Building a Movie Database System

1. Before the Change

Query and Execution Plan Without Optimizations

```
Explain analyze SELECT "Title", "ReleaseYear", "Genre"
FROM movies_partitioned
WHERE "Genre" = 'Action' AND "ReleaseYear" > 2015;
```

Execution Plan Details:

- Description of the optimizer plan:
  The query likely triggers a **full table scan** since no secondary indexes or materialized views are defined. Every row in the `Movies` table is scanned to find matching records.
- Identified Performance Issues:
  - **High resource usage** due to scanning the entire table.
  - **Slow query execution time**, especially with a large dataset.
  - Bottlenecks in data retrieval for frequently queried columns (e.g., `genre`, `release_year`).

Performance Metrics Before Optimizations

- **Execution Time**: 5.066 ms for a dataset of 20000 rows.
- **Resource Usage**: High CPU and I/O activity

Here is a suggested structure for your **Movie Database System Report** based on the submission instructions. Each section addresses the specific requirements mentioned:

---

2. After the Change

Query and Execution Plan With Optimizations

- **Changes Implemented**:
  - **Secondary Indexes**: Created indexes on frequently filtered columns:

    ```
    CREATE INDEX idx_movies_genre ON movies_partitioned ("Genre");
    CREATE INDEX idx_movies_director ON movies_partitioned ("Director");
    ```

  - **Materialized View**: Precomputed and stored commonly queried data:

    ```
    CREATE MATERIALIZED VIEW Top_10_Movies_By_Genre AS
    ```

```
SELECT "Genre", "Title", "Id", "Director", "ReleaseYear", "WatchCount",
"Rating"
FROM movies_partitioned
WHERE "Genre" IS NOT NULL
ORDER BY "WatchCount" DESC
LIMIT 10;
```

- **Query Example After Changes**:

```
Explain Analyze SELECT "Title", "Genre", "Director"
FROM movies_partitioned
WHERE "Genre" = 'Horror';
```

Execution Plan Details After Optimizations

- **Description of Optimizer Plan**:
  - Use of **indexes** for faster filtering by `genre` and `director`.
  - **Materialized view** directly stores pre-filtered results, avoiding full scans of the base table.
- **Performance Improvements**:
  - Reduced data retrieval time due to precomputed data in the materialized view.
  - Query now uses index lookups instead of full table scans.

Performance Metrics After Optimizations

- **Execution Time**: 1.353 ms "Bitmap Heap Scan on movies_partitioned (cost=36.39..282.26 rows=3110 width=32) (actual time=0.373..1.210 rows=3110 loops=1)"
- **Resource Usage**: Significant reduction in CPU and I/O activity.

---

3. For Partitioning and API

Partitioning and Performance Improvements

Partitioning Approach:

- Partitioned the `Movies` table by `release_year` for improved performance on year-based queries

Query Execution Plan:

- Queries now target specific partitions, significantly reducing the data scanned.

API Documentation for Movies

This section provides details about the **Movies API** endpoints, including their purpose, methods, request parameters, and example responses.

1. GET /api/Movies

- **Description**: Retrieves a list of all movies in the database.
- **Method**: `GET`
- **Response Example**:


```
[
    {
      "id": 1,
      "title": "Inception",
      "genre": "Sci-Fi",
      "director": "Christopher Nolan",
      "releaseYear": 2010,
      "watchCount": 1200000,
      "rating": 9.0
    },
    {
      "id": 2,
      "title": "The Dark Knight",
      "genre": "Action",
      "director": "Christopher Nolan",
      "releaseYear": 2018,
      "watchCount": 2000000,
      "rating": 9.1
    },
]
```


---

2. POST /api/Movies

- **Description**: Adds a new movie to the database.
- **Method**: `POST`
- **Request Body** (example):


```
{
  "id": 5,
  "title": "The Godfather",
  "genre": "Crime",
  "director": "Francis Ford Coppola",
  "releaseYear": 2020,
  "watchCount": 2500000,
  "rating": 9.2
}
```


---

3. GET /api/Movies/{id}

- **Description**: Retrieves details for a specific movie by its ID.
- **Method**: `GET`

- **Path Parameter**:
  - `{id}`: The unique identifier of the movie (integer).
- **Response Example**:

```
{
   "id": 1,
   "title": "Inception",
    "genre": "Sci-Fi",
   "director": "Christopher Nolan",
    "releaseYear": 2010,
    "watchCount": 1200000,
    "rating": 9.0
}
```

---

4. PUT /api/Movies/{id}

- **Description**: Updates an existing movie's details.
- **Method**: `PUT`
- **Path Parameter**:
  - `{id}`: The unique identifier of the movie to update (integer).
- **Request Body** (example):

```
{
   "id": 1,
   "title": "Inception",
    "genre": "Sci-Fi",
   "director": "Christopher Nolan",
    "releaseYear": 2020 (Updated),
    "watchCount": 47859 (Updated) ,
    "rating": 9.0
}
```

- **Response Example**:

```
{
      "id": 1,
      "title": "Inception",
       "genre": "Sci-Fi",
      "director": "Christopher Nolan",
       "releaseYear": 2020 (Updated) ,
       "watchCount": 47859 ,
       "rating": 9.0
}
```

---

5. DELETE /api/Movies/{id}

- **Description**: Deletes a specific movie from the database.
- **Method**: `DELETE`
- **Path Parameter**:
  - `{id}`: The unique identifier of the movie to delete (integer).
- **Response Example**:

```
{
  "message": "Movie with ID 1 was deleted successfully."
}
```

---

6. GET /api/Movies/search

- **Description**: Searches for movies based on query parameters.
- **Method**: `GET`
- **Query Parameters**:
  - `genre`: Filter by genre.
  - `Director`: Filter by director.
- **Example Request**:

```
GET api/Movies/search?genre=Comedy&director=Director_246' \
```

Response Example:


```
[
   {
     "id": 20662,
      "title": "Movie_1020661",
       "genre": "Comedy",
        "director": "Director_246",
        "releaseYear": 2010,
        "watchCount": 730,
        "rating": 9.5
    },
    {
       "id": 25310,
       "title": "Movie_1025309",
       "genre": "Comedy",
       "director": "Director_246",
       "releaseYear": 2015,
       "watchCount": 60,
       "rating": 5.5
     }
]
```