UNIVERSITY OF DAYTON

DATABASE DESIGN

of

THEATER MANAGEMENT SYSTEM

(FINAL REPORT)

BY

TANISHA BHOLA

Under the guidance of

Dr. Eman Elrifaei

DATABASE MANAGEMENT SYSTEMS

CPS 542

Department of Computer Science

Contents

1.1 INTRODUCTION

In today's entertainment landscape, theaters serve as pivotal hubs for cultural experiences, gathering individuals from diverse backgrounds to revel in the enchantment of cinema. However, managing a multi-screen theater complex poses myriad challenges, including scheduling movie screenings, optimizing seat allocations, handling ticket bookings, and collating customer feedback. The Theater Management System addresses these challenges by providing a comprehensive solution that streamlines operations, enhances customer experiences, and empowers theater administrators with powerful tools for efficient managementThe Theater Management System is meticulously crafted to cater to the demands of modern theaters, which often boast multiple screens showcasing a diverse array of movies across different genres and formats. By centralizing key functionalities such as scheduling, ticketing, customer management, and feedback collection, the system offers a unified platform that simplifies intricate tasks and fosters a seamless movie-going experience for patrons.

The primary objective of the Theater Management System is to elevate the overall theater experience for both customers and administrators. For customers, the system offers an intuitive interface where they can effortlessly peruse movie listings, verify showtimes, book tickets, and provide feedback on their experiences. By facilitating convenient online ticket booking and personalized services, the system aims to augment customer satisfaction and foster loyalty.

For theater administrators, the system provides robust tools for managing theaters, screens, schedules, movies, tickets, customers, and feedback. Administrators can efficiently allocate resources, optimize scheduling, track performance metrics, and analyze customer feedback to make informed decisions for enhancing operations and service quality.

In summary, the Theater Management System represents a significant advancement in theater operations, harnessing technology to streamline processes, enrich customer experiences, and propel business growth. By embracing innovation and adapting to the evolving needs of the entertainment industry, theaters can remain competitive and continue enchanting audiences with unforgettable movie-going experiences.

1.2 ENTITIES AND ATTRIBUTES WITH KEYS:

Theatre: Centralizes information about theaters, encompassing crucial details like name, location, capacity, and contact number, enabling streamlined management and operation of theater venues while ensuring optimal customer service and engagement.

Attributes:

- Theatre_ID: Unique identifier for each theater.

- Name: Name of the theater.
- Location: Location of the theater.
- Capacity: Seating capacity of the theater.
- Contact_Number: Contact number of the theater.

Screen: Tracks essential data regarding screens within theaters, encompassing attributes such as screen type, seating capacity, and their association with specific theaters, facilitating efficient organization and utilization of screening resources, thus enhancing the overall movie-watching experience for patrons.

Attributes:

- Screen_ID: Unique identifier for each screen.
- Theatre_ID [FK]: Foreign key linking the screen to its respective theater.
- Screen_Number: Identifier for the screen within the theater.
- Screen_Type: Type of screen (e.g., 2D, 3D).
- Seating_Capacity: Maximum number of seats available in the screen.

Schedule: Orchestrates movie screening schedules for each screen, encapsulating vital information such as schedule ID, associated theater and screen, movie ID, and showtime, ensuring seamless coordination of movie screenings and effective allocation of resources to maximize audience satisfaction and revenue generation.

Attributes:

- Schedule_ID: Unique identifier for each schedule entry.
- Theatre_ID [FK]: Foreign key linking the schedule to its respective theater.
- Screen_ID [FK]: Foreign key linking the schedule to its respective screen.
- Movie_ID [FK]: Foreign key linking the schedule to the movie being screened.
- Show_Time: Date and time of the movie screening.

Movies: Catalogs comprehensive information about movies available for screening, including title, genre, director, and release date, providing a diverse selection of cinematic experiences to cater to the varied preferences of audiences and enrich their overall entertainment journey.

Attributes:

- Movie_ID: Unique identifier for each movie.
- Title: Title of the movie.
- Genre: Genre of the movie.
- Director: Director of the movie.
- Release_Date: Release date of the movie.

Tickets: Manages the booking and sale of tickets for movie screenings, capturing crucial details such as ticket ID, associated schedule and movie, customer ID, seat number, price, and booking status, facilitating seamless ticketing processes, and ensuring an efficient and convenient booking experience for customers.

Attributes:

- Ticket_ID: Unique identifier for each ticket.
- Schedule_ID [FK]: Foreign key linking the ticket to its respective schedule.
- Movie_ID [FK]: Foreign key linking the ticket to the movie being screened.
- Customer_ID [FK]: Foreign key linking the ticket to the customer who booked it.
- Seat_Number: Identifier for the seat booked by the customer.
- Price: Price of the ticket.
- Status: Booking status of the ticket (e.g., booked, canceled).

Customer: Maintains detailed records of customers, encompassing attributes such as customer ID, name, contact information, and membership status, enabling personalized services, effective communication, and targeted marketing strategies to enhance customer satisfaction and loyalty.

Attributes:

- Customer_ID: Unique identifier for each customer.
- Customer_Name: Name of the customer.
- Customer_Email: Email address of the customer.
- Customer_Phone_Number: Phone number of the customer.
- Membership_Status: Membership status of the customer.

Feedback: Gathers valuable feedback from customers about their movie-watching experiences, including ratings and comments, fostering a culture of continuous improvement and enabling theater management to make data-driven decisions to enhance service quality and customer satisfaction.

Attributes:

- Customer_ID [FK]: Foreign key linking the feedback to the customer who provided it.
- Movie_ID [FK]: Foreign key linking the feedback to the movie it pertains to.
- Rating: Rating given by the customer for the movie.
- Comments: Additional comments or feedback provided by the customer.

1.3 DESIGN ASSUMPTIONS

Theatre to Screen:
Each theatre must have at least one screen to facilitate movie screenings. This ensures that the theatre can effectively host movie showings and maximize revenue generation. The screens are primarily dedicated to movie screenings but may also be used for other purposes or special events.

Screen to Schedule:
Each screen must have at least one schedule for movie screenings to effectively manage movie showings and allocate resources. While screens are primarily associated with movie screenings, there may be additional schedules for events or maintenance that do not involve movie screenings.

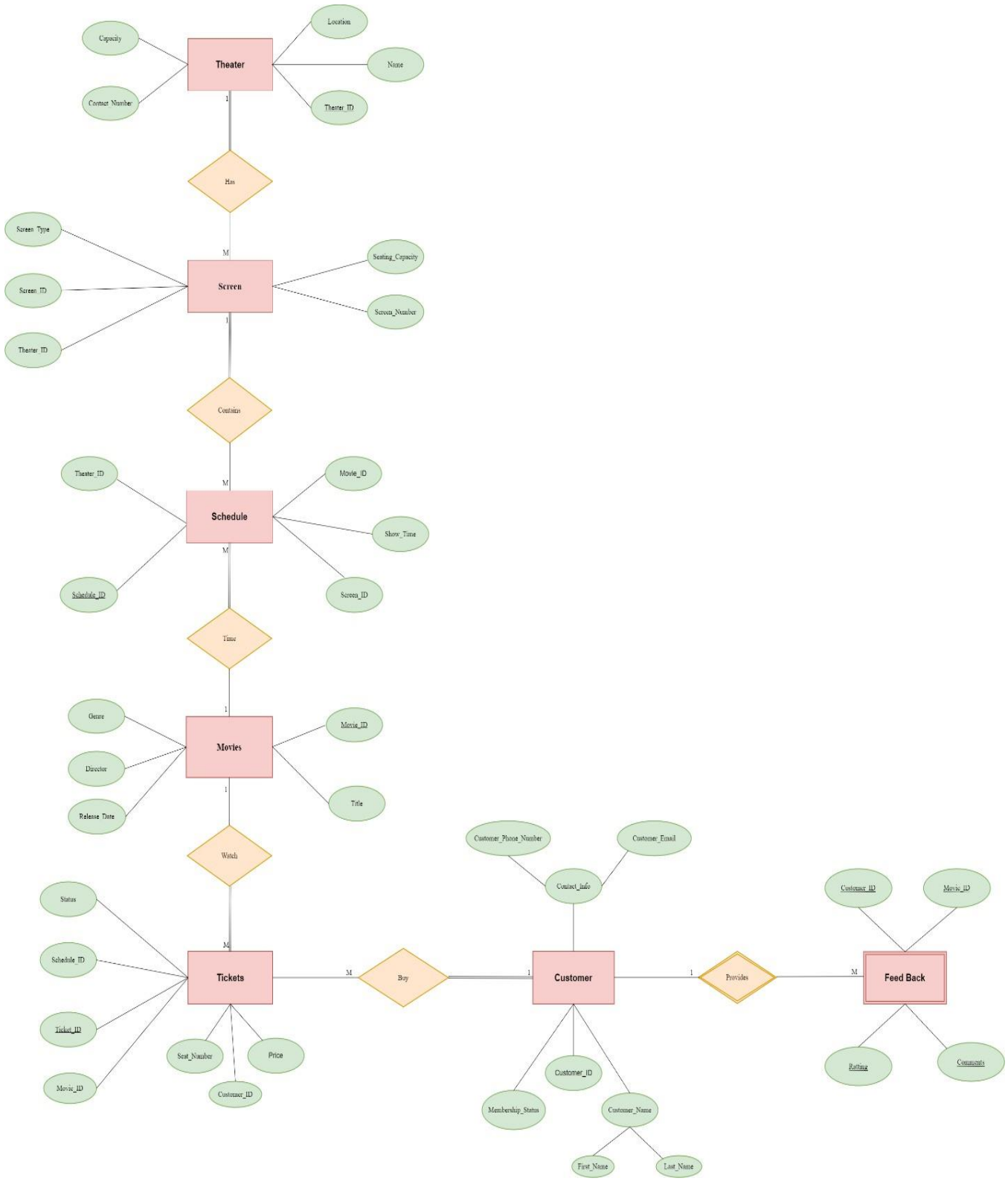
Schedule to Movies:
Every movie screening schedule must be associated with a specific movie to inform audiences and ensure accurate programming. Not every movie necessarily has a scheduled screening, as programming decisions may vary based on factors such as audience demand or special shows.

Movies to Tickets:
Tickets are associated with specific movies to facilitate audience access to movie screenings and manage seating arrangements. While tickets are primarily associated with movie screenings, there may be tickets for other events or services offered by the theatre, such as special screenings or live performances.

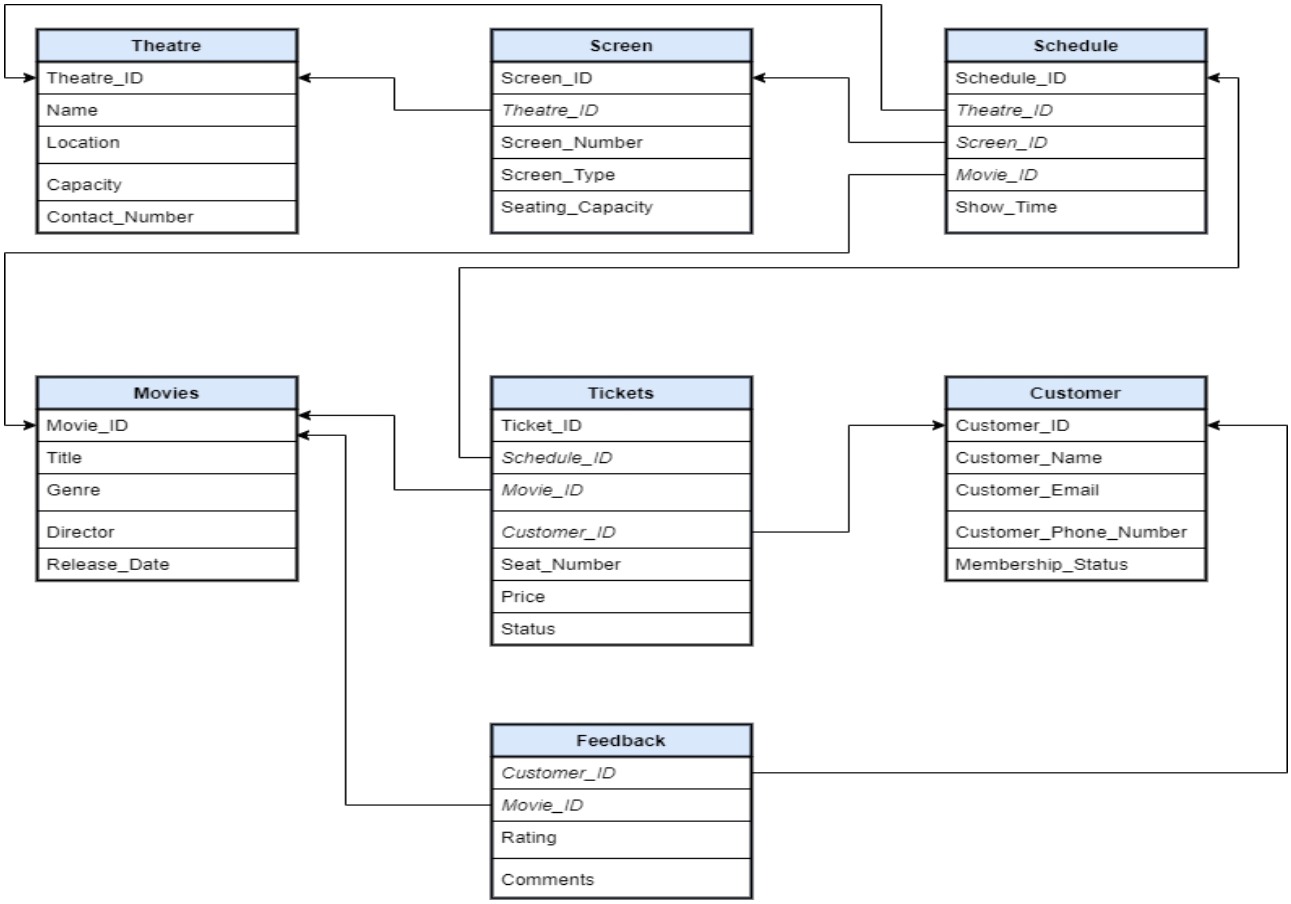
Tickets to Customer:
Each ticket sold by the theatre must be associated with a specific customer to track sales and facilitate customer service. Also, not every customer necessarily purchases tickets, as some may visit the theatre for other services or events.

Customer to Feedback:
Customer feedback is valuable for improving service quality and enhancing customer satisfaction. While customers are encouraged to provide feedback on their experience with the theatre and movie, not every customer necessarily provides feedback. Additionally, feedback entries may not always be associated with specific customers, as there may be anonymous feedback or feedback provided by other stakeholders.

1.4 ER DIAGRAM


Theater
Capacity
Contact Number
Location
Name
Theater_ID
1
Has
M
Screen
Screen_Type
Screen_ID
Theater_ID
Seating_Capacity
Screen_Number
1
Contains
M
Schedule
Theater_ID
Schedule_ID
Movie_ID
Show_Time
Screen_ID
M
Time
1
Movies
Genre
Director
Release_Date
Movie_ID
Title
1
Watch
M
Tickets
Status
Schedule_ID
Ticket_ID
Movie_ID
Seat_Number
Price
Customer_ID
M
Buy
1
Customer
Customer_Phone_Number
Customer_Email
Contact_Info
Customer_ID
Membership_Status
Customer_Name
First_Name
Last_Name
1
Provides
M
Feed Back
Customer_ID
Movie_ID
Rating
Comments

1.5 RELATIONAL SCHEMA



1.6 RELATIONAL SCHEMA MAPPING

- Theatre (Theatre_ID, Name, Location, Capacity, Contact_Number)
- Has(Theatre_ID, Screen_ID)
- Screen (Screen_ID, Theatre_ID [FK], Screen_Number, Screen_Type, Seating_Capacity)
    - Foreign Key: Theatre_ID references Theatre(Theatre_ID)
- Contains(Screen_ID, Schedule_ID)
- Schedule (Schedule_ID, Screen_ID [FK], Movie_ID [FK], Show_Time)
    - Foreign Keys:
        - Screen_ID references Screen(Screen_ID)
        - Movie_ID references Movies(Movie_ID)
- Movies (Movie_ID, Title, Genre, Director, Release_Date)
- Time(Schedule_ID, Movie_ID)

- Tickets (Ticket_ID, Schedule_ID [FK], Customer_ID [FK], Seat_Number, Price, Status)
    - Foreign Keys:
        - Schedule_ID references Schedule(Schedule_ID)
        - Customer_ID references Customer(Customer_ID)
- Watch(Movie_ID, Ticket_ID)
- Customer (Customer_ID, Customer_Name, Customer_Email, Customer_Phone_Number, Membership_Status)
- Buy(Ticket_ID, Customer_ID)
- Feedback (Customer_ID [FK], Movie_ID [FK], Rating, Comments)
    - Foreign Keys:
        - Customer_ID references Customer(Customer_ID)
        - Movie_ID references Movies(Movie_ID)
- Provides(Customer_ID, Rating, Comments)

1.7 ENTITIES AND ATTRIBUTES

- Theatre :

Theatre_ID, Name, Location, Capacity, Contact_Number

- Screen :
  Screen_ID, Screen_Number, Screen_Type, Seating_Capacity

- Schedule :
  Schedule_ID, Show_Time

- Movies :
  Movie_ID, Title, Genre, Director, Release_Date

- Tickets :
  Ticket_ID, Seat_Number, Price, Status

- Customer :
  Customer_ID, Customer_Name, Customer_Email, Customer_Phone_Number, Membership_Status

- Feedback :
  Rating, Comments

1.8 RELATIONS

Theater has Screens(One to Many)
Screen contains Schedule(One to Many)
Schedule time Movie(Many to One)
Movie watch Tickets(One to Many)
Tickets buy Customer(Many to One)
Customer provides Feedback(Many to One)

1.9 NORMALIZATION OF THE RELATIONAL SCHEMA:

Normalization is done to minimize the data redundancy in the database model of the Theater Management System.

For normalization, you need to check if the system is in 1NF,2NF, and 3NF.

1NF: A relation is in 1NF if it contains an atomic value. For each multi-valued attribute, create a new table, in which you place the key to the original table and the multi-valued attribute. Keep the original table, with its key.

2NF: A relation will be in 2NF if it is in 1NF, and all non-key attributes are fully functional and dependent on the primary key.

3NF: As it should be in 2NF and every non-prime attribute of the relation is non-transitively dependent on every key in the relation also whenever a non-trivial functional dependency X→A exists, then either X is a super key or A is a member of some candidate key.

BCNF (Boyce Codd's normal form): Stronger than 3NF, it should be in 3NF and whenever a non-trivial functional dependency X→A exists, X should be a super key.

1. **Theatre (Theatre_ID, Name, Location, Capacity, Contact_Number)**

- Candidate Key: Theatre_ID
- 1NF: Already in 1NF as all attributes are atomic.
- 2NF: No partial dependencies exist as all non-key attributes are fully functionally dependent on the primary key.
- 3NF: No transitive dependencies exist; each non-key attribute depends only on the primary key.
- Conclusion: Theatre is in 3NF.
- There are no non-trivial functional dependencies other than trivial ones (Theatre_ID -> Theatre_ID).

- The relation is already in BCNF.

2. Screen (Screen_ID, Theatre_ID [FK], Screen_Number, Screen_Type, Seating_Capacity)

- Candidate Key: Screen_ID
- 1NF: Already in 1NF as all attributes are atomic.
- 2NF: No partial dependencies exist as all non-key attributes are fully functionally dependent on the primary key.
- 3NF: No transitive dependencies exist; each non-key attribute depends only on the primary key.
- Conclusion: Screen is in 3NF.
- Functional Dependency: Screen_ID -> Theatre_ID, Screen_Number, Screen_Type, Seating_Capacity
- Theatre_ID is not a superkey, as Screen_ID uniquely identifies all other attributes.
  - To achieve BCNF, we need to decompose the relation.
    So, the only relation that needs to be decomposed to achieve BCNF is Screen. We can decompose it as follows:
    Screen_1 (Screen_ID, Theatre_ID [FK])
  - Candidate Key: {Screen_ID}
    Screen_2 (Screen_ID [FK], Screen_Number, Screen_Type, Seating_Capacity)
  - Candidate Key: {Screen_ID}

  Now, all relations are in BCNF.

3. Schedule (Schedule_ID, Screen_ID [FK], Movie_ID [FK], Show_Time)

- Candidate Key: Schedule_ID
- 1NF: Already in 1NF as all attributes are atomic.
- 2NF: No partial dependencies exist as all non-key attributes are fully functionally dependent on the primary key.
- 3NF: No transitive dependencies exist; each non-key attribute depends only on the primary key.
- Conclusion: Schedule is in 3NF.
- There are no non-trivial functional dependencies other than trivial ones.
- The relation is already in BCNF.

4. Movies (Movie_ID, Title, Genre, Director, Release_Date)

- Candidate Key: Movie_ID
- 1NF: Already in 1NF as all attributes are atomic.
- 2NF: No partial dependencies exist as all non-key attributes are fully functionally dependent on the primary key.
- 3NF: No transitive dependencies exist; each non-key attribute depends only on the primary key.

- Conclusion: Movies is in 3NF.
- There are no non-trivial functional dependencies other than trivial ones.
- The relation is already in BCNF.

5. Tickets (Ticket_ID, Schedule_ID [FK], Customer_ID [FK], Seat_Number, Price, Status)

- Candidate Key: Ticket_ID
- 1NF: Already in 1NF as all attributes are atomic.
- 2NF: No partial dependencies exist as all non-key attributes are fully functionally dependent on the primary key.
- 3NF: No transitive dependencies exist; each non-key attribute depends only on the primary key.
- Conclusion: Tickets is in 3NF.
- There are no non-trivial functional dependencies other than trivial ones.
- The relation is already in BCNF.

6. Customer (Customer_ID, Customer_Name, Customer_Email, Customer_Phone_Number, Membership_Status)

- Attributes: Customer_ID (PK), Customer_Name, Customer_Email, Customer_Phone_Number, Membership_Status
- Candidate Key: Customer_ID
- 1NF: Already in 1NF as all attributes are atomic.
- 2NF: No partial dependencies exist as all non-key attributes are fully functionally dependent on the primary key.
- 3NF: No transitive dependencies exist; each non-key attribute depends only on the primary key.
- Conclusion: Customer is in 3NF.
- There are no non-trivial functional dependencies other than trivial ones.
- The relation is already in BCNF.

7. Feedback (Customer_ID [FK], Movie_ID [FK], Rating, Comments)

- Attributes: Customer_ID (FK), Movie_ID (FK), Rating, Comments
- Candidate Key: (Customer_ID, Movie_ID)
- 1NF: Already in 1NF as all attributes are atomic.
- 2NF: No partial dependencies exist as all non-key attributes are fully functionally dependent on the primary key.
- 3NF: No transitive dependencies exist; each non-key attribute depends only on the primary key.
- Conclusion: Feedback is in 3NF.
- There are no non-trivial functional dependencies other than trivial ones.

- The relation is already in BCNF.

1.10 PHYSICAL MODEL

THEATER TABLE:

```
CREATE TABLE  "THEATRE"
  (       "THEATRE_ID" NUMBER(10,0),
          "NAME" NCLOB,
          "LOCATION" NCLOB,
          "CAPACITY" NUMBER(3,0),
          "CONTACT_NUMBER" NUMBER(10,0),
           CONSTRAINT "THEATRE_ID" PRIMARY KEY ("THEATRE_ID") ENABLE
  ) ;
```

TICKETS TABLE:

```
CREATE TABLE  "TICKETS"
  (          "TICKET_ID" NUMBER(10,0) NOT NULL ENABLE,
             "SEAT_NUMBER" VARCHAR2(10) NOT NULL ENABLE,
             "PRICE" VARCHAR2(10) NOT NULL ENABLE,
             "STATUS" VARCHAR2(12) NOT NULL ENABLE,
             "FK_SCHEDULE_ID" NUMBER(4,0) NOT NULL ENABLE,
             "FK_MOVIE_ID" NUMBER(4,0) NOT NULL ENABLE,
             "FK_CUSTOMER_ID" NUMBER(4,0) NOT NULL ENABLE,
             PRIMARY KEY ("TICKET_ID") ENABLE
  ) ;ALTER TABLE  "TICKETS" ADD FOREIGN KEY ("FK_SCHEDULE_ID")
              REFERENCES  "SCHEDULE" ("SCHEDULE_ID") ENABLE;ALTER TABLE
"TICKETS" ADD FOREIGN KEY ("FK_MOVIE_ID")
              REFERENCES  "MOVIE" ("MOVIE_ID") ENABLE;ALTER TABLE  "TICKETS"
ADD FOREIGN KEY ("FK_CUSTOMER_ID")
              REFERENCES  "CUSTOMER" ("CUSTOMER_ID") ENABLE;
```

SCREEN TABLE:

```
CREATE TABLE  "SCREEN"
  (          "SCREEN_ID" NUMBER(10,0) NOT NULL ENABLE,
             "THEATRE_ID" NUMBER(10,0) NOT NULL ENABLE,
             "SCREEN_NUMBER" NUMBER(3,0) NOT NULL ENABLE,
             "SCREEN_TYPE" NCLOB NOT NULL ENABLE,
             "SEATING_CAPACITY" NUMBER(3,0) NOT NULL ENABLE,
              CONSTRAINT "SCREEN_ID" PRIMARY KEY ("SCREEN_ID") ENABLE
```

```
  ) ;ALTER TABLE  "SCREEN" ADD CONSTRAINT "THEATER_ID" FOREIGN KEY
("THEATRE_ID")
            REFERENCES  "THEATRE" ("THEATRE_ID") ENABLE;
```

SCHEDULE TABLE:

```
CREATE TABLE  "SCHEDULE"
  (         "SCHEDULE_ID" NUMBER(10,0) NOT NULL ENABLE,
            "THEATRE_ID" NUMBER(10,0) NOT NULL ENABLE,
            "SCREEN_ID" NUMBER(3,0) NOT NULL ENABLE,
            "MOVIE_ID" NUMBER(10,0) NOT NULL ENABLE,
            "SHOW_TIME" VARCHAR2(30) NOT NULL ENABLE,
             CONSTRAINT "PK_SCHEDULE_ID" PRIMARY KEY ("SCHEDULE_ID")
ENABLE
  ) ;ALTER TABLE  "SCHEDULE" ADD CONSTRAINT "FK_SCREEN_ID" FOREIGN KEY
("SCREEN_ID")
            REFERENCES  "SCREEN" ("SCREEN_ID") ENABLE;ALTER TABLE
"SCHEDULE" ADD CONSTRAINT "FK_THEATRE_ID" FOREIGN KEY ("THEATRE_ID")
            REFERENCES  "THEATRE" ("THEATRE_ID") ENABLE;ALTER TABLE
"SCHEDULE" ADD CONSTRAINT "MOVIE_ID" FOREIGN KEY ("MOVIE_ID")
            REFERENCES  "MOVIE" ("MOVIE_ID") ENABLE;
```

MOVIE TABLE:

```
CREATE TABLE  "MOVIE"
  (         "MOVIE_ID" NUMBER(10,0) NOT NULL ENABLE,
            "TITLE" VARCHAR2(100) NOT NULL ENABLE,
            "GENRE" VARCHAR2(50) NOT NULL ENABLE,
            "DIRECTOR" VARCHAR2(100) NOT NULL ENABLE,
            "RELEASE_DATE" DATE NOT NULL ENABLE,
             CONSTRAINT "PK_MOVIE_ID" PRIMARY KEY ("MOVIE_ID") ENABLE
  ) ;
```

FEEDBACK TABLE:

```
CREATE TABLE  "FEEDBACK"
  (         "RATING" NUMBER(3,0) NOT NULL ENABLE,
            "COMMENTS" VARCHAR2(300) NOT NULL ENABLE,
            "FK_CUSTOMER_ID" NUMBER,
            "FK_MOVIE_ID" NUMBER
  ) ;ALTER TABLE  "FEEDBACK" ADD CONSTRAINT "FK_CUSTOMER_ID" FOREIGN
KEY ("FK_CUSTOMER_ID")
            REFERENCES  "CUSTOMER" ("CUSTOMER_ID") ENABLE;ALTER TABLE
"FEEDBACK" ADD CONSTRAINT "FK_MOVIE_ID" FOREIGN KEY ("FK_MOVIE_ID")
            REFERENCES  "MOVIE" ("MOVIE_ID") ENABLE;
```

CUSTOMER TABLE:

```
CREATE TABLE  "CUSTOMER"
  (         "CUSTOMER_ID" NUMBER(10,0) NOT NULL ENABLE,
            "CUSTOMER_NAME" VARCHAR2(50) NOT NULL ENABLE,
            "CUSTOMER_EMAIL" VARCHAR2(100) NOT NULL ENABLE,
            "CUSTOMER_PHONE_NUMBER" VARCHAR2(15) NOT NULL ENABLE,
            "MEMBERSHIP_STATUS" CHAR(10) NOT NULL ENABLE,
            CONSTRAINT "PK_CUSTOMER_ID" PRIMARY KEY ("CUSTOMER_ID")
ENABLE
  ) ;
```

THEATER TABLE DATA:

THEATRE

Table Data Indexes Model Constraints Grants Statistics UI Defaults Triggers Dependencies SQL

Query Count Rows Insert Row

EDIT	THEATRE_ID	NAME	LOCATION	CAPACITY	CONTACT_NUMBER
	1	Starlight Cinema	123 Main Street Cityville	500	8463465467
	2	Sunset Theater	456 Elm Street Townsville	300	6476543763
	3	Majestic Movies	789 Oak Avenue Villagetown	700	7346876487
	4	Galaxy Cinemas	321 Pine Road Suburbia	400	9847897598
	5	Royal Theatre	654 Cedar Lane Hamletville	600	7834687456
	6	Dreamland Cinema	987 Maple Drive Countryside	350	8734873467
	7	Silver Screen Theater	210 Walnut Lane Metropolis	550	8793248364
	8	Paradise Picture House	543 Cherry Street Seaside	450	8475874589
	9	Oasis Odeon	876 Birch Boulevard Lakeside	650	9438759845
	10	Magic Movie Palace	109 Pineapple Avenue Wonderland	550	8734873487
	11	Grand Theater	321 Peach Place Riverside	700	3847987498
	12	Crystal Cinema	654 Lemon Lane Mountainview	400	9384899384
	13	Sunshine Cinematheque	987 Grape Grove Hilltop	300	8937983783
	14	Golden Gate Theatre	210 Orange Orchard Cityscape	800	3489034898
	15	Rainbow Movies	543 Strawberry Street Countryside	450	9899328948

row(s) 1 - 15 of 20

SCREEN TABLE DATA:

SCREEN

Table Data Indexes Model Constraints Grants Statistics UI Defaults Triggers Dependencies S

Query Count Rows Insert Row

EDIT	SCREEN_ID	THEATRE_ID	SCREEN_NUMBER	SCREEN_TYPE	SEATING_CAPACITY
	101	1	1	3D	100
	102	1	2	2D	120
	103	1	3	3D	80
	104	2	1	2D	90
	105	2	2	3D	110
	106	3	1	2D	150
	107	3	2	3D	100
	108	4	1	3D	70
	109	4	2	2D	80
	110	5	1	2D	120
	111	5	2	3D	90
	112	6	1	3D	110
	113	6	2	2D	100
	114	7	1	2D	80
	115	7	2	3D	70

SCHEDULE TABLE DATA:

SCHEDULE

Table Data Indexes Model Constraints Grants Statistics UI Defaults Triggers Dependen

Query Count Rows Insert Row

EDIT	SCHEDULE_ID	THEATRE_ID	SCREEN_ID	MOVIE_ID	SHOW_TIME
	201	1	101	301	16-04-2024 18:00
	202	1	102	302	16-04-2024 20:00
	203	2	103	303	16-04-2024 19:00
	204	2	104	304	16-04-2024 17:00
	205	3	105	305	16-04-2024 21:00
	206	3	106	306	16-04-2024 18:30
	207	4	107	307	16-04-2024 20:30
	208	4	108	308	16-04-2024 16:00
	209	5	109	309	16-04-2024 22:00
	210	5	110	310	16-04-2024 17:30
	211	6	111	311	16-04-2024 19:30
	212	6	112	312	16-04-2024 20:15
	213	7	113	313	16-04-2024 21:45
	214	7	114	314	16-04-2024 16:45
	215	8	115	315	16-04-2024 18:45

row(s) 1 - 15 of 20

MOVIE TABLE DATA:

MOVIE

Table Data Indexes Model Constraints Grants Statistics UI Defaults Triggers Dependencies SQL

Query Count Rows Insert Row

EDIT	MOVIE_ID	TITLE	GENRE	DIRECTOR	RELEASE_DATE
	301	Avengers: Endgame	Action, Adventure, Sci-Fi	Anthony Russo	04/26/2019
	302	Titanic	"Drama, Romance"	James Cameron	12/19/1997
	303	The Lion King	"Animation, Adventure, Drama"	Roger Allers	06/24/1994
	304	Jurassic Park	"Adventure, Sci-Fi"	Steven Spielberg	06/11/1993
	305	Avatar	"Action, Adventure, Fantasy"	James Cameron	12/18/2009
	306	The Dark Knight	"Action, Crime, Drama"	Christopher Nolan	07/18/2008
	307	Inception	"Action, Adventure, Sci-Fi"	Christopher Nolan	07/16/2010
	308	Forrest Gump	"Drama, Romance"	Robert Zemeckis	07/06/1994
	309	The Shawshank Redemption	Drama	Frank Darabont	10/14/1994
	310	The Matrix	"Action, Sci-Fi"	Lana Wachowski	03/31/1999
	311	Interstellar	"Adventure, Drama, Sci-Fi"	Christopher Nolan	11/07/2014
	312	The Lord of the Rings: The Return of the King	"Action, Adventure, Drama"	Peter Jackson	12/17/2003
	313	The Godfather	"Crime, Drama"	Francis Ford Coppola	03/24/1972
	314	The Godfather: Part II	"Crime, Drama"	Francis Ford Coppola	12/20/1974
	315	The Dark Knight Rises	"Action, Adventure, Thriller"	Christopher Nolan	07/20/2012

row(s) 1 - 15 of 20

TICKETS TABLE DATA:

TICKETS

Table Data Indexes Model Constraints Grants Statistics UI Defaults Triggers Dependencies SQL

Query Count Rows Insert Row

EDIT	TICKET_ID	SEAT_NUMBER	PRICE	STATUS	FK_SCHEDULE_ID	FK_MOVIE_ID	FK_CUSTOMER_ID
	401	A12	$12.50	Booked	201	301	501
	402	B8	$10.00	Booked	202	302	502
	403	C15	$8.50	Booked	203	303	503
	404	D3	$9.00	Booked	204	304	504
	405	E20	$11.00	Booked	205	305	505
	406	F10	$13.00	Booked	206	306	506
	407	G5	$12.00	Booked	207	307	507
	408	H14	$9.50	Booked	208	308	508
	409	I18	$10.50	Booked	209	309	509
	410	J7	$11.50	Booked	210	310	510
	411	K4	$13.50	Booked	211	311	511
	412	L11	$14.00	Booked	212	312	512
	413	M9	$8.00	Booked	213	313	513
	414	N17	$7.50	Booked	214	314	514
	415	O13	$10.00	Booked	215	315	515

row(s) 1 - 15 of 20

CUSTOMER TABLE DATA:

CUSTOMER

Table Data Indexes Model Constraints Grants Statistics UI Defaults Triggers Dependencies SQL

Query Count Rows Insert Row

EDIT	CUSTOMER_ID	CUSTOMER_NAME	CUSTOMER_EMAIL	CUSTOMER_PHONE_NUMBER	MEMBERSHIP_STATUS
	501	John Smith	john@example.com	555-5678	Premium
	502	Alice Johnson	alice@example.com	555-1234	Regular
	503	Michael Brown	michael@example.com	555-9012	Premium
	504	Sarah Davis	sarah@example.com	555-3456	Regular
	505	Emily Wilson	emily@example.com	555-6789	Premium
	506	David Martinez	david@example.com	555-0123	Regular
	507	Jennifer Taylor	jennifer@example.com	555-7890	Premium
	508	James Anderson	james@example.com	555-2345	Regular
	509	Jessica Thomas	jessica@example.com	555-8901	Premium
	510	Robert Lee	robert@example.com	555-2345	Regular
	511	Lisa Hall	lisa@example.com	555-6789	Premium
	512	Kevin Young	kevin@example.com	555-0123	Regular
	513	Amanda Clark	amanda@example.com	555-3456	Premium
	514	Daniel Rodriguez	daniel@example.com	555-6789	Regular
	515	Nicole Walker	nicole@example.com	555-9012	Premium

row(s) 1 - 15 of 20

FEEDBACK TABLE DATA:

FEEDBACK

Table Data Indexes Model Constraints Grants Statistics UI Defaults Triggers Dependencies SQL

Query Count Rows Insert Row

EDIT	RATING	COMMENTS	FK_CUSTOMER_ID	FK_MOVIE_ID
	5	Amazing movie! Loved every minute of it.	501	301
	4	Great storyline, but pacing was a bit slow.	502	302
	3	Decent movie, but not my favorite genre.	503	303
	5	Classic film! Still holds up after all these years.	504	304
	4	Visually stunning! A must-watch for fans of the genre.	505	305
	3	Interesting concept, but execution fell short.	506	306
	5	Absolutely loved it! Will definitely watch again.	507	307
	4	Good movie, but some scenes felt unnecessary.	508	308
	3	Average film, nothing too memorable.	509	309
	5	Fantastic performance by the lead actor!	510	310
	4	Mind-bending plot kept me engaged throughout.	511	311
	3	Expected more from this movie, but it was okay.	512	312
	5	Incredible cinematography! A visual masterpiece.	513	313
	4	Solid film with a compelling storyline.	514	314
	3	Not my cup of tea, but well-made nonetheless.	515	315

row(s) 1 - 15 of 20

Download

1.11 QUERIES & OUTPUT:

- SELECT * FROM THEATRE

The SQL query "SELECT * FROM THEATRE" retrieves all columns and rows from the "THEATRE" table in a database.

Results Explain Describe Saved SQL History

THEATRE_ID	NAME	LOCATION	CAPACITY	CONTACT_NUMBER
1	Starlight Cinema	123 Main Street Cityville	500	8463465467
2	Sunset Theater	456 Elm Street Townsville	300	6476543763
3	Majestic Movies	789 Oak Avenue Villagetown	700	7346876487
4	Galaxy Cinemas	321 Pine Road Suburbia	400	9847897598
5	Royal Theatre	654 Cedar Lane Hamletville	600	7834687456
6	Dreamland Cinema	987 Maple Drive Countryside	350	8734873467
7	Silver Screen Theater	210 Walnut Lane Metropolis	550	8793248364
8	Paradise Picture House	543 Cherry Street Seaside	450	8475874589
9	Oasis Odeon	876 Birch Boulevard Lakeside	650	9438759845
10	Magic Movie Palace	109 Pineapple Avenue Wonderland	550	8734873487
11	Grand Theater	321 Peach Place Riverside	700	3847987498
12	Crystal Cinema	654 Lemon Lane Mountainview	400	9384899384
13	Sunshine Cinematheque	987 Grape Grove Hilltop	300	8937983783
14	Golden Gate Theatre	210 Orange Orchard Cityscape	800	3489034898
15	Rainbow Movies	543 Strawberry Street Countryside	450	9899328948
16	Serenity Screens	876 Blueberry Boulevard Lakeside	600	9832479488
17	Emerald Empire Theater	109 Raspberry Road Riverside	500	9834983747
18	Twilight Theater	321 Cherry Blossom Avenue Metropolis	550	9294283774
19	Moonlight Multiplex	654 Magnolia Manor Seaside	700	9873846784
20	Stellar Cinema	987 Ivy Lane Suburbia	400	9999999984

20 rows returned in 0.02 seconds Download

- SELECT * FROM MOVIE

The SQL query "SELECT * FROM MOVIE" retrieves all columns and rows from the "MOVIE" table.

Results Explain Describe Saved SQL History

MOVIE_ID	TITLE	GENRE	DIRECTOR	RELEASE_DATE
301	Avengers: Endgame	Action, Adventure, Sci-Fi	Anthony Russo	04/26/2019
302	Titanic Night	"Drama, Romance"	James Cameron	12/19/1997
303	The Lion King	"Animation, Adventure, Drama"	Roger Allers	06/24/1994
304	Jurassic Park	"Adventure, Sci-Fi"	Steven Spielberg	06/11/1993
305	Titanic Night	"Action, Adventure, Fantasy"	James Cameron	12/18/2009
306	The Dark Knight	"Action, Crime, Drama"	Christopher Nolan	07/18/2008
307	Inception	"Action, Adventure, Sci-Fi"	Christopher Nolan	07/16/2010
308	Forrest Gump	"Drama, Romance"	Robert Zemeckis	07/06/1994
309	The Shawshank Redemption	Drama	Frank Darabont	10/14/1994
310	The Matrix	"Action, Sci-Fi"	Lana Wachowski	03/31/1999
311	Interstellar	"Adventure, Drama, Sci-Fi"	Christopher Nolan	11/07/2014
312	The Lord of the Rings: The Return of the King	"Action, Adventure, Drama"	Peter Jackson	12/17/2003
313	The Godfather	"Crime, Drama"	Francis Ford Coppola	03/24/1972
314	The Godfather: Part II	"Crime, Drama"	Francis Ford Coppola	12/20/1974
315	The Dark Knight Rises	"Action, Adventure, Thriller"	Christopher Nolan	07/20/2012
316	The Shawshank Redemption	Drama	Frank Darabont	10/14/1994
317	Schindler's List	"Biography, Drama, History"	Steven Spielberg	12/15/1993
318	The Lord of the Rings: The Fellowship of the Ring	"Action, Adventure, Drama"	Peter Jackson	12/19/2001
319	Fight Club	Drama	Edward Norton	10/15/1999
320	Forrest Gump	"Drama, Romance"	Robert Zemeckis	07/06/1994

20 rows returned in 0.00 seconds Download

- SELECT * FROM CUSTOMER

The SQL query "SELECT * FROM CUSTOMER" retrieves all columns and rows from the "CUSTOMER" table in a database.

Results Explain Describe Saved SQL History

CUSTOMER_ID	CUSTOMER_NAME	CUSTOMER_EMAIL	CUSTOMER_PHONE_NUMBER	MEMBERSHIP_STATUS
501	John Smith	john17@example.com	555-5678	Premium
502	Alice Johnson	alice@example.com	555-1234	Regular
503	Michael Brown	michael@example.com	555-9012	Premium
504	Sarah Davis	sarah@example.com	555-3456	Regular
505	Emily Wilson	emily@example.com	555-6789	Premium
506	David Martinez	david@example.com	555-0123	Regular
507	Jennifer Taylor	jennifer@example.com	555-7890	Premium
508	James Anderson	james@example.com	555-2345	Regular
509	Jessica Thomas	jessica@example.com	555-8901	Premium
510	Robert Lee	robert@example.com	555-2345	Regular
511	Lisa Hall	lisa@example.com	555-6789	Premium
512	Kevin Young	kevin@example.com	555-0123	Regular
513	Amanda Clark	amanda@example.com	555-3456	Premium
514	Daniel Rodriguez	daniel@example.com	555-6789	Regular
515	Nicole Walker	nicole@example.com	555-9012	Premium
516	Matthew Hill	matthew@example.com	555-2345	Regular
517	Samantha King	samantha@example.com	555-5678	Premium
518	Christopher Wright	christopher@example.com	555-8901	Regular
519	Ashley White	ashley@example.com	555-0123	Premium
520	Joshua Scott	joshua@example.com	555-3456	Regular

20 rows returned in 0.01 seconds Download

- SELECT Name, Capacity
  FROM Theatre
  WHERE Capacity > (SELECT AVG(Capacity) FROM Theatre);

  SELECT Name, Capacity: This part of the query specifies that we want to retrieve two columns from the "Theatre" table: "Name" and "Capacity".
  WHERE Capacity > (SELECT AVG(Capacity) FROM Theatre): This is the condition specified for filtering rows. It checks if the capacity of each theatre is greater than the average capacity of all theatres. The subquery (SELECT AVG(Capacity) FROM Theatre) calculates the average capacity across all theatres in the "Theatre" table, and the main query retrieves theatres whose capacity exceeds this average.

Results Explain Describe Saved SQL History

NAME	CAPACITY
Majestic Movies	700
Royal Theatre	600
Silver Screen Theater	550
Oasis Odeon	650
Magic Movie Palace	550
Grand Theater	700
Golden Gate Theatre	800
Serenity Screens	600
Twilight Theater	550
Moonlight Multiplex	700

10 rows returned in 0.01 seconds Download

- SELECT Movie.Title, COUNT(*) AS Total_Booked_Tickets
  FROM SCHEDULE, MOVIE
  WHERE Schedule.Movie_ID = Movie.Movie_ID
  GROUP BY Movie.Title;

  SELECT Movie.Title, COUNT(*) AS Total_Booked_Tickets: This part of the query specifies that we want to retrieve two columns: "Title" from the "Movie" table and a count of booked tickets, aliased as "Total_Booked_Tickets". The count function COUNT(*) counts the number of rows returned by the query.
  FROM SCHEDULE, MOVIE: Indicates that we are querying data from both the "SCHEDULE" and "MOVIE" tables.
  WHERE Schedule.Movie_ID = Movie.Movie_ID: Specifies the condition for joining the "SCHEDULE" and "MOVIE" tables. It matches the "Movie_ID" column from the "SCHEDULE" table with the "Movie_ID" column from the "MOVIE" table, ensuring that we only retrieve data for movies that have corresponding entries in the schedule.

GROUP BY Movie.Title: Groups the results by the "Title" column from the "MOVIE" table. This means that the count of booked tickets will be calculated separately for each movie title.

Results Explain Describe Saved SQL History

TITLE	TOTAL_BOOKED_TICKETS
Titanic	1
Inception	1
The Matrix	1
The Lion King	1
The Lord of the Rings: The Return of the King	1
The Lord of the Rings: The Fellowship of the Ring	1
The Godfather: Part II	1
Schindler's List	1
Fight Club	1
Jurassic Park	1
More than 10 rows available. Increase rows selector to view more rows.	

10 rows returned in 0.00 seconds Download

- SELECT Movie.Genre, COUNT(*) AS Total_Tickets_Sold
  FROM Tickets, schedule, movie
  WHERE TICKETS.FK_Schedule_ID = Schedule.Schedule_ID AND
  Schedule.Movie_ID = Movie.Movie_ID
  GROUP BY Movie.Genre
  ORDER BY Total_Tickets_Sold DESC

  SELECT Movie.Genre, COUNT(*) AS Total_Tickets_Sold: This part of the query specifies that we want to retrieve two columns: "Genre" from the "Movie" table and a count of tickets sold, aliased as "Total_Tickets_Sold". The count function COUNT(*) counts the number of rows returned by the query.
  FROM Tickets, Schedule, Movie: Indicates that we are querying data from the "Tickets", "Schedule", and "Movie" tables.
  WHERE TICKETS.FK_Schedule_ID = Schedule.Schedule_ID AND Schedule.Movie_ID = Movie.Movie_ID: Specifies the conditions for joining the "Tickets", "Schedule", and "Movie" tables. It matches the foreign key "FK_Schedule_ID" from the "Tickets" table with the primary key "Schedule_ID" from the "Schedule" table and matches the "Movie_ID" column from the "Schedule" table with the "Movie_ID" column from the "Movie" table, ensuring that we only retrieve data for tickets sold for scheduled movies.

GROUP BY Movie.Genre: Group the results by the "Genre" column from the "Movie" table. This means that the count of tickets sold will be calculated separately for each movie genre.
ORDER BY Total_Tickets_Sold DESC: Orders the results by the "Total_Tickets_Sold" column in descending order. This ensures that movie genres with the highest number of tickets sold appear first in the results.

Results Explain Describe Saved SQL History

GENRE	TOTAL_TICKETS_SOLD
"Drama, Romance"	3
Drama	3
"Crime, Drama"	2
"Action, Adventure, Drama"	2
"Adventure, Sci-Fi"	1
"Action, Sci-Fi"	1
"Action, Adventure, Thriller"	1
Action, Adventure, Sci-Fi	1
"Action, Adventure, Fantasy"	1
"Action, Crime, Drama"	1
"Biography, Drama, History"	1
"Action, Adventure, Sci-Fi"	1
"Animation, Adventure, Drama"	1
"Adventure, Drama, Sci-Fi"	1

14 rows returned in 0.01 seconds Download

- UPDATE MOVIE SET TITLE = 'Title Night'
  WHERE Director = 'James Cameron'

  UPDATE MOVIE: Specifies that we are updating data in the "MOVIE" table.
  SET TITLE = 'Title Night': Sets the value of the "TITLE" column to 'Title Night'.
  WHERE Director = 'James Cameron': Specifies the condition for which rows to update. In this case, it updates rows where the value in the "Director" column is 'James Cameron'.

Autocommit Rows 20 Save Run

```
UPDATE Movie SET TITLE = 'Titanic Night'
WHERE Director = 'James Cameron'
```

Results Explain Describe Saved SQL History

```
2 row(s) updated.
```

0.07 seconds

- SELECT * FROM MOVIE WHERE Director = 'James Cameron'

SELECT *: This part of the query selects all columns from the specified table.
FROM movie: Indicates that we are querying data from the "MOVIE" table.
WHERE director = 'James Cameron': Specifies the condition for filtering rows. It selects only those rows where the value in the "director" column is 'James Cameron'.

Results Explain Describe Saved SQL History

MOVIE_ID	TITLE	GENRE	DIRECTOR	RELEASE_DATE
302	Titanic Night	"Drama, Romance"	James Cameron	12/19/1997
305	Titanic Night	"Action, Adventure, Fantasy"	James Cameron	12/18/2009

2 rows returned in 0.00 seconds Download

- UPDATE CUSTOMER SET CUSTOMER_Email = 'john17@example.com' WHERE CUSTOMER_NAME = 'John Smith'

UPDATE CUSTOMER: Specifies that we are updating data in the "CUSTOMER" table.**SET CUSTOMER_Email = 'john17@example.com':** Sets the value of the "CUSTOMER_Email" column to 'john17@example.com'.
WHERE CUSTOMER_NAME = 'John Smith': Specifies the condition for which rows to update. In this case, it updates rows where the value in the "CUSTOMER_NAME" column is 'John Smith'.

Autocommit Rows 20 Save Run

```
UPDATE Customer SET Customer_email = 'john17@example.com'
WHERE Customer_Name = 'John Smith'
```

Results Explain Describe Saved SQL History

```
1 row(s) updated.
```

0.01 seconds

- SELECT * From Customer
  WHERE Customer_Name = 'John Smith'

  SELECT *: This part of the query selects all columns from the specified table.
  FROM Customer: Indicates that we are querying data from the "Customer" table.
  WHERE Customer_Name = 'John Smith': Specifies the condition for filtering rows. It selects only those rows where the value in the "Customer_Name" column is 'John Smith'.

Results Explain Describe Saved SQL History

CUSTOMER_ID	CUSTOMER_NAME	CUSTOMER_EMAIL	CUSTOMER_PHONE_NUMBER	MEMBERSHIP_STATUS
501	John Smith	john17@example.com	555-5678	Premium

1 rows returned in 0.01 seconds Download

- UPDATE TICKETS SET Ticket_ID = 421
  WHERE FK_Schedule_ID = 201

  UPDATE TICKETS: Specifies that we are updating data in the "TICKETS" table.
  SET Ticket_ID = 421: Sets the value of the "Ticket_ID" column to 421.
  WHERE FK_Schedule_ID = 201: Specifies the condition for which rows to update. In this case, it updates rows where the value in the "FK_Schedule_ID" column is 201.

Autocommit Rows 20 Save Run

```
UPDATE TICKETS SET Ticket_ID = 421
WHERE FK_Schedule_ID = 201
```

Results Explain Describe Saved SQL History

```
1 row(s) updated.
```

0.00 seconds

- SELECT * from Tickets
  WHERE FK_Schedule_ID = 201

  SELECT *: This part of the query selects all columns from the specified table.
  FROM Tickets: Indicates that we are querying data from the "Tickets" table.
  WHERE FK_Schedule_ID = 201: Specifies the condition for filtering rows. It selects only those rows where the value in the "FK_Schedule_ID" column is 201.

Results Explain Describe Saved SQL History

TICKET_ID	SEAT_NUMBER	PRICE	STATUS	FK_SCHEDULE_ID	FK_MOVIE_ID	FK_CUSTOMER_ID
421	A12	$12.50	Booked	201	301	501

1 rows returned in 0.00 seconds Download

- DELETE FROM TICKETS
  WHERE Seat_Number='A12'

DELETE FROM TICKETS: Specifies that we are deleting data from the "TICKETS" table.
WHERE Seat_Number = 'A12': Specifies the condition for which rows to delete. In this case, it deletes rows where the value in the "Seat_Number" column is 'A12'.

Autocommit Rows 20 Save Run

```
DELETE FROM Tickets
WHERE Seat_Number = 'A12';
```

Results Explain Describe Saved SQL History

```
1 row(s) deleted.
```

0.01 seconds

- SELECT * from Tickets

  SELECT *: This part of the query selects all columns from the specified table.
  FROM Tickets: Indicates that we are querying data from the "Tickets" table.

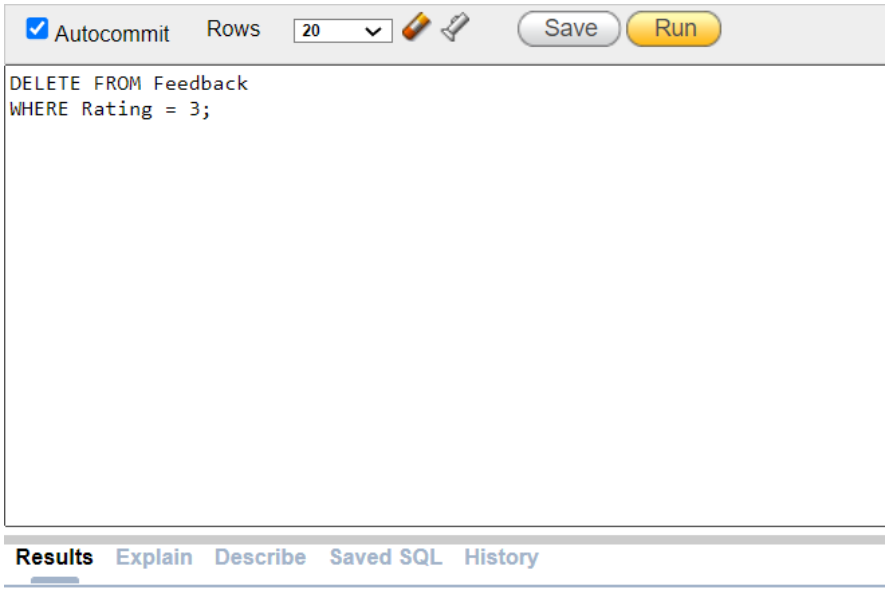
Results Explain Describe Saved SQL History

TICKET_ID	SEAT_NUMBER	PRICE	STATUS	FK_SCHEDULE_ID	FK_MOVIE_ID	FK_CUSTOMER_ID
402	B8	$10.00	Booked	202	302	502
403	C15	$8.50	Booked	203	303	503
404	D3	$9.00	Booked	204	304	504
405	E20	$11.00	Booked	205	305	505
406	F10	$13.00	Booked	206	306	506
407	G5	$12.00	Booked	207	307	507
408	H14	$9.50	Booked	208	308	508
409	I18	$10.50	Booked	209	309	509
410	J7	$11.50	Booked	210	310	510
411	K4	$13.50	Booked	211	311	511
412	L11	$14.00	Booked	212	312	512
413	M9	$8.00	Booked	213	313	513
414	N17	$7.50	Booked	214	314	514
415	O13	$10.00	Booked	215	315	515
416	P19	$11.00	Booked	216	316	516
417	Q6	$9.00	Booked	217	317	517
418	R21	$12.50	Booked	218	318	518
419	S12	$10.00	Booked	219	319	519
420	T16	$8.50	Booked	220	320	520

19 rows returned in 0.00 seconds Download

- DELETE FROM FEEDBACK
  WHERE Rating= 3

  DELETE FROM FEEDBACK: Specifies that we are deleting data from the "FEEDBACK" table.
  WHERE Rating = 3: Specifies the condition for which rows to delete. In this case, it deletes rows where the value in the "Rating" column is equal to 3.




```
6 row(s) deleted.
```

0.00 seconds

- SELECT * FROM Feedback

  SELECT *: This part of the query selects all columns from the specified table.
  FROM Feedback: Indicates that we are querying data from the "Feedback" table

Results Explain Describe Saved SQL History

RATING	COMMENTS	FK_CUSTOMER_ID	FK_MOVIE_ID
5	Amazing movie! Loved every minute of it.	501	301
4	Great storyline, but pacing was a bit slow.	502	302
5	Classic film! Still holds up after all these years.	504	304
4	Visually stunning! A must-watch for fans of the genre.	505	305
5	Absolutely loved it! Will definitely watch again.	507	307
4	Good movie, but some scenes felt unnecessary.	508	308
5	Fantastic performance by the lead actor!	510	310
4	Mind-bending plot kept me engaged throughout.	511	311
5	Incredible cinematography! A visual masterpiece.	513	313
4	Solid film with a compelling storyline.	514	314
5	Edge-of-your-seat suspense! Couldn't look away.	516	316
4	Historically accurate and emotionally gripping.	517	317
5	Absolutely stunning! Deserves all the praise it gets.	519	319
4	Enjoyable movie with a satisfying ending.	520	320

14 rows returned in 0.00 seconds Download

- INSERT INTO FEEDBACK (RATING, COMMENTS, FK_CUSTOMER_ID, FK_MOVIE_ID)
  VALUES (1, 'Mind-Blowing', 503, 303);

  INSERT INTO FEEDBACK: Specifies that you want to insert data into the "FEEDBACK" table.
  (RATING, COMMENTS, FK_CUSTOMER_ID, FK_MOVIE_ID): Specifies the columns into which you're inserting data.
  VALUES: Indicates that you're providing the values to be inserted.
  (1, 'Mind-Blowing', 503, 303): Provides the values to be inserted. In this case, the rating is 1, the comment is 'Mind-Blowing', the foreign key for the customer ID is 503, and the foreign key for the movie ID is 303.

Autocommit Rows 20 Save Run

```
INSERT INTO FEEDBACK (RATING, COMMENTS, FK_CUSTOMER_ID, FK_MOVIE_ID)
VALUES (1, 'Mind-Blowing', 503, 303);
```

Results Explain Describe Saved SQL History

```
1 row(s) inserted.
```

0.00 seconds

- SELECT * FROM FEEDBACK

SELECT *: This part of the query selects all columns from the specified table.
FROM Feedback: Indicates that we are querying data from the "Feedback" table

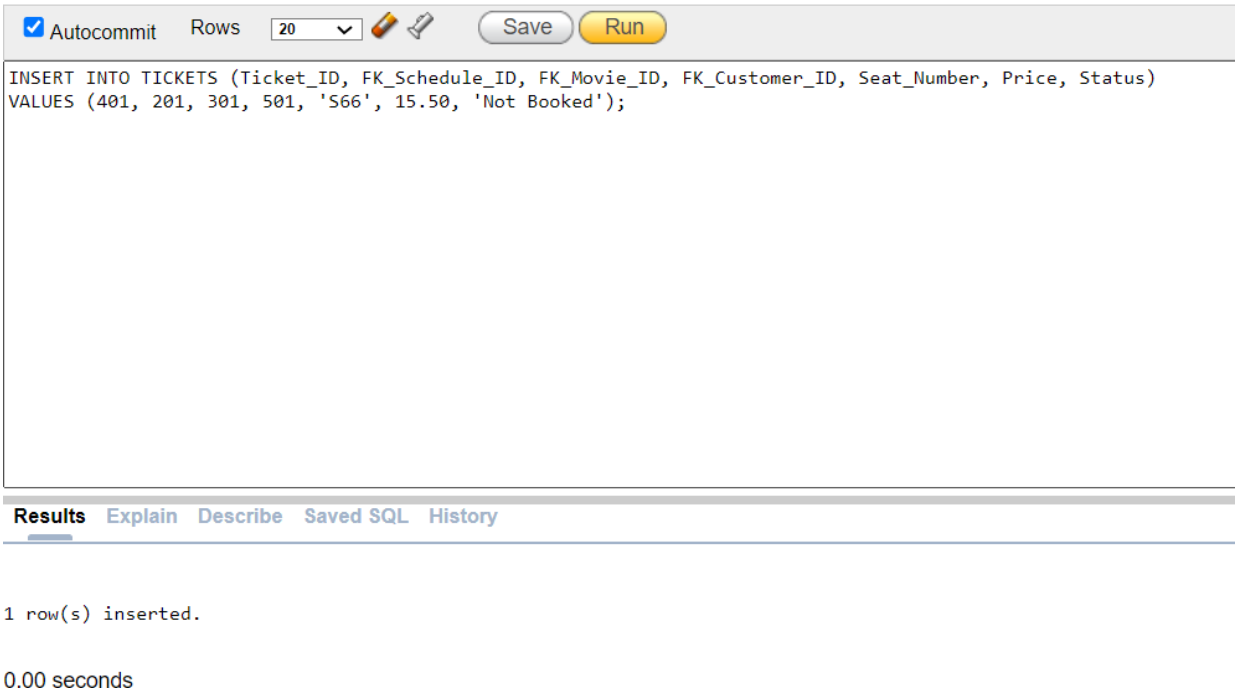
Results Explain Describe Saved SQL History

RATING	COMMENTS	FK_CUSTOMER_ID	FK_MOVIE_ID
5	Amazing movie! Loved every minute of it.	501	301
4	Great storyline, but pacing was a bit slow.	502	302
5	Classic film! Still holds up after all these years.	504	304
4	Visually stunning! A must-watch for fans of the genre.	505	305
5	Absolutely loved it! Will definitely watch again.	507	307
4	Good movie, but some scenes felt unnecessary.	508	308
5	Fantastic performance by the lead actor!	510	310
4	Mind-bending plot kept me engaged throughout.	511	311
5	Incredible cinematography! A visual masterpiece.	513	313
4	Solid film with a compelling storyline.	514	314
5	Edge-of-your-seat suspense! Couldn't look away.	516	316
4	Historically accurate and emotionally gripping.	517	317
5	Absolutely stunning! Deserves all the praise it gets.	519	319
4	Enjoyable movie with a satisfying ending.	520	320
1	Mind-Blowing	503	303

15 rows returned in 0.00 seconds Download

- INSERT INTO TICKETS (Ticket_ID, Schedule_ID, Movie_ID, Customer_ID, Seat_Number, Price, Status)
  VALUES (401, 201, 301, 501, 'S66', 15.50, 'Not Booked');

  INSERT INTO TICKETS: Specifies that you want to insert data into the "TICKETS" table.
  (Ticket_ID, Schedule_ID, Movie_ID, Customer_ID, Seat_Number, Price, Status): Specifies the columns into which you're inserting data.
  VALUES: Indicates that you're providing the values to be inserted.
  (401, 201, 301, 501, 'S66', 15.50, 'Not Booked'): Provides the values to be inserted. In this case, the ticket ID is 401, the schedule ID is 201, the movie ID is 301, the customer ID is 501, the seat number is 'S66', the price is 15.50, and the status is 'Not Booked'.



- SELECT * FROM TICKETS

  SELECT *: This part of the query selects all columns from the specified table.
  FROM Tickets: Indicates that we are querying data from the "Tickets" table.

Results Explain Describe Saved SQL History

TICKET_ID	SEAT_NUMBER	PRICE	STATUS	FK_SCHEDULE_ID	FK_MOVIE_ID	FK_CUSTOMER_ID
401	S66	15.5	Not Booked	201	301	501
402	B8	$10.00	Booked	202	302	502
403	C15	$8.50	Booked	203	303	503
404	D3	$9.00	Booked	204	304	504
405	E20	$11.00	Booked	205	305	505
406	F10	$13.00	Booked	206	306	506
407	G5	$12.00	Booked	207	307	507
408	H14	$9.50	Booked	208	308	508
409	I18	$10.50	Booked	209	309	509
410	J7	$11.50	Booked	210	310	510
411	K4	$13.50	Booked	211	311	511
412	L11	$14.00	Booked	212	312	512
413	M9	$8.00	Booked	213	313	513
414	N17	$7.50	Booked	214	314	514
415	O13	$10.00	Booked	215	315	515
416	P19	$11.00	Booked	216	316	516
417	Q6	$9.00	Booked	217	317	517
418	R21	$12.50	Booked	218	318	518
419	S12	$10.00	Booked	219	319	519
420	T16	$8.50	Booked	220	320	520

20 rows returned in 0.00 seconds Download