128)

Explanation: The explanation given for the “title” column in the “movies” table carries over to the “name” column in the “theaters” table. The city column describes the city in which the theater is located. Since this information may not be available, we have allowed the column to be null.

Sales columns:

Id: primary key auto increment
movie_id: int references movies(id) on delete cascade
theater_id: int references theaters(id) on delete cascade
sale_date: date not null
total_sale: numeric default 0

Explanation:
This table describes the total sales for a given movie, theater and date. The movie is uniquely identified by the movie_id column, which references the id column of the movies table. If a movie is deleted from the movies table, all corresponding entries are automatically deleted from the sales table. The theater is uniquely identified by the theater_id column, which reference the id column of the theaters table. If a theater is deleted from the theaters table, all corresponding entries are automatically deleted from the sales table. The sale_date column identifies the date and can’t be null. Finally, the total_sale column identifies the dollar amount of the total sales made for that particular movie, theater, and date. It defaults to 0.

API Calls and Indices

Url: https://localhost:4324/salesovercategory

Parameters:

Category: The independent variable. This can be either “movie” or “theater”.

st_date: The lower end (inclusive) of the range of dates that will be summed over.

end_date: The upper end (inclusive) of the range of dates that will be summed over.

Name (optional): The name of the variable conjugate to the category. E.g. if category="theater", then name is a movie title, and vice versa. If name is given, then the conjugate variable is fixed, otherwise it is summed over.

Limit: The number of entries to return. They are returned in descending order of total sales.

Note that to enact the query given in the problem description, fix st_date = end_date, set category=theater, do not provide the name parameter, and set limit=1.

Examples:

Url: https://localhost:4324/salesovercategory?st_date=11-28-2024&end_date=11-28-2024&category=theater&name=Wicked&limit=3

Explanation: We have fixed the date and the movie, and have returned the top three grossing theaters in descending order of revenue.

Indices: The index (movie_id, sales_date, total_sales) ensures fast lookup for this query. After the movie_id and sales_date are fixed, the results are sorted by total_sales.

Url: https://localhost:4324/salesovercategory?st_date=11-28-2024&end_date=11-28-2024&category=movie&name=THE%20LOT%20Libert%20Station&limit=3

Explanation: We have fixed the date and the theater, and have returned the top three grossing movies in descending order of revenue.

Indices: The index (theater_id, sales_date, total_sales) ensures fast lookup for this query. After the theater_id and sales_date are fixed, the results are sorted by total_sales.

Url: https://localhost:4324/salesovercategory?st_date=11-28-2024&end_date=11-30-2024&category=theater&name=Wicked&limit=3

Explanation: We have fixed the movie, summed over the range of dates, and have returned the top three grossing theaters in descending order of revenue.

Indices: The unique index (movie_id, theater_id, sales_date) ensures fast lookup for this query. After the movie_id is fixed, for each theater_id, we sum over the given range for the sales_date.

Url: https://localhost:4324/salesovercategory?st_date=11-28-2024&end_date=11-30-2024&category=movie&name=THE%20LOT%20Libert%20Station&limit=3

Explanation: We have fixed the theater, summed over the range of dates, and have returned the top three grossing movies in descending order of revenue.

Indices: The unique index (movie_id, theater_id, sales_date) ensures fast lookup for this query. For each movie_id, we fix the theater_id, and sum over the given range for the sales_date.

Url: https://localhost:4324/salesovercategory?st_date=11-28-2024&end_date=11-28-2024&category=theater&limit=3

Explanation: We have fixed the date, summed over all of the movies, and have returned the top three grossing theaters in descending order of revenue.

Indices: The index (theater_id, sales_date) ensures fast lookup for this query. For each theater_id, we fix the sales_date, and sum over all of the movies. This index is subsumed in the aforementioned (theater_id, sales_date, total_sales) index.

Url: https://localhost:4324/salesovercategory?st_date=11-28-2024&end_date=11-28-2024&category=movie&limit=3

Explanation: We have fixed the date, summed over all of the theaters, and have returned the top three grossing movies in descending order of revenue.

Indices: The index (movie_id, sales_date) ensures fast lookup for this query. For each movie_id, we fix the sales_date, and sum over all of the theaters. This index is subsumed in the aforementioned (movie_id, sales_date, total_sales) index.

Url: https://localhost:4324/salesovercategory?st_date=11-28-2024&end_date=11-30-2024&category=theater&limit=3

Explanation: We have summed over the range of dates, and all of the movies, and returned the top three grossing theaters.

Indices: The index (theater_id, sales_date) ensures fast lookup for this query. For each theater_id, we sum over all of the dates in the date range, summing over all of the movies for each one. This index is subsumed in the aforementioned (theater_id, sales_date, total_sales) index.

Url: https://localhost:4324/salesovercategory?st_date=11-28-2024&end_date=11-30-2024&category=movie&limit=3

Explanation: We have summed over the range of dates, and all of the theaters, and returned the top three grossing movies.

Indices: The index (movie_id, sales_date) ensures fast lookup for this query. For each movie_id, we sum over all of the dates in the date range, summing over all of the theaters for each one. This index is subsumed in the aforementioned (movie_id, sales_date, total_sales) index.

Url: https://localhost:4324/salesovertime

Parameters:

st_date: The sales_date is now the independent variable. st_date is the lower end (inclusive) of the range of dates for which total_sales will be returned.

end_date: The upper end (inclusive) of the range of dates for which total_sales will be returned.

Category (optional): The variable(s) to be fixed. If category="theater" ("movie"), then the theater(movie) variable will be fixed, and the movie(theater) variable will be summed over. If category="both", then both variables will be fixed. In this case, the theater name must be provided in "name" parameter and the movie title must be provided in the "movie_name" parameter. Finally, if category is not provided, both the movies and the theaters are summed over.

Name: Only relevant if the "category" parameter is provided. The fixed value of the variable given in the "category" parameter. If category="theater"("movie"), then name is the theater name(movie title). If category="both", then name is the theater name, while the movie title will be given in the parameter "movie_name".

movie_name: Only relevant if category="both". In this case, it corresponds to the fixed movie title.

Examples:

Url: https://localhost:4324/salesovertime?st_date=11-28-2024&end_date=11-30-2024&category=theater&name=AMC Saratoga 14

Explanation: We have fixed the theater, summed over the movies, and returned the total_sales for all dates in the date range provided.

Indices: The index (theater_id, sales_date) ensures fast lookup for this query. After the theater_id is fixed, we run through all of the dates in the date range, summing over all of the movies for each one. This index is subsumed in the aforementioned (theater_id, sales_date, total_sales) index.

Url: https://localhost:4324/salesovertime?st_date=11-28-2024&end_date=11-30-2024&category=movie&name=Wicked

Explanation: We have fixed the movie, summed over the theaters, and returned the total_sales for all dates in the date range provided.

Indices: The index (movie_id, sales_date) ensures fast lookup for this query. After the movie_id is fixed, we run through all of the dates in the date range, summing over all of the theaters for each one. This index is subsumed in the aforementioned (movie_id, sales_date, total_sales) index.

Url: https://localhost:4324/salesovertime?st_date=11-28-2024&end_date=11-30-2024

Explanation: We sum over all theaters and movies, returning the total_sales for each date in the date range provided.

Indices: The index (sales_date) ensures fast lookup for this query. We run through all of the dates in the date range, summing over the theaters and movies for each one.

Url: https://localhost:4324/salesovertime?st_date=11-28-2024&end_date=11-30-2024&category=both&name=Regal LA Live&movie_name=Red One

Explanation: We fix both the theater and the movie, and return total_sales for each date in the date range.

Indices: The unique index (movie_id, theater_id, sales_date) ensures fast lookup for this query. We first fix the movie_id and theater_id, and then run through all of the dates in the date range provided.

Instructions

NodeJS:

1. Configure the file queries.js to your PostgreSQL database server.
2. Run the command “npm install” from the terminal.
3. Run the command “node create_tables.js” from the terminal.
4. Run the command “node insert_data.js” from the terminal.
5. Run the command “node routes.js” from the terminal.
6. Open your browser and visit any of the links above or create your own requests based on the specifications above.

Python:

1. Configure the file queries.py to your PostgreSQL database server.
2. Activate your python environment in the terminal
3. Run the command ‘pip install “Flask[async]"’ from the terminal.
4. Run the command ‘pip install "psycopg[binary,pool]"’ from the terminal.
5. Run the command “python create_tables.py” from the terminal.
6. Run the command “python insert_data.py” from the terminal.
7. Run the command “python routes.py” from the terminal.
8. Open your browser and visit any of the links above or create your own requests based on the specifications above. Replace https://localhost:4324 with http://localhost:5000 .