

## Database Design

### Data Definition Language Commands

```
CREATE TABLE user(  
    username VARCHAR(32) PRIMARY KEY NOT NULL,  
    email VARCHAR(64),  
    password VARCHAR(128));
```

```
CREATE TABLE admin(  
    username VARCHAR(32) REFERENCES user(username) ON DELETE CASCADE,  
    adminId INT PRIMARY KEY,  
    joiningDate DATE);
```

```
CREATE TABLE movie(  
    movieId INT PRIMARY KEY NOT NULL,  
    movieName VARCHAR(150),  
    description VARCHAR(1024),  
    releaseDate DATE,  
    adminId INT REFERENCES admin(adminId) ON DELETE NULL);
```

```
CREATE TABLE shows(  
    showId INT PRIMARY KEY NOT NULL,  
    movieId INT REFERENCES movie(movieId) ON DELETE CASCADE,  
    showTiming DATETIME,  
    screen INT,  
    price REAL,  
    adminId INT REFERENCES admin(adminId) ON DELETE NULL);
```

```
CREATE TABLE booking(  
    bookingId INT PRIMARY KEY NOT NULL,  
    username VARCHAR REFERENCES user(username) ON DELETE NULL,  
    timestamp DATETIME,  
    showId INT REFERENCES shows(showId) ON DELETE NULL);
```

```
CREATE TABLE seat(  
    seatId INT NOT NULL,  
    showId INT REFERENCES shows(showId) ON DELETE NULL,  
    bookingId INT REFERENCES booking(bookingId) ON DELETE NULL,  
    availability BOOLEAN,  
    PRIMARY KEY(seatId, showId));
```

## Connection Screenshots

### Database:

| Table                                   | Action | Rows   | Type   | Collation          | Size      | Overhead |
|---|--------|--------|--------|--------------------|-----------|----------|
| <input type="checkbox"/> <b>admin</b>   |        | 4      | InnoDB | utf8mb4_general_ci | 32.0 KiB  | -        |
| <input type="checkbox"/> <b>booking</b> |        | 7,200  | InnoDB | utf8mb4_general_ci | 784.0 KiB | -        |
| <input type="checkbox"/> <b>movie</b>   |        | 2,515  | InnoDB | utf8mb4_general_ci | 1.7 MiB   | -        |
| <input type="checkbox"/> <b>seat</b>    |        | 45,000 | InnoDB | utf8mb4_general_ci | 3.0 MiB   | -        |
| <input type="checkbox"/> <b>shows</b>   |        | 1,800  | InnoDB | utf8mb4_general_ci | 272.0 KiB | -        |
| <input type="checkbox"/> <b>user</b>    |        | 2,501  | InnoDB | utf8mb4_general_ci | 304.0 KiB | -        |
| 6 tables                                | Sum    | 59,020 | InnoDB | utf8mb4_general_ci | 6.1 MiB   | 0 B      |

### User Table:

| #                          | Name            | Type         | Collation          | Attributes | Null | Default | Comments | Extra | Action             |
|----------------------------|-----------------|--------------|--------------------|------------|------|---------|----------|-------|--------------------|
| <input type="checkbox"/> 1 | <b>username</b> | varchar(32)  | utf8mb4_general_ci |            | No   | None    |          |       | Change  Drop  More |
| <input type="checkbox"/> 2 | <b>email</b>    | varchar(64)  | utf8mb4_general_ci |            | No   | None    |          |       | Change  Drop  More |
| <input type="checkbox"/> 3 | <b>password</b> | varchar(128) | utf8mb4_general_ci |            | No   | None    |          |       | Change  Drop  More |






















✓ Showing rows 0 - 24 (2501 total, Query took 0.0010 seconds.)

```
SELECT * FROM `user`
```

☐ Profiling [\[ Edit inline \]](#) [\[ Edit \]](#) [\[ Explain SQL \]](#) [\[ Create PHP code \]](#) [\[ Refresh \]](#)

1 > >> | Number of rows: 25 | Filter rows:  | Sort by key: None

Extra options

|                          |  |  |  | username         | email                          | password                         |
|--------------------------|--|--|--|------------------|--------------------------------|----------------------------------|
| <input type="checkbox"/> |  Edit |  Copy |  Delete | able-emili       | emili.Paille@gmail.com         | a303df6ce4e8f48b22b19060b3f7d010 |
| <input type="checkbox"/> |  Edit |  Copy |  Delete | able-patricia    | patricia.Langehennig@gmail.com | 940806728cdcff6d05b306f761ec3888 |
| <input type="checkbox"/> |  Edit |  Copy |  Delete | absent-chloette  | chloette.Brantner@illinois.edu | 7615eb5435c63be4278061059f062276 |
| <input type="checkbox"/> |  Edit |  Copy |  Delete | absent-jillene   | jillene.Beltron@illinois.edu   | 01eb31cb933d823756029d37ea922204 |
| <input type="checkbox"/> |  Edit |  Copy |  Delete | absolute-janot   | janot.Russell@illinois.edu     | c9b1c32e9f0da54214f36c72105be8d3 |
| <input type="checkbox"/> |  Edit |  Copy |  Delete | abstract-junina  | junina.Coleman@gmail.com       | 055d5257cc43577bde2a295e8e012d1a |
| <input type="checkbox"/> |  Edit |  Copy |  Delete | abstract-kyrstin | kyrstin.Washington@hotmail.com | 76ef3a85cd4e62bdd144874c374f85c9 |

### Admin Table:

|                          | # | Name               | Type        | Collation          | Attributes | Null | Default | Comments | Extra | Action             |
|--------------------------|---|--------------------|-------------|--------------------|------------|------|---------|----------|-------|--------------------|
| <input type="checkbox"/> | 1 | <b>username</b> 🔑  | varchar(32) | utf8mb4_general_ci |            | No   | None    |          |       | Change  Drop  More |
| <input type="checkbox"/> | 2 | <b>adminId</b> 🔑   | int(11)     |                    |            | No   | None    |          |       | Change  Drop  More |
| <input type="checkbox"/> | 3 | <b>joiningDate</b> | date        |                    |            | No   | None    |          |       | Change  Drop  More |

✓ Showing rows 0 - 3 (4 total, Query took 0.0007 seconds.)

```
SELECT * FROM `admin`
```

☐ Profiling [ [Edit inline](#) ] [ [Edit](#) ] [ [Explain SQL](#) ] [ [Create PHP code](#) ] [ [Refresh](#) ]

☐ Show all | Number of rows:  Filter rows:

Extra options

|                          |      |      |        | username | adminId | joiningDate |
|--------------------------|------|------|--------|----------|---------|-------------|
| <input type="checkbox"/> | Edit | Copy | Delete | rishisg2 | 1       | 2022-10-01  |
| <input type="checkbox"/> | Edit | Copy | Delete | vchheda2 | 2       | 2022-09-19  |
| <input type="checkbox"/> | Edit | Copy | Delete | amb20    | 3       | 2012-10-03  |
| <input type="checkbox"/> | Edit | Copy | Delete | pma7     | 4       | 2022-10-16  |

### Movie Table:

|                          | # | Name               | Type          | Collation          | Attributes | Null | Default | Comments | Extra | Action             |
|--------------------------|---|--------------------|---------------|--------------------|------------|------|---------|----------|-------|--------------------|
| <input type="checkbox"/> | 1 | <b>movieId</b> 🔑   | varchar(32)   | utf8mb4_general_ci |            | No   | None    |          |       | Change  Drop  More |
| <input type="checkbox"/> | 2 | <b>movieName</b>   | varchar(150)  | utf8mb4_general_ci |            | No   | None    |          |       | Change  Drop  More |
| <input type="checkbox"/> | 3 | <b>description</b> | varchar(1024) | utf8mb4_general_ci |            | No   | None    |          |       | Change  Drop  More |
| <input type="checkbox"/> | 4 | <b>releaseDate</b> | date          |                    |            | No   | None    |          |       | Change  Drop  More |
| <input type="checkbox"/> | 5 | <b>adminId</b> 🔑   | int(11)       |                    |            | No   | None    |          |       | Change  Drop  More |

Showing rows 0 - 24 (2515 total, Query took 0.0022 seconds.)

```
SELECT * FROM `movie`
```

☐ Profiling [ [Edit inline](#) ] [ [Edit](#) ] [ [Explain SQL](#) ] [ [Create PHP code](#) ] [ [Refresh](#) ]
















1 > >> | Number of rows: 25 Filter rows: Search this table Sort by key: None

1 > >> | Number of rows: 25 Filter rows: Search this table Sort by key: None
















1 > >> | Number of rows: 25 Filter rows: Search this table Sort by key: None

1 > >> | Number of rows: 25 Filter rows: Search this table Sort by key: None

Extra options

|                          |   |   |   | movieId                          | movieName                       | description   | releaseDate | adminId |
|--------------------------|---|---|---|----------------------------------|---------------------------------|---|-------------|---------|
| <input type="checkbox"/> |  |  |  | m/10                             | 10                              | A successful, middle-aged Hollywood songwriter fal... | 2022-09-28  | 2       |
| <input type="checkbox"/> |  |  |  | m/1000013-12_angry_men           | 12 Angry Men (Twelve Angry Men) | Following the closing arguments in a murder trial,... | 2022-09-28  | 3       |
| <input type="checkbox"/> |  |  |  | m/1000079-20000_leagues_under_th | 20,000 Leagues Under The Sea    | In 1866, Professor Pierre M. Aronnax (Paul Lukas) ... | 2022-09-27  | 3       |
| <input type="checkbox"/> |  |  |  | m/10000_bc                       | 10,000 B.C.                     | Mammoth hunter D'Leh (Steven Strait) has long been... | 2022-09-27  | 1       |
| <input type="checkbox"/> |  |  |  | m/1000121-39_steps               | The 39 Steps                    | While on vacation in London, Canadian Richard Hann... | 2022-09-27  | 1       |
| <input type="checkbox"/> |  |  |  | m/1000123-310_to_yuma            | 3:10 to Yuma                    | Dan Evans (Van Heflin), a drought-plagued Arizona ... | 2022-09-27  | 2       |
| <input type="checkbox"/> |  |  |  | m/10002008-charly                | Charly (A Heartbeat Away)       | Cultural differences, past loves and personal cris... | 2022-09-25  | 4       |
| <input type="checkbox"/> |  |  |  | m/1000204-abraham_lincoln        | Abraham Lincoln                 | The 16th U.S. president (Walter Huston) is portray... | 2022-09-25  | 2       |
| <input type="checkbox"/> |  |  |  | m/10002114-dark_water            | Dark Water                      | In this moody Japanese horror film, newly-single m... | 2022-09-25  | 4       |
| <input type="checkbox"/> |  |  |  | m/1000224-accused                | The Accused                     | Out drinking one night after a fight with her boyf... | 2022-09-25  | 2       |
| <input type="checkbox"/> |  |  |  | m/10002516-lost_city             | The Lost City                   | Fico Fellove (Andy Garcia), an apolitical Havana c... | 2022-09-25  | 2       |
| <input type="checkbox"/> |  |  |  | m/10002519-breaking_point        | The Breaking Point              | A charter-boat captain winds up in the middle of a... | 2022-09-24  | 3       |
| <input type="checkbox"/> |  |  |  | m/1000253-adams_rib              | Adam's Rib                      | A courtroom rivalry finds its way into the househo... | 2022-09-24  | 1       |

Shows Table:

|                          | # | Name  | Type        | Collation          | Attributes | Null | Default | Comments | Extra | Action   |
|--------------------------|---|---|-------------|--------------------|------------|------|---------|----------|-------|--|
| <input type="checkbox"/> | 1 | showId   | int(11)     |                    |            | No   | None    |          |       |  Change  Drop <a href="#">More</a> |
| <input type="checkbox"/> | 2 | movieId  | varchar(32) | utf8mb4_general_ci |            | No   | None    |          |       |  Change  Drop <a href="#">More</a> |
| <input type="checkbox"/> | 3 | showTiming  | datetime    |                    |            | Yes  | NULL    |          |       |  Change  Drop <a href="#">More</a> |
| <input type="checkbox"/> | 4 | screen  | int(11)     |                    |            | Yes  | NULL    |          |       |  Change  Drop <a href="#">More</a> |
| <input type="checkbox"/> | 5 | price   | double      |                    |            | Yes  | NULL    |          |       |  Change  Drop <a href="#">More</a> |
| <input type="checkbox"/> | 6 | adminId  | int(11)     |                    |            | Yes  | NULL    |          |       |  Change  Drop <a href="#">More</a> |

✓ Showing rows 0 - 24 (1800 total, Query took 0.0019 seconds.)

`SELECT * FROM `shows``

☐ Profiling [ [Edit inline](#) ] [ [Edit](#) ] [ [Explain SQL](#) ] [ [Create PHP code](#) ] [ [Refresh](#) ]

1 ▾

> >>



























Number of rows:

25 ▾












Filter rows:

Sort by key:

Extra options

| <div><div>←T→</div><div>▼</div></div> |  |  |  | showId | movieId                 | showTiming          | screen | price | adminId |
|---------------------------------------|--|--|--|--------|-------------------------|---------------------|--------|-------|---------|
| <input type="checkbox"/>              |  Edit   |  Copy   |  Delete   | 1      | m/1042466-boiling_point | 2022-08-01 01:00:00 | 1      | 14    | 4       |
| <input type="checkbox"/>              |  Edit   |  Copy   |  Delete   | 2      | m/1042466-boiling_point | 2022-09-01 01:00:00 | 1      | 14    | 4       |
| <input type="checkbox"/>              |  Edit   |  Copy   |  Delete   | 3      | m/1042466-boiling_point | 2022-10-01 01:00:00 | 1      | 14    | 4       |
| <input type="checkbox"/>              |  Edit   |  Copy   |  Delete   | 4      | m/1042466-boiling_point | 2022-11-01 01:00:00 | 1      | 14    | 4       |
| <input type="checkbox"/>              |  Edit   |  Copy   |  Delete   | 5      | m/1042466-boiling_point | 2022-08-01 01:00:00 | 2      | 14    | 4       |
| <input type="checkbox"/>              |  Edit   |  Copy   |  Delete   | 6      | m/1042466-boiling_point | 2022-09-01 01:00:00 | 2      | 14    | 4       |
| <input type="checkbox"/>              |  Edit   |  Copy   |  Delete   | 7      | m/1042466-boiling_point | 2022-10-01 01:00:00 | 2      | 14    | 4       |
| <input type="checkbox"/>              |  Edit  |  Copy  |  Delete  | 8      | m/1042466-boiling_point | 2022-11-01 01:00:00 | 2      | 14    | 4       |
| <input type="checkbox"/>              |  Edit |  Copy |  Delete | 9      | m/1042466-boiling_point | 2022-08-01 01:00:00 | 3      | 14    | 4       |
| <input type="checkbox"/>              |  Edit |  Copy |  Delete | 10     | m/1042466-boiling_point | 2022-09-01 01:00:00 | 3      | 14    | 4       |
| <input type="checkbox"/>              |  Edit |  Copy |  Delete | 11     | m/1042466-boiling_point | 2022-10-01 01:00:00 | 3      | 14    | 4       |
| <input type="checkbox"/>              |  Edit |  Copy |  Delete | 12     | m/1042466-boiling_point | 2022-11-01 01:00:00 | 3      | 14    | 4       |
| <input type="checkbox"/>              |  Edit |  Copy |  Delete | 13     | m/1042466-boiling_point | 2022-08-01 01:00:00 | 4      | 14    | 4       |

## Booking Table:

|                          | # | Name  | Type        | Collation          | Attributes | Null | Default | Comments | Extra | Action   |
|--------------------------|---|---|-------------|--------------------|------------|------|---------|----------|-------|--|
| <input type="checkbox"/> | 1 | bookingId  | int(11)     |                    |            | No   | None    |          |       |  Change  Drop <a href="#">More</a> |
| <input type="checkbox"/> | 2 | username   | varchar(32) | utf8mb4_general_ci |            | No   | None    |          |       |  Change  Drop <a href="#">More</a> |
| <input type="checkbox"/> | 3 | timestamp   | datetime    |                    |            | No   | None    |          |       |  Change  Drop <a href="#">More</a> |
| <input type="checkbox"/> | 4 | showId     | int(11)     |                    |            | No   | None    |          |       |  Change  Drop <a href="#">More</a> |

✓ Showing rows 0 - 24 (7200 total, Query took 0.0018 seconds.)

```
SELECT * FROM `booking`
```

☐ Profiling [ [Edit inline](#) ] [ [Edit](#) ] [ [Explain SQL](#) ] [ [Create PHP code](#) ] [ [Refresh](#) ]

1

>

>>


































Number of rows:

25











Filter rows:

Search this table

Extra options

|                          |   |                      |   | bookingId            | username  | timestamp              | showId |
|--------------------------|---|----------------------|---|----------------------|---|------------------------|--------|
| <input type="checkbox"/> |    | <a href="#">Edit</a> |    | <a href="#">Copy</a> |    | <a href="#">Delete</a> |        |
|                          | 1   | subtle-vanessa       | 2022-08-01 00:00:00   | 1                    |   |                        |        |
| <input type="checkbox"/> |    | <a href="#">Edit</a> |    | <a href="#">Copy</a> |    | <a href="#">Delete</a> |        |
|                          | 2   | slight-jen           | 2022-08-01 00:00:00   | 1                    |   |                        |        |
| <input type="checkbox"/> |    | <a href="#">Edit</a> |    | <a href="#">Copy</a> |    | <a href="#">Delete</a> |        |
|                          | 3   | global-tessie        | 2022-08-01 00:00:00   | 1                    |   |                        |        |
| <input type="checkbox"/> |    | <a href="#">Edit</a> |    | <a href="#">Copy</a> |    | <a href="#">Delete</a> |        |
|                          | 4   | unable-tildie        | 2022-08-01 00:00:00   | 1                    |   |                        |        |
| <input type="checkbox"/> |    | <a href="#">Edit</a> |    | <a href="#">Copy</a> |    | <a href="#">Delete</a> |        |
|                          | 5   | ytterbic-joana       | 2022-09-01 00:00:00   | 2                    |   |                        |        |
| <input type="checkbox"/> |  | <a href="#">Edit</a> |  | <a href="#">Copy</a> |  | <a href="#">Delete</a> |        |
|                          | 6   | universal-anastasia  | 2022-09-01 00:00:00   | 2                    |   |                        |        |
| <input type="checkbox"/> |  | <a href="#">Edit</a> |  | <a href="#">Copy</a> |  | <a href="#">Delete</a> |        |
|                          | 7   | round-tresa          | 2022-09-01 00:00:00   | 2                    |   |                        |        |
| <input type="checkbox"/> |  | <a href="#">Edit</a> |  | <a href="#">Copy</a> |  | <a href="#">Delete</a> |        |
|                          | 8   | junior-alissa        | 2022-09-01 00:00:00   | 2                    |   |                        |        |
| <input type="checkbox"/> |  | <a href="#">Edit</a> |  | <a href="#">Copy</a> |  | <a href="#">Delete</a> |        |
|                          | 9   | large-petrina        | 2022-10-01 00:00:00   | 3                    |   |                        |        |
| <input type="checkbox"/> |  | <a href="#">Edit</a> |  | <a href="#">Copy</a> |  | <a href="#">Delete</a> |        |
|                          | 10  | rude-agata           | 2022-10-01 00:00:00   | 3                    |   |                        |        |
| <input type="checkbox"/> |  | <a href="#">Edit</a> |  | <a href="#">Copy</a> |  | <a href="#">Delete</a> |        |
|                          | 11  | bottom-maxi          | 2022-10-01 00:00:00   | 3                    |   |                        |        |

### Seat Table:

|                          | # | Name   | Type       | Collation | Attributes | Null | Default | Comments | Extra | Action   |
|--------------------------|---|--|------------|-----------|------------|------|---------|----------|-------|--|
| <input type="checkbox"/> | 1 | seatId  | int(11)    |           |            | No   | None    |          |       |  <a href="#">Change</a>  <a href="#">Drop</a> <a href="#">More</a> |
| <input type="checkbox"/> | 2 | bookingId  | int(11)    |           |            | No   | None    |          |       |  <a href="#">Change</a>  <a href="#">Drop</a> <a href="#">More</a> |
| <input type="checkbox"/> | 3 | showId  | int(11)    |           |            | No   | None    |          |       |  <a href="#">Change</a>  <a href="#">Drop</a> <a href="#">More</a> |
| <input type="checkbox"/> | 4 | availability   | tinyint(1) |           |            | No   | None    |          |       |  <a href="#">Change</a>  <a href="#">Drop</a> <a href="#">More</a> |

✓ Showing rows 0 - 24 (45000 total, Query took 0.0017 seconds.)

`SELECT * FROM `seat``

☐ Profiling [ [Edit inline](#) ] [ [Edit](#) ] [ [Explain SQL](#) ] [ [Create PHP code](#) ] [ [Refresh](#) ]

1 > >> | Number of rows: 25 Filter rows:

Extra options

|                          |  |  |  | seatId | bookingId | showId | availability |
|--------------------------|--|--|--|--------|-----------|--------|--------------|
| <input type="checkbox"/> |  |  |  | 1      | 3         | 1      | 0            |
| <input type="checkbox"/> |  |  |  | 1      | 8         | 2      | 0            |
| <input type="checkbox"/> |  |  |  | 1      | 9         | 3      | 0            |
| <input type="checkbox"/> |  |  |  | 1      | 16        | 4      | 0            |
| <input type="checkbox"/> |  |  |  | 1      | 18        | 5      | 0            |
| <input type="checkbox"/> |  |  |  | 1      | 24        | 6      | 0            |
| <input type="checkbox"/> |  |  |  | 1      | 27        | 7      | 0            |
| <input type="checkbox"/> |  |  |  | 1      | 29        | 8      | 0            |
| <input type="checkbox"/> |  |  |  | 1      | 34        | 9      | 0            |
| <input type="checkbox"/> |  |  |  | 1      | 39        | 10     | 0            |
| <input type="checkbox"/> |  |  |  | 1      | 44        | 11     | 0            |

We have the following number of records in each table:

- User – 2501 records
- Admin – 4 records
- Movie – 2515 records
- Shows – 1800 records
- Booking – 7200 records
- Seat – 45000 records

The data in movie table is imported from the Rotten Tomatoes database, while other tables are populated using auto-generated data (scripts added in the appendix section at the end).

## Advanced Queries

### Query 1: Checking seat availability for all shows of movies

```
SELECT m.movieName, s.showTiming, s.screen, COUNT(seat.seatId) as Availability
FROM movie m JOIN shows s USING(movieId) JOIN seat USING(showId)
WHERE seat.availability = 1
GROUP BY m.movieName, s.showTiming, s.Screen
LIMIT 15;
```

*NOTE: LIMIT 15 is added to match the task description, actual query would not have this limit.*

```
mysql> SELECT m.movieName, s.showTiming, s.screen, COUNT(seat.seatId) as Availability
-> FROM movie m JOIN shows s USING(movieId) JOIN seat USING(showId)
-> WHERE seat.availability = 1
-> GROUP BY m.movieName, s.showTiming, s.screen
-> LIMIT 15;
```

| movieName     | showTiming          | screen | Availability |
|---------------|---------------------|--------|--------------|
| Boiling Point | 2022-08-01 01:00:00 | 1      | 7            |
| Boiling Point | 2022-09-01 01:00:00 | 1      | 7            |
| Boiling Point | 2022-10-01 01:00:00 | 1      | 11           |
| Boiling Point | 2022-11-01 01:00:00 | 1      | 5            |
| Boiling Point | 2022-08-01 01:00:00 | 2      | 11           |
| Boiling Point | 2022-09-01 01:00:00 | 2      | 6            |
| Boiling Point | 2022-10-01 01:00:00 | 2      | 6            |
| Boiling Point | 2022-11-01 01:00:00 | 2      | 13           |
| Boiling Point | 2022-09-01 01:00:00 | 3      | 6            |
| Boiling Point | 2022-10-01 01:00:00 | 3      | 11           |
| Boiling Point | 2022-08-01 01:00:00 | 4      | 11           |
| Boiling Point | 2022-09-01 01:00:00 | 4      | 8            |
| Boiling Point | 2022-11-01 01:00:00 | 4      | 6            |
| Boiling Point | 2022-08-01 01:00:00 | 5      | 10           |
| Boiling Point | 2022-09-01 01:00:00 | 5      | 6            |

15 rows in set (0.60 sec)

## Indexing on query 1

### Default indexing

```
| -> Table scan on <temporary> (actual time=0.001..0.036 rows=360 loops=1)
-> Aggregate using temporary table (actual time=61.901..61.956 rows=360 loops=1)
-> Nested loop inner join (cost=7862.08 rows=4531) (actual time=2.653..48.357 rows=14234 loops=1)
-> Nested loop inner join (cost=6276.30 rows=4531) (actual time=2.635..39.237 rows=14234 loops=1)
-> Filter: (seat.availability = 1) (cost=4690.52 rows=4531) (actual time=2.617..13.957 rows=14234 loops=1)
-> Table scan on seat (cost=4690.52 rows=45308) (actual time=0.042..10.856 rows=45000 loops=1)
-> Index lookup on s using PRIMARY (showId=seat.showId) (cost=0.25 rows=1) (actual time=0.001..0.002 rows=1 loops=14234)
-> Single-row index lookup on m using PRIMARY (movieId=s.movieId) (cost=0.25 rows=1) (actual time=0.000..0.000 rows=1 loops=14234)
```



## Indexing on seat(availability)

CREATE INDEX avail\_idx ON seat(availability)

```
| -> Table scan on <temporary> (actual time=0.003..0.034 rows=360 loops=1)
    -> Aggregate using temporary table (actual time=49.546..49.598 rows=360 loops=1)
        -> Nested loop inner join (cost=18148.59 rows=22654) (actual time=0.113..36.639 rows=14234 loops=1)
            -> Nested loop inner join (cost=10219.69 rows=22654) (actual time=0.093..27.909 rows=14234 loops=1)
                -> Index lookup on seat using avail_idx (availability=1) (cost=2290.79 rows=22654) (actual time=0.077..3.610 rows=14234 loops=1)
                    -> Index lookup on s using PRIMARY (showId=seat.showId) (cost=0.25 rows=1) (actual time=0.001..0.002 rows=1 loops=14234)
                -> Single-row index lookup on m using PRIMARY (movieId=s.movieId) (cost=0.25 rows=1) (actual time=0.000..0.000 rows=1 loops=14234)
```

After creating an index on Availability attribute of Seat relation we note the following things:

The pace of accessing the first record decreased for the table scan, however the actual execution time of the operation reduced. Digging deeper and trying to analyze why creating an index on seat availability improves performance reveals one major factor. The aggregation operation in original query takes 61.901ms, (larger than the next costliest operation of 48.35ms by 28%) and is the main point of optimization in the query. Indexing on seats helps us optimize this particular operation by reducing the time required to access the first record by approximately 12 seconds or roughly 20%). Further going down the subtrees, we can see that time to fetch the first record reduces by a magnitude of almost 23x and execution time by 1/3rd. Similar results are reciprocated in the second nested loop. Additionally, in the subtree structure, the query optimizer uses a combination of an index lookup, a filter, and a table scan, which require about 52ms, and this structure is supplanted by two index lookups in the indexed table. While one index lookup (taking  $0.002\text{ms} * 14234 = 28\text{ms}$ ) is similar to the original database, the remaining structure take about 3.6ms and is quicker by over 20 seconds. However, a caveat to note is that in the query running on the indexed database, the query optimizers accuracy is taking a hit. The query optimizer predicts wrongly on multiple occasions the number of rows it must check by approximately 37% points (predicts 22,654, while it actually checks 14,234).

## Indexing on movie(movieName)

CREATE INDEX movie\_idx ON movie(movieName(5))

```
| -> Table scan on <temporary> (actual time=0.002..0.239 rows=1797 loops=1)
    -> Aggregate using temporary table (actual time=68.206..68.549 rows=1797 loops=1)
        -> Nested loop inner join (cost=7862.08 rows=4531) (actual time=2.648..50.040 rows=14234 loops=1)
            -> Nested loop inner join (cost=6276.30 rows=4531) (actual time=2.629..40.618 rows=14234 loops=1)
                -> Filter: (seat.availability = 1) (cost=4690.52 rows=4531) (actual time=2.607..14.254 rows=14234 loops=1)
                    -> Table scan on seat (cost=4690.52 rows=45308) (actual time=0.072..11.074 rows=45000 loops=1)
                -> Index lookup on s using PRIMARY (showId=seat.showId) (cost=0.25 rows=1) (actual time=0.001..0.002 rows=1 loops=14234)
            -> Single-row index lookup on m using PRIMARY (movieId=s.movieId) (cost=0.25 rows=1) (actual time=0.000..0.000 rows=1 loops=14234)
```

After creating an index on movie index attribute of Movie relation we note the following things:

The pace of accessing the first record decreased, along with the time required for executing the Table scan. Digging deeper and trying to analyze why creating an index on movie indexes decreases performance reveals one major factor. The aggregation operation in original query takes 61.901ms, (larger than the next costliest operation of 48.35ms by 28%) and is the main point of optimization in the query. Indexing on movie impedes this operation by increasing the time

required to access the first record by approximately 7 seconds). Furthermore, indexing on movie actually causes the query optimizer to check approximately 1400 more records. Further going down the subtrees, we can see that time to fetch the first record is similar to the original query, but the overall execution time actually shoots up from ~87ms to ~91ms. Additionally, in the deeper subtree structure, the query optimizer uses a combination of an index lookup, a filter, and a table scan, which require about 52ms, and this structure is identical for the indexed table. However, for this similar structure in the indexed table, the query optimizer performs worse as the execution time further increasing by ~1ms.

### Indexing on show(showTiming)

CREATE INDEX show\_idx ON show(showTiming)

```
| -> Table scan on <temporary> (actual time=0.003..0.051 rows=360 loops=1)
  -> Aggregate using temporary table (actual time=71.375..71.453 rows=360 loops=1)
    -> Nested loop inner join (cost=7862.08 rows=4531) (actual time=2.693..55.163 rows=14234 loops=1)
      -> Nested loop inner join (cost=6276.30 rows=4531) (actual time=2.669..43.937 rows=14234 loops=1)
        -> Filter: (seat.availability = 1) (cost=4690.52 rows=4531) (actual time=2.628..14.633 rows=14234 loops=1)
          -> Table scan on seat (cost=4690.52 rows=45308) (actual time=0.146..11.416 rows=45000 loops=1)
            -> Index lookup on s using PRIMARY (showId=seat.showId) (cost=0.25 rows=1) (actual time=0.001..0.002 rows=1 loops=14234)
              -> Single-row index lookup on m using PRIMARY (movieId=s.movieId) (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=14234)
```

After creating an index on show index attribute of Show relation we note the following things:

The pace of accessing the first record decreased, along with the time required for executing the Table scan. Digging deeper and trying to analyze why creating an index on movie indexes decreases performance reveals one major factor. The aggregation operation in original query takes 61.901ms, (larger than the next costliest operation of 48.35ms by 28%) and is the main point of optimization in the query. Indexing on movie impedes this operation by increasing the time required to access the first record by approximately 11 seconds. Further going down the subtrees, and looking at the nested loops for join, we can see that time to fetch the first record is similar to the original query, but the overall execution time actually shoots up from ~87ms to ~98ms (or approximately, 13%). Additionally, in the deeper subtree structure, the query optimizer uses a combination of an index lookup, a filter, and a table scan, which require about 52ms, and this structure is identical for the indexed table. However, for this similar structure in the indexed table, the query optimizer performs worse as the execution time further increasing by ~2ms for accessing the same number of records. Hence, with an increase in the number of the records, we can expect the performance to deteriorate at a sub-linear rate.

## Query 2: Find the top trending/popular movies

```
SELECT m.movieName as TopTrendingMovies
FROM movie m JOIN shows s USING(movieId) JOIN booking b USING(showId)
GROUP BY m.movieName
ORDER BY SUM(b.bookingId) DESC
LIMIT 15;
```

*NOTE: LIMIT 15 is added to match the task description, actual query would not have this limit.*

```
mysql> SELECT m.movieName as TopTrendingMovies
-> FROM movie m JOIN shows s USING(movieId) JOIN booking b USING(showId)
-> GROUP BY m.movieName
-> ORDER BY SUM(b.bookingId) DESC
-> LIMIT 15;

+-----+
| TopTrendingMovies |
+-----+
| Hamlet            |
| Everyday People   |
| Cry, the Beloved Country |
| The Babysitter    |
| Salaam Namaste    |
| Body Parts        |
| Mr. Untouchable   |
| A Wake In Providence (Almost Married) |
| Why We Fight      |
| A Tuba to Cuba    |
| All About the Benjamins |
| George Washington |
| A Clear Shot      |
| 7 Days (Les 7 jours du talion) |
| The Island of Dr. Moreau |
+-----+
15 rows in set (0.01 sec)
```

## Indexing on query 2

### Default indexing

```
| -> Sort: 'sum(b.bookingId)' DESC (actual time=11.941..11.947 rows=89 loops=1)
  -> Table scan on <temporary> (actual time=0.002..0.028 rows=89 loops=1)
    -> Aggregate using temporary table (actual time=11.854..11.885 rows=89 loops=1)
      -> Nested loop inner join (cost=1989.93 rows=7261) (actual time=0.122..6.857 rows=7200 loops=1)
        -> Nested loop inner join (cost=812.50 rows=1800) (actual time=0.108..1.493 rows=1800 loops=1)
          -> Index scan on s using adminId (cost=182.50 rows=1800) (actual time=0.043..0.466 rows=1800 loops=1)
            -> Single-row index lookup on m using PRIMARY (movieId=s.movieId) (cost=0.25 rows=1) (actual time=0.000..0.000 rows=1 loops=1800)
          -> Index lookup on b using showId (showId=s.showId) (cost=0.25 rows=4) (actual time=0.002..0.003 rows=4 loops=1800)
```

## Indexing on movie(movieName)

CREATE INDEX movie\_idx ON movie(movieName(5))

```
| -> Sort: `sum(b.bookingId)` DESC (actual time=11.541..11.548 rows=89 loops=1)
| -> Table scan on <temporary> (actual time=0.003..0.029 rows=89 loops=1)
|   -> Aggregate using temporary table (actual time=11.448..11.480 rows=89 loops=1)
|     -> Nested loop inner join (cost=1989.93 rows=7261) (actual time=0.242..6.771 rows=7200 loops=1)
|       -> Nested loop inner join (cost=812.50 rows=1800) (actual time=0.229..1.609 rows=1800 loops=1)
|         -> Index scan on s using adminId (cost=182.50 rows=1800) (actual time=0.197..0.648 rows=1800 loops=1)
|         -> Single-row index lookup on m using PRIMARY (movieId=s.movieId) (cost=0.25 rows=1) (actual time=0.000..0.000 rows=1 loops=1800)
|       -> Index lookup on b using showId (showId=s.showId) (cost=0.25 rows=4) (actual time=0.002..0.002 rows=4 loops=1800)
|
```

After creating an index on movieName attribute, we notice that there is not a very large difference in the time. The first improvement in performance happens at the aggregation operation with a comparatively lower time of 0.4 s. At the second inner join, performance worsens and further worsens at the index scan. At the index lookup step, the query performs slightly better by decreasing time by 0.001 s. There is not much difference in the cost comparing the default index and the index on movieName attribute. Overall, the query neither shows a substantial increase nor decrease in performance. Digging deeper and trying to analyze why creating an index on Query 2 has not changed the performance, we notice that for indexing on movieName, there is not much difference here at the aggregation step. As the aggregation operation is the main point of optimization in the query, the overall performance is not affected much.

## Indexing on show(showId)

CREATE INDEX show\_idx ON show(showId)

```
| -> Sort: `sum(b.bookingId)` DESC (actual time=11.571..11.578 rows=89 loops=1)
| -> Table scan on <temporary> (actual time=0.003..0.052 rows=89 loops=1)
|   -> Aggregate using temporary table (actual time=11.470..11.525 rows=89 loops=1)
|     -> Nested loop inner join (cost=1989.93 rows=7261) (actual time=0.053..6.744 rows=7200 loops=1)
|       -> Nested loop inner join (cost=812.50 rows=1800) (actual time=0.043..1.399 rows=1800 loops=1)
|         -> Index scan on s using show_idx (cost=182.50 rows=1800) (actual time=0.027..0.430 rows=1800 loops=1)
|         -> Single-row index lookup on m using PRIMARY (movieId=s.movieId) (cost=0.25 rows=1) (actual time=0.000..0.000 rows=1 loops=1800)
|       -> Index lookup on b using showId (showId=s.showId) (cost=0.25 rows=4) (actual time=0.002..0.003 rows=4 loops=1800)
|
```

After creating an index on showId attribute, we note the following:

The time for accessing the first record increases with this attribute. But going down in the tree, we see slight increase in performance in the aggregation step, then further increase at the first nested inner join and the second nested inner join step. The index scan step also shows better performance compared to default indexing, but index lookup ends up showing the same performance. As before, there is no difference observed in the cost. Overall, indexing with showId shows a slight increase in performance. Digging deeper and trying to analyze why creating an index on Query 2 has not changed the performance much, we notice that for indexing on showId, there is not much difference here at the aggregation step rather a slight increase in performance. As the aggregation operation is the main point of optimization in the query, the overall performance is not affected much with only a slight increase.

## Indexing on booking(bookingId)

CREATE INDEX booking\_idx ON booking(bookingId)

```
| -> Sort: `sum(b.bookingId)` DESC (actual time=12.658..12.664 rows=89 loops=1)
|   -> Table scan on <temporary> (actual time=0.003..0.032 rows=89 loops=1)
|     -> Aggregate using temporary table (actual time=12.567..12.601 rows=89 loops=1)
|       -> Nested loop inner join (cost=1989.93 rows=7261) (actual time=0.140..7.121 rows=7200 loops=1)
|         -> Nested loop inner join (cost=812.50 rows=1800) (actual time=0.128..1.562 rows=1800 loops=1)
|           -> Index scan on s using adminId (cost=182.50 rows=1800) (actual time=0.051..0.477 rows=1800 loops=1)
|             -> Single-row index lookup on m using PRIMARY (movieId=s.movieId) (cost=0.25 rows=1) (actual time=0.000..0.000 rows=1 loops=1800)
|               -> Index lookup on b using showId (showId=s.showId) (cost=0.25 rows=4) (actual time=0.002..0.003 rows=4 loops=1800)
```

After creating an index on bookingId attribute, we note the following:

The time for accessing the first record increases with this attribute by 0.7s. In the aggregation step, we see a slight increase in the time of 0.001 s. Going further down in the tree, we see performance decreases further at the first nested inner join and the second nested inner join step. The index scan step also shows a worse performance compared to default indexing of 0.01s but index lookup ends up showing the same performance. As before, there is no difference observed in the cost. Overall, the query does not perform well with the bookingId attribute. Digging deeper and trying to analyze why creating an index on Query 2 has decreased the performance, we notice that for indexing on bookingId, there is a decrease in performance. As the aggregation operation is the main point of optimization in the query, the overall performance has degraded.

## APPENDIX

```
!pip install unique-names-generator
from unique_names_generator import get_random_name
from unique_names_generator.data import ADJECTIVES, NAMES

!pip install names
import names

import numpy as np
import pandas as pd
import hashlib

# getting 2500 random users
users = {}
emails = ['gmail.com', 'illinois.edu', 'hotmail.com', 'rocketmail.com', 'yahoo.com', 'outlook.com']
for _ in range(2500):
    try:
        uname = get_random_name(combo=[ADJECTIVES, NAMES], separator="-", style="lowercase")
        adj, name = uname.split('-')
        provider = np.random.choice(emails, 1, p=[0.3, 0.3, 0.1, 0.1, 0.1, 0.1])
        surname = names.get_last_name()
        email = name + '.' + surname + '@' + provider[0]
        password = name[:-2] + str(np.random.randint(1950, 2022))
        hashed_password = hashlib.md5(password.encode())
        users[uname] = [email, password, hashed_password.hexdigest()]
    except Exception as Err:
        continue
users_df = pd.DataFrame([{"UserName": name, "Email": email, "Password": password, "Hashed Password": md5} for name, (email, password, md5) in users.items()])
users_df.to_excel('sample_data/users.xlsx')

# Generating admin CSV
aname = ['rishisg2', 'amb20', 'vchheda2', 'pma7']
aemail = ['rishisg2@illinois.edu', 'amb20@illinois.edu', 'vchedda2@illinois.edu', 'pma7@illinois.edu']
admins = {}
for i in range(len(aname)):
    name = aname[i]
    email = aemail[i]
    password = name[:-1]
    hashed_password = hashlib.md5(password.encode())
    admins[name] = [email, password, hashed_password.hexdigest()]

admins_df = pd.DataFrame([{"UserName": name, "Email": email, "Password": password, "Hashed Password": md5} for name, (email, password, md5) in admins.items()])
admins_df.to_excel('sample_data/admin.xlsx')
```

```

# creating movies

datar = pd.read_csv("sample_data/rotten_tomatoes_movies.csv")
movies = pd.DataFrame(datar)
movies = movies.drop([
    'Unnamed: 4', 'Unnamed: 5', 'Unnamed: 6',
    'Unnamed: 7', 'Unnamed: 8'], axis=1)
movies.rename(columns={
    'Unnamed: 3': 'AdminId',
    'rotten_tomatoes_link': 'MovieId',
    'movie_title': 'MovieName',
    'movie_info': 'Description',
    'streaming_release_date': 'ReleaseDate'}, inplace=True)

a = [1,2,3,4]
adminid = []
for i in range(movies.shape[0]):
    adminid.append(np.random.choice(a, 1)[0])

movies['AdminId'] = adminid
movies = movies[movies['ReleaseDate'].notna()]

dates = list(movies['ReleaseDate'])
dates_modified = []
for d in dates:
    d = d.split('/')
    d = d[-1] + '-' + d[0] + '-' + d[1]
    dates_modified.append(d)
movies['ReleaseDate'] = dates_modified

print(movies.shape[0])
movies.to_excel('sample_data/movies.xlsx')
movies.head(10)

# making shows
# for three months
show_dt = []
times = [i for i in range(1,7,2)]
screens = [i for i in range(1, 6)]
id = 0
done = {}
price = [10, 12, 14, 8, 9, 5, 13, 16, 7]
admin = [1,2,3,4]
movie_ids = movies['MovieId']

for i in range(1, 31):
    a = np.random.choice(admin, 1)

    for j in times:
        p = np.random.choice(price, 1)
        while True:
            mid = np.random.choice(movie_ids, 1)[0]

```



```

        if mid not in done:
            done[mid] = 1
            break

for k in screens:
    if i <= 9:
        d1 = '2022-08-0' + str(i) + " " + "0" + str(j) + ":00:00"
        d2 = '2022-09-0' + str(i) + " " + "0" + str(j) + ":00:00"
        d3 = '2022-10-0' + str(i) + " " + "0" + str(j) + ":00:00"
        d4 = '2022-11-0' + str(i) + " " + "0" + str(j) + ":00:00"
    else:
        d1 = '2022-08-' + str(i) + " " + "0" + str(j) + ":00:00"
        d2 = '2022-09-' + str(i) + " " + "0" + str(j) + ":00:00"
        d3 = '2022-10-' + str(i) + " " + "0" + str(j) + ":00:00"
        d4 = '2022-11-' + str(i) + " " + "0" + str(j) + ":00:00"

    if i <= 30:
        id += 1; show_dt.append([id, mid, d1, k, p, a])
        id += 1; show_dt.append([id, mid, d2, k, p, a])
        id += 1; show_dt.append([id, mid, d3, k, p, a])
        id += 1; show_dt.append([id, mid, d4, k, p, a])
    else:
        id += 1; show_dt.append([id, mid, d1, k, p, a])
        id += 1; show_dt.append([id, mid, d3, k, p, a])

shows_df = pd.DataFrame([{"ShowId": x[0], "MovieId": x[1], "ShowTiming": x
[2], "Screen": x[3], "Price":x[4][0], "AdminId": x[5][0]} for x in show_dt
])
shows_df.to_excel('sample_data/shows.xlsx')

from collections import defaultdict
import numpy as np

users = pd.read_csv('sample_data/users.csv')
shows = pd.DataFrame(pd.read_excel('sample_data/shows.xlsx'))
showid = shows[['ShowId', 'ShowTiming']]

u = users['mid-sophi']
s = [i for i in range(5)]
b = [i for i in range(1, 5)]
c = [i for i in range(1, 6)]

booking = []
seats = []
bid = 0

for i in range(showid.shape[0]):
    sb = defaultdict(list)
    sid = 0
    # create 50 seats for every show
    sts = [[0 for _ in range(5)] for _ in range(5)]

    # decide number of rows and which rows will have bookings

```



```

n = np.random.choice(c, 1)
nb = np.random.choice(b, 4, replace=False)

# for row
for r in nb:
    # pick a random user for creating a booking
    usr = np.random.choice(u, 1)

    # randomly pick number of seats booked by user
    t = np.random.choice(s, 1)[0]

    # register the booking
    bid += 1
    tm1, tm2 = showid.loc[i][1].split(' ')
    tm2 = tm2.split(':')
    tm2[0] = str(int(tm2[0]) - 1)
    if int(tm2[0]) < 0: tm2[0] = 0
    if len(str(tm2[0])) < 2: tm2[0] = '0'+str(tm2[0])
    # tm1 = tm1.split('-')
    # tm1 = '/'.join(tm1)
    tmf = tm1 + ' ' + ':'.join(tm2)
    booking.append([bid, usr, tmf, showid.loc[i][0]])
    sb[showid.loc[i][0]].append(bid)

    # update the booked shows
    for k in range(t):
        sts[r][k] = 1

# register seats (25 seats per show)
sid += 1
for r in range(len(sts)):
    for c in range(len(sts[0])):
        if sts[r][c] == 1:
            seats.append([sid, None, showid.loc[i][0], sts[r][c]])
        else:
            nbid = np.random.choice(list(sb.values())[0], 1)
            seats.append([sid, nbid, showid.loc[i][0], sts[r][c]])
            sid += 1

booking_df = pd.DataFrame(booking)
booking_df.to_csv('sample_data/bookings.csv')
seats_df = pd.DataFrame(seats)
seats_df.to_csv('sample_data/seats.csv')

```