bookmytalkies
A Movie Booking App (DB)

Vinod Shivarudrappa

8-13-2021

Table of Contents

Overview

- This app shows all movies available across all theaters at a location (could be a city or town)
- When a User/Customer opens this app, they would be presented with currently running Movies
- Based on the Customer's selection, the app would then show the current Showtimes per the user's selected location radius.
- Once the user selects for Showtime and Theater, the app would then prompt to choose a Seat and ultimately make a Booking.
- For the sake of simplicity, all Theaters have a single screen and a seating capacity of 50

Use Cases and Fields

1. The most important use of the database for this app is the data/information about theaters and their current showtimes. Approved theaters, their address along with showtimes should always be stored in the database. I envision the tables and fields below for this purpose:

Table: **Theater**

Field	What it Stores	Why it's Needed
Theater ID	Unique ID of a theater in the DB	To uniquely identify a theater such that there are no duplicates
Theater Name	Name of the theater	A name of the theater identifies it on the app
Street	Street where the theater is located	Location information can be retrieved from the address – this will help show the relevant theaters per user location
City	City where the theater is located	
State	State where the theater is located	
Zip Code	Zip where the theater is located	
Tax Rate	Tax rate based on address	Tax rate will contribute to the booking amount at the end

Table: **Seats**

Field	What it Stores	Why it's Needed
Seat ID	Unique ID of a seat in the DB	To uniquely identify a seat such that there are no duplicates
Seat Row	Row alphabet of the seat	Seats in a theater a matrix of alphabetical rows and numbered seats. This is the alphabetical row

Seat Number	Seat Number of the seat	In the matrix mentioned above, this number corresponds to the column that identifies the seat in an alphabetical row

Table: **Showtimes**

Field	What it Stores	Why it's Needed
Showtime ID	Unique ID of a showtime in the DB	To uniquely identify a showtime such that there are no duplicates
Start Date	Start Date of a movie at a theater	Needed to track active movies
End Date	End Date of a movie at a theater	Needed to track active movies
Start Time	Start time of the show at a theater	Needed to identify active movies playing and the slot in which the user may book their tickets
End Time	End time of the show at a theater	Same as above, except ending time
Language	Language that the show is in	A movie could be dubbed in multiple languages – this identifies the language that this show is played in
Price	Pricing information for this showtime at a theater	With the price and taxes, the final amount that the customer pays is calculated

Table: **Movie**

Field	What it Stores	Why it's Needed
Movie ID	Unique ID of a movie in the DB	To uniquely identify a movie such that there are no duplicates
Movie Title	Name of the movie	Name of the movie that shows on the app
Release Date	Date the movie was released	Identifies active or upcoming movie
Runtime	Duration in minutes of the movie's play time	Required if the user wants to sort by play time
Status	Status of the movie – Released, Planned, In Production or Postproduction	Required to book tickets – only if the movie is released. If movie in post-production, the app could say "coming soon"
Adult?	Flag to indicated if movie is rated R	Required to ensure customer booking the ticket is an adult

2. Next, the app would need to store user information as they sign up and book tickets. For this purpose, the database could have a Customer table as below

Table: **Customer**

Field	What it Stores	Why it's Needed
Customer ID	Unique ID of a customer/user in the DB	To uniquely identify a customer such that there are no duplicates
First Name	First Name of the Customer	Along with the Last Name, this will identify the customer and the booking
Last Name	Last Name of the Customer	Along with the First Name, this will identify the customer and the booking
Phone Number	Numerical phone number	Optional field that could be used for booking reminders/two-factor auth
Email	Email Address	Email address can serve as user ID for the app, also to send over tickets after booking
Date of Birth	Date of birth of the customer	Need to determine age for booking
Zip Code	Zip code of the customer	Required to store information about the customer's home location

3. When the customer makes a Booking, that information can be stored in the Booking table.
In order to track seats that have been booked for a showtime, a separate table would be required - the Seats table only has a list of all seats but no booking information. We'll call it Seat Reservation – and have IDs fields from Booking, Seat tables. Showtime information can be inferred from Booking.

Table: **Booking**

Field	What it Stores	Why it's Needed
Booking ID	Unique ID of the booking made in the DB	To uniquely identify a booking such that there are no duplicates
Net Amount	Final price of the ticket for a show at a theater	With Showtime.price and taxes, this represents the final amount that the customer pays
Taxes	Taxes paid by the customer	Used to calculate Net Amount – based on theater location
Booking Time	Timestamp at which the booking was made	Required to track history
Booking Status	Status of the made Booking – Confirmed/Cancelled	Required to track history

Table: **Seat Reservation**

Field	What it Stores	Why it's Needed
Seat Reservation ID	ID field that identifies each reservation	To uniquely identify reservations for a seat booking made for that showtime
Booking ID	Booking ID – from Booking table	To identify a booking made
Seat ID	Seat ID – from Seats table	To identify the seat(s) that have been booked for this showtime

Structural Database Rules

Defining structural rules based on use cases. Walking along the process of how a customer would experience making a Movie Booking, here are the steps and structural rules that I foresee for my bookmytalkies database:

User/Customer opens the app and is presented with Movies currently playing

- At this stage, the user is browsing the movies that are **currently playing**.
- Meaning, the Movies entity will be accessed, along with Showtimes and Theater entities since Movies on its own doesn't have enough information to tell if it is active in theaters

I see the below rules for this use case:

R1: Each Movie may be associated with a Theater; each Theater may be associated with many Movies
R2: Each Movie may be associated with many Showtimes; each Showtime is associated with only one Movie
R3: Each Showtime must be associated with only one Theater; each Theater may be associated with many Showtimes

The following can be inferred from the above rules:

- A movie can exist on its own in the database – it's a central entity which in the real world can contain information about all movies that exist – even the ones that would be released in the future
  (side note: this information can be retrieved through an API perhaps).
- However, for it show up on the app, it needs to have an active showtime and a theater associated
- For example, if a theater admin (that has access only to his Theater in the DB) decides they will play a movie, from an administration standpoint, at the backend though a friendly UI, they would tentatively enter the movie's showtimes. This can be tracked using Start Date and End Date – which would automatically render the movie to be active and display on the app. Start Time and End Time in Showtimes entity would represent the "show times" of that movie
- It is also to note that a Showtime cannot exist without a Theater and a Movie – a very weak entity
- A unique Showtime can also only be associated only with one Theater

Based on the Customer's selection of a Movie, the app would then prompt the user for their location and show the current Showtimes per the user's selected location radius. Once the user selects a Theater and Showtime, the app would then prompt to choose a Seat and ultimately make a Booking

- User's *current* location – is ephemeral since they might be a different location each time. This need not be stored in the database. The app could have it like a cookie locally on-device.
- Showtimes shown per location radius – this is based on the Address field in the Theater entity. Code could be written within the app to derive the zip, no special DB rules required here
- Seat selection – this is based on the Showtime and Theater.
  Since we are assuming that each theater has only one screen, there is no relationship required with Movie – only Showtime and Theater
  Customer chooses whether he'd like to book:
  - Normal Seat : No extra price
  - Accessible Seat : No extra price
  - Reclining Seat : +5% of Showtime price
  - Box Seat : +10% of Showtime price

  Based on the above selection, the final booking price would be calculated

- Make a Booking – the app would record the customer's information and make an entry in the Booking table. One same customer can make multiple bookings – this is not the same as the customer choosing more than one seat in the same booking. Seat information from these bookings are stored in Seat Reservation table

The following can be inferred from the above rules:

R4: Each Theater must be associated with multiple Seats; each Seat must be associated with only one Theater
R5: Each Customer may make multiple Bookings; each Booking must be associated with only one Customer
R6: Each Showtime may have a Booking; each Booking however must have a Showtime reserved
R7: Each Booking must have a Seat Reservation; each Seat Reservation must be on a Booking
R8: Each Seat Reservation must have only one Seat reserved; Each Seat may or may not have a Seat Reservation
R9: Each Seat is a Normal Seat, an Accessible Seat, a Reclining Seat or a Box Seat

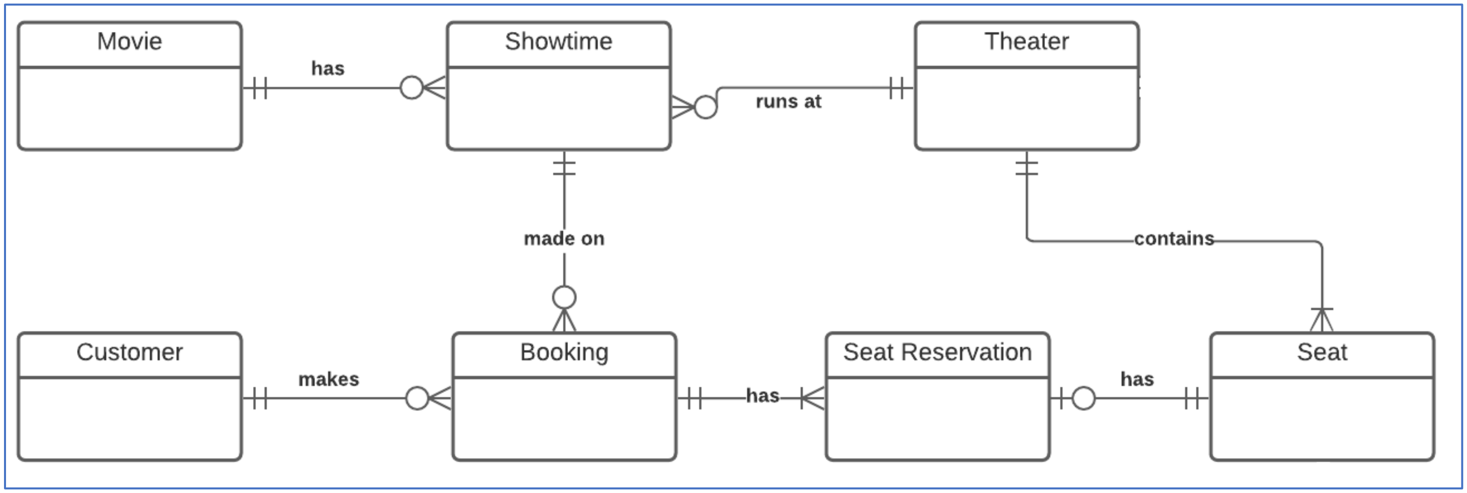
Business Rules – Constraints

1. A customer may sign up and not make any booking, but they must be 13 or older to do so
2. Customer must be at least the age of 18 to make a booking – especially true for R rated movies
3. Customer email address must be unique

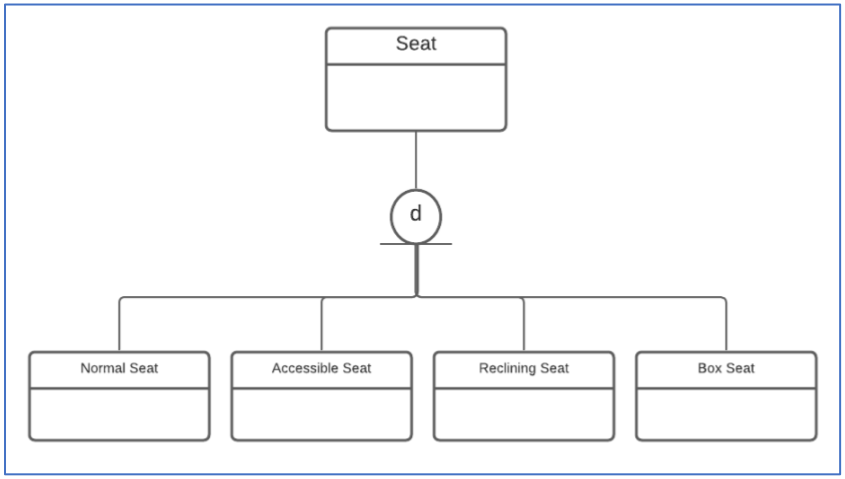
Entity-relationship Diagram

Based on the rules elucidated above, here are the Entity Relationship Diagrams:

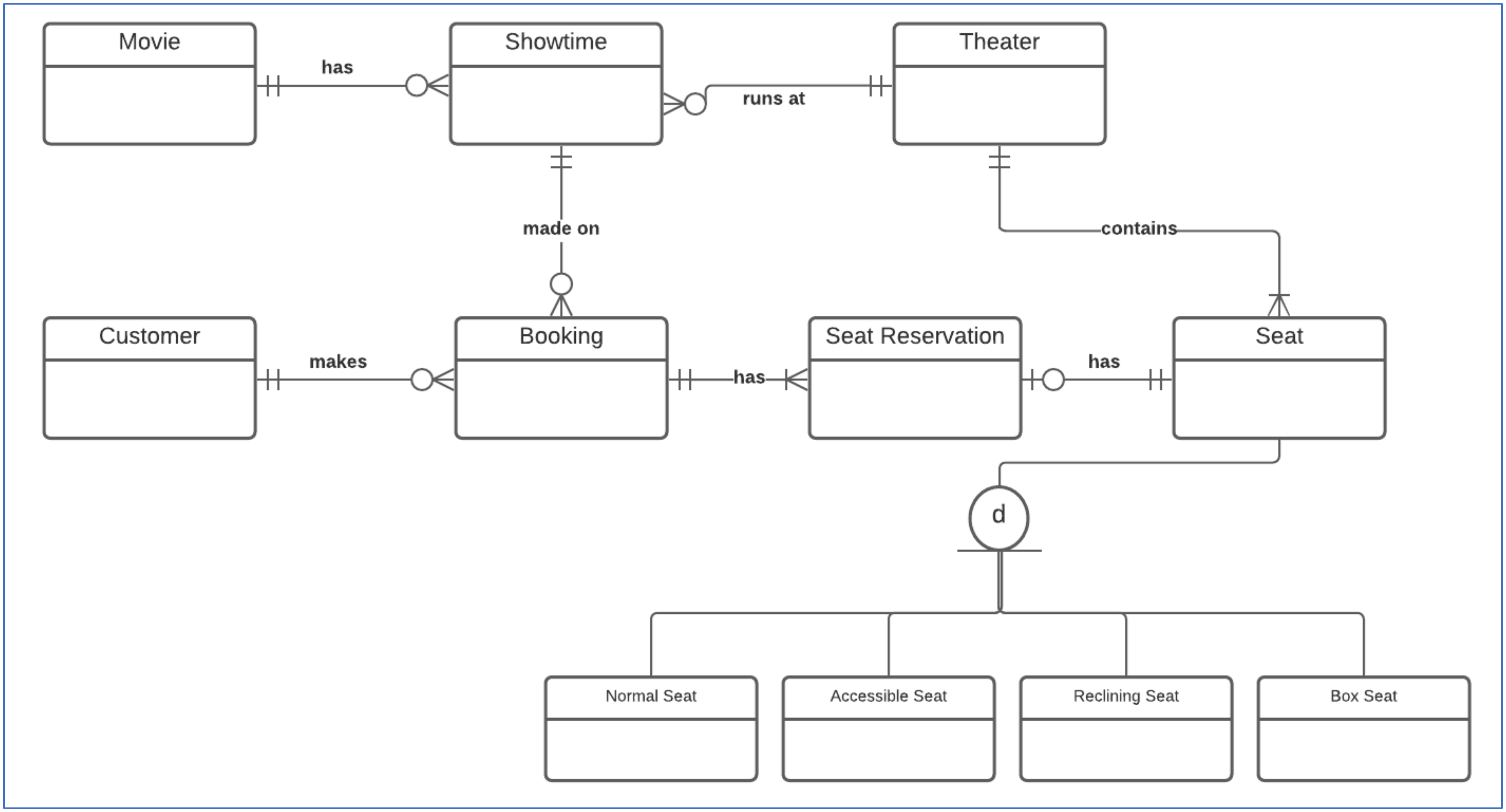
Conceptual ERD:



Specialization for Seat entity:



Conceptual diagram with specialization included:



Physical ERD:

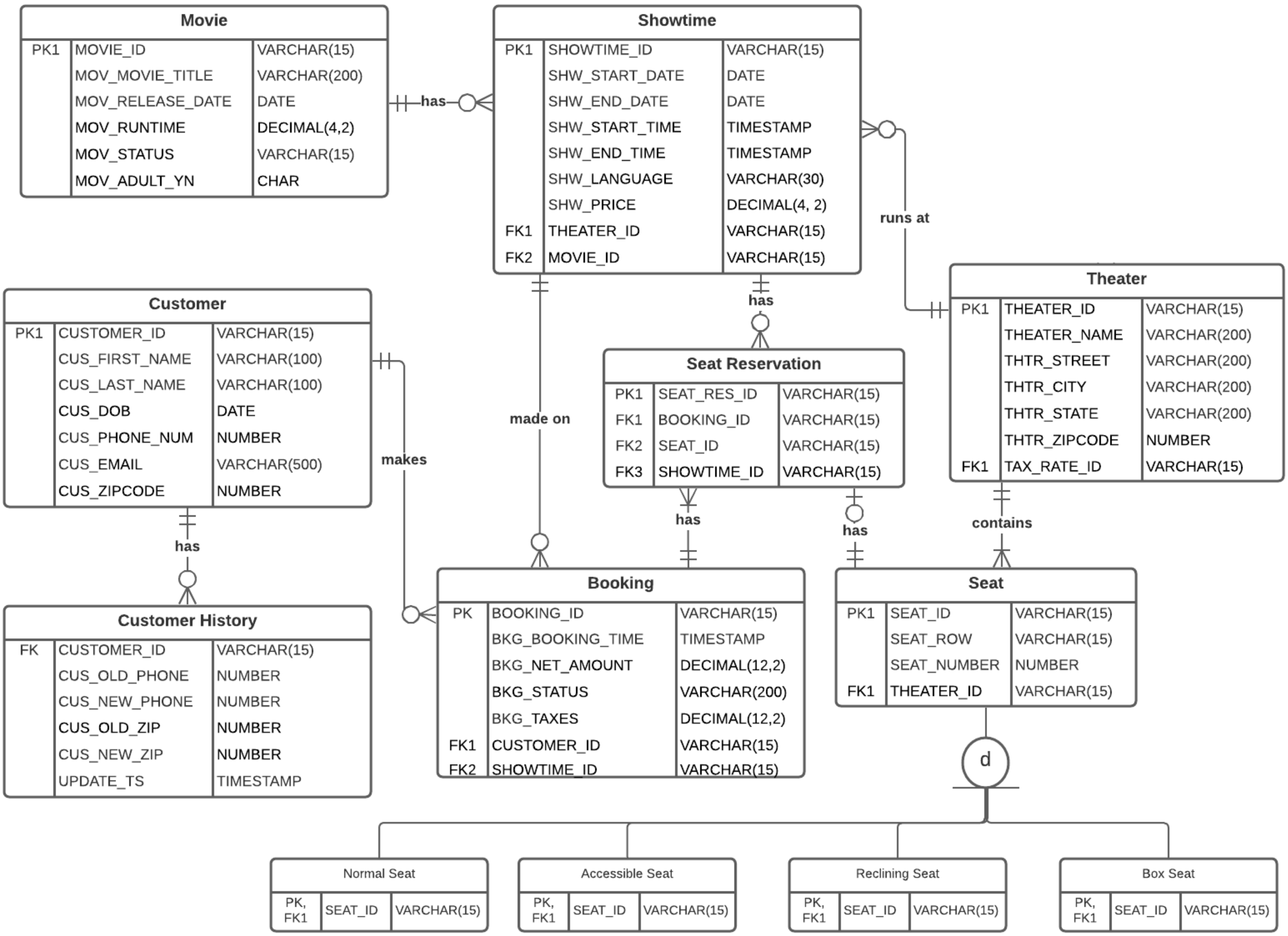
Additional Fields:

After a reflection of what fields would be required and nice-to-have, I have added the below fields:

Table	Attribute	Data Type	Reasoning
Customer	Date of Birth	DATE	Need to determine customer age while booking
Customer	Zip Code	NUMBER	Required to store customer's home location
Theater	Street	VARCHAR (200)	Exploding the address field to St, City, State and Zip code
Theater	City	VARCHAR (200)	
Theater	State	VARCHAR (200)	
Theater	Zip Code	VARCHAR (200)	
Booking	Booking Time	TIMESTAMP	Updated Booking Date → Timestamp
Booking	Booking Status	VARCHAR (200)	Current status of the booking – Confirmed/Cancelled
Seat Reservation	Showtime ID	VARCHAR (15)	Added SHOWTIME_ID as a foreign key here to prevent the same seat being booked more than once

I have also removed the field "Ticket No." from the Booking table since it was redundant with Booking ID.

I have tried to imagine and capture the necessary attributes for bookmytalkies. Perhaps as the app grows and becomes more complex, a need for more fields within the database might become necessary but given the use cases and structural rules for now, these attributes are enough. These changes have been made to reflect in the physical ERD below.



Movie

PK1	MOVIE_ID	VARCHAR(15)
	MOV_MOVIE_TITLE	VARCHAR(200)
	MOV_RELEASE_DATE	DATE
	MOV_RUNTIME	DECIMAL(4,2)
	MOV_STATUS	VARCHAR(15)
	MOV_ADULT_YN	CHAR

Showtime

PK1	SHOWTIME_ID	VARCHAR(15)
	SHW_START_DATE	DATE
	SHW_END_DATE	DATE
	SHW_START_TIME	TIMESTAMP
	SHW_END_TIME	TIMESTAMP
	SHW_LANGUAGE	VARCHAR(30)
	SHW_PRICE	DECIMAL(4, 2)
FK1	THEATER_ID	VARCHAR(15)
FK2	MOVIE_ID	VARCHAR(15)

Theater

PK1	THEATER_ID	VARCHAR(15)
	THEATER_NAME	VARCHAR(200)
	THTR_STREET	VARCHAR(200)
	THTR_CITY	VARCHAR(200)
	THTR_STATE	VARCHAR(200)
	THTR_ZIPCODE	NUMBER
FK1	TAX_RATE_ID	VARCHAR(15)

Customer

PK1	CUSTOMER_ID	VARCHAR(15)
	CUS_FIRST_NAME	VARCHAR(100)
	CUS_LAST_NAME	VARCHAR(100)
	CUS_DOB	DATE
	CUS_PHONE_NUM	NUMBER
	CUS_EMAIL	VARCHAR(500)
	CUS_ZIPCODE	NUMBER

Seat Reservation

PK1	SEAT_RES_ID	VARCHAR(15)
FK1	BOOKING_ID	VARCHAR(15)
FK2	SEAT_ID	VARCHAR(15)
FK3	SHOWTIME_ID	VARCHAR(15)

Customer History

FK	CUSTOMER_ID	VARCHAR(15)
	CUS_OLD_PHONE	NUMBER
	CUS_NEW_PHONE	NUMBER
	CUS_OLD_ZIP	NUMBER
	CUS_NEW_ZIP	NUMBER
	UPDATE_TS	TIMESTAMP

Booking

PK	BOOKING_ID	VARCHAR(15)
	BKG_BOOKING_TIME	TIMESTAMP
	BKG_NET_AMOUNT	DECIMAL(12,2)
	BKG_STATUS	VARCHAR(200)
	BKG_TAXES	DECIMAL(12,2)
FK1	CUSTOMER_ID	VARCHAR(15)
FK2	SHOWTIME_ID	VARCHAR(15)

Seat

PK1	SEAT_ID	VARCHAR(15)
	SEAT_ROW	VARCHAR(15)
	SEAT_NUMBER	NUMBER
FK1	THEATER_ID	VARCHAR(15)

Normal Seat

PK, FK1	SEAT_ID	VARCHAR(15)

Accessible Seat

PK, FK1	SEAT_ID	VARCHAR(15)

Reclining Seat

PK, FK1	SEAT_ID	VARCHAR(15)

Box Seat

PK, FK1	SEAT_ID	VARCHAR(15)

Normalization

I aim to keep my database normalized at BCNF level. As a reference, here are one-line descriptions for each of the normal forms.

Normal Form	Description
First normal form (1NF)	Table format, no repeating groups, and PK identified
Second normal form (2NF)	1NF and no partial dependencies
Third normal form (3NF)	2NF and no transitive dependencies
Boyce-Codd normal form (BCNF)	Every determinant is a candidate key (special case of 3NF)

1st Normal Form (1NF) - Table format, no repeating groups, and PK identified

From the above physical ERD, here are my observations:
It is already in table format
Primary Keys have been identified along with foreign keys and here are the dependencies for each table

1. Movie
   MOVIE_ID → MOV_MOVIE_TITLE, MOV_RELEASE_DATE, MOV_RUNTIME, MOV_STATUS, MOV_ADULT_YN

2. Customer
   CUSTOMER_ID → CUS_FIRST_NAME, CUS_LAST_NAME, CUS_PHONE_NUM, CUS_EMAIL, CUS_ZIPCODE

3. Showtime
   SHOWTIME_ID → SHW_START_DATE, SHW_END_DATE, SHW_START_TIME, SHW_END_TIME, SHW_LANGUAGE, SHW_PRICE

4. Booking
   BOOKING_ID → BKG_TICKET_NUM, BKG_BOOKING_TIME, BKG_NET_AMOUNT, BKG_TAXES

5. Theater
   THEATER_ID → THEATER_NAME, THTR_STREET, THTR_CITY, THTR_STATE, THTR_ZIPCODE, THTR_TAX_RATE

6. Seat
   SEAT_ID → SEAT_ROW, SEAT_NUMBER, THEATER_ID

There are no repeating groups for the given relationships

With this, the ERD is at 1 NF. Proceeding to 2NF…

Second Normal Form (2NF) - 1NF and no partial dependencies

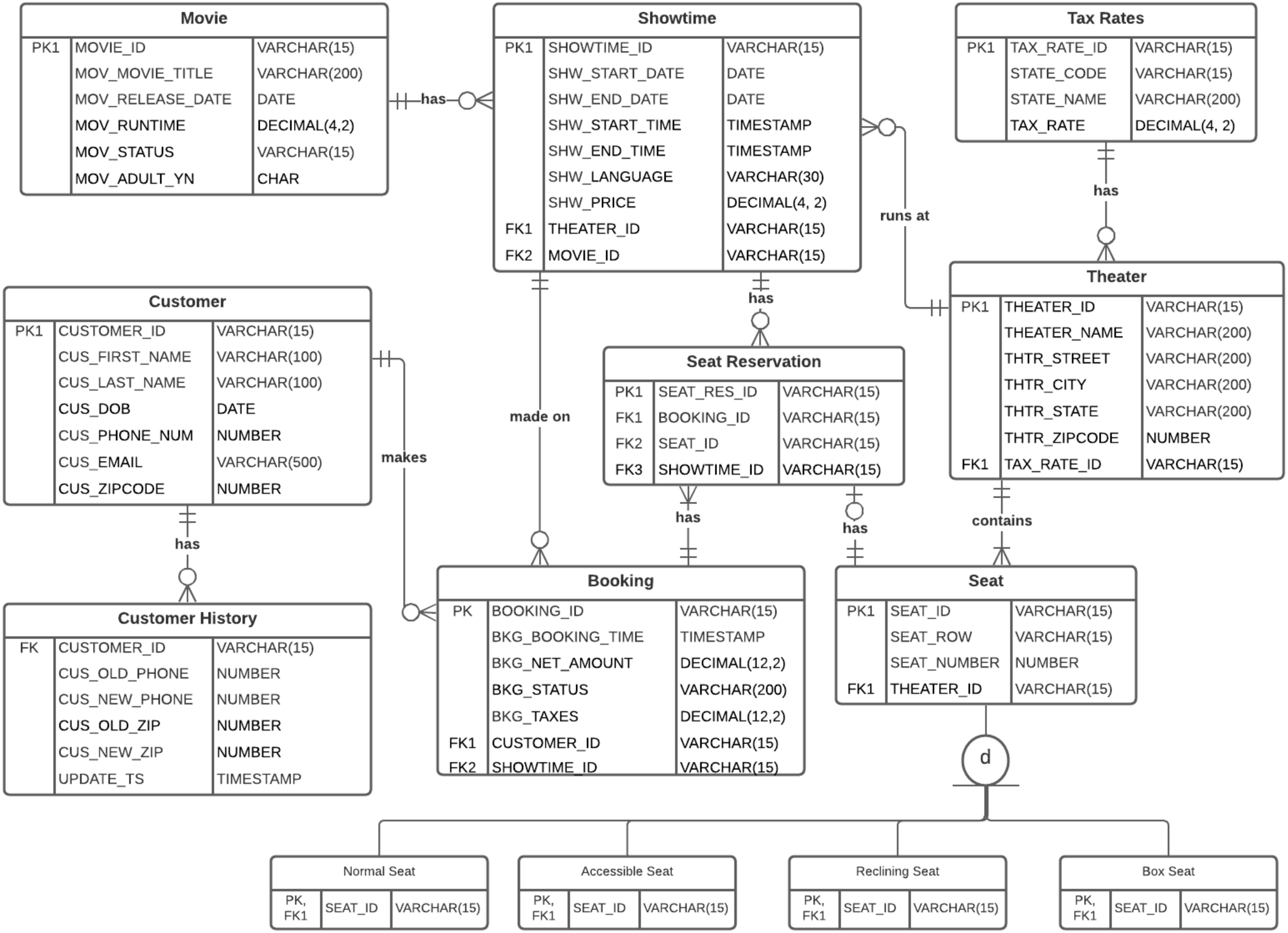
From the dependencies above, it is clear that

- The tables are in 1NF; and
- There are no partial dependencies. A partial dependency would exist if a subset of the candidate key (composite) is capable to derive other non-prime attributes; but all our keys are a single attribute

Third normal form (3NF) - 2NF and no transitive dependencies

Now that 2NF has been established, transitive dependencies need to be checked.
Just by simple observation, it is seen that Tax Rate in the Theater entity is dependent on State (keeping it sales tax at State level)
To remediate this, we introduce a new entity called Tax Rates that contain tax rate (as percentages) for each state – and link it to Theater

With this, the physical diagram at 3NF would be updated as below:



Movie

PK1	MOVIE_ID	VARCHAR(15)
	MOV_MOVIE_TITLE	VARCHAR(200)
	MOV_RELEASE_DATE	DATE
	MOV_RUNTIME	DECIMAL(4,2)
	MOV_STATUS	VARCHAR(15)
	MOV_ADULT_YN	CHAR

Showtime

PK1	SHOWTIME_ID	VARCHAR(15)
	SHW_START_DATE	DATE
	SHW_END_DATE	DATE
	SHW_START_TIME	TIMESTAMP
	SHW_END_TIME	TIMESTAMP
	SHW_LANGUAGE	VARCHAR(30)
	SHW_PRICE	DECIMAL(4, 2)
FK1	THEATER_ID	VARCHAR(15)
FK2	MOVIE_ID	VARCHAR(15)

Tax Rates

PK1	TAX_RATE_ID	VARCHAR(15)
	STATE_CODE	VARCHAR(15)
	STATE_NAME	VARCHAR(200)
	TAX_RATE	DECIMAL(4, 2)

Customer

PK1	CUSTOMER_ID	VARCHAR(15)
	CUS_FIRST_NAME	VARCHAR(100)
	CUS_LAST_NAME	VARCHAR(100)
	CUS_DOB	DATE
	CUS_PHONE_NUM	NUMBER
	CUS_EMAIL	VARCHAR(500)
	CUS_ZIPCODE	NUMBER

Seat Reservation

PK1	SEAT_RES_ID	VARCHAR(15)
FK1	BOOKING_ID	VARCHAR(15)
FK2	SEAT_ID	VARCHAR(15)
FK3	SHOWTIME_ID	VARCHAR(15)

Theater

PK1	THEATER_ID	VARCHAR(15)
	THEATER_NAME	VARCHAR(200)
	THTR_STREET	VARCHAR(200)
	THTR_CITY	VARCHAR(200)
	THTR_STATE	VARCHAR(200)
	THTR_ZIPCODE	NUMBER
FK1	TAX_RATE_ID	VARCHAR(15)

Customer History

FK	CUSTOMER_ID	VARCHAR(15)
	CUS_OLD_PHONE	NUMBER
	CUS_NEW_PHONE	NUMBER
	CUS_OLD_ZIP	NUMBER
	CUS_NEW_ZIP	NUMBER
	UPDATE_TS	TIMESTAMP

Booking

PK	BOOKING_ID	VARCHAR(15)
	BKG_BOOKING_TIME	TIMESTAMP
	BKG_NET_AMOUNT	DECIMAL(12,2)
	BKG_STATUS	VARCHAR(200)
	BKG_TAXES	DECIMAL(12,2)
FK1	CUSTOMER_ID	VARCHAR(15)
FK2	SHOWTIME_ID	VARCHAR(15)

Seat

PK1	SEAT_ID	VARCHAR(15)
	SEAT_ROW	VARCHAR(15)
	SEAT_NUMBER	NUMBER
FK1	THEATER_ID	VARCHAR(15)

Normal Seat

PK, FK1	SEAT_ID	VARCHAR(15)

Accessible Seat

PK, FK1	SEAT_ID	VARCHAR(15)

Reclining Seat

PK, FK1	SEAT_ID	VARCHAR(15)

Box Seat

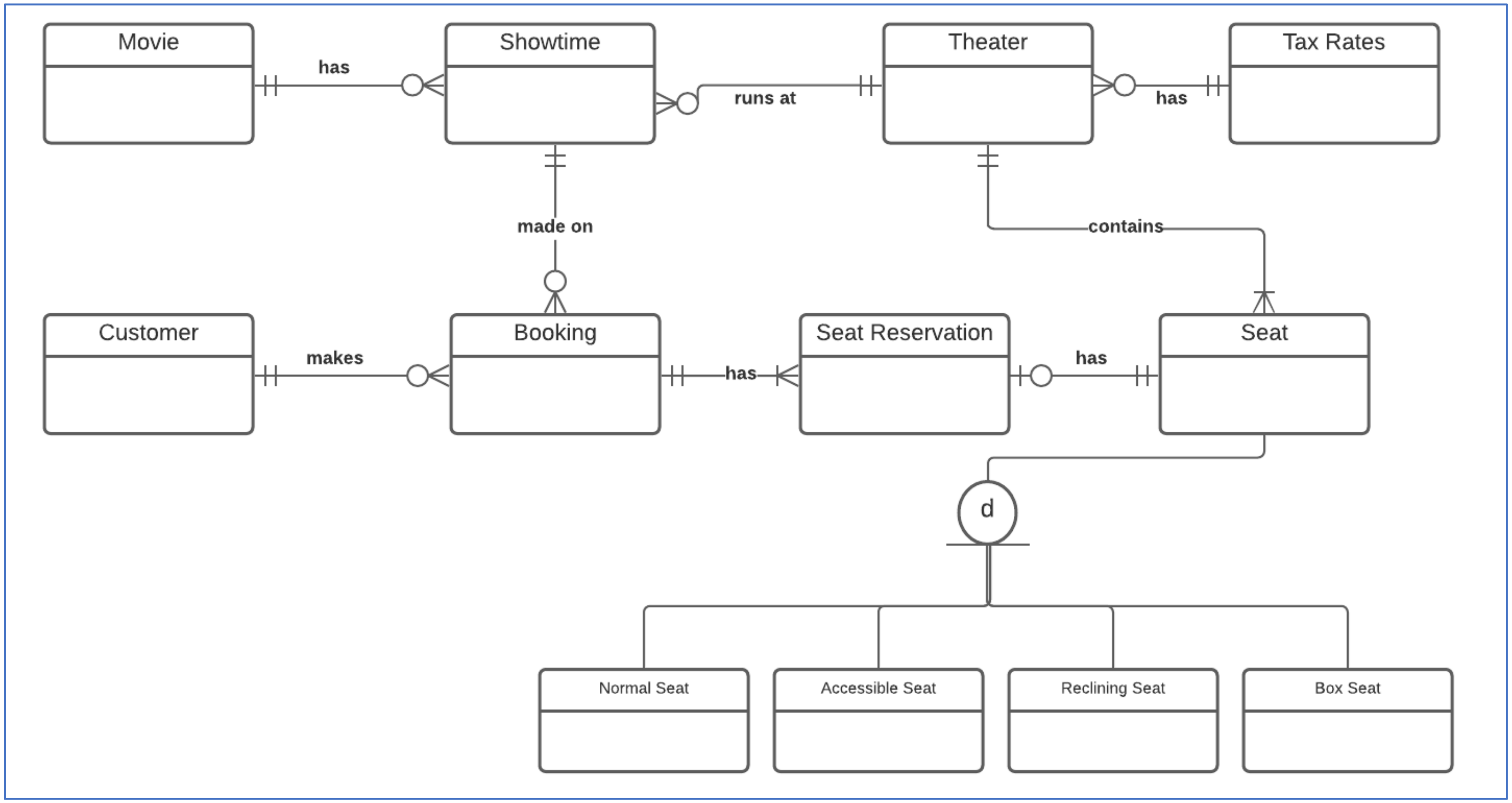
PK, FK1	SEAT_ID	VARCHAR(15)

Going further, upon further observation, I can still break the tables down so that the structure overall becomes normalized to BCNF – for example, the address fields in Theater can have their own table and I can reference them as foreign keys. Furthermore, customer phone number can be determined by customer ID and email if it was a composite key. However, this will add more complexity to an already complex database system for an app that is aimed to be MVP at this point. For this reason, I stop normalizing here and proceed to implement the database.

Adding the new entities gives rise to new database structural rules as below:

R10: Each Theater must be associated with at most one Tax Rate; a Tax Rate may be associated with all Theaters in that State

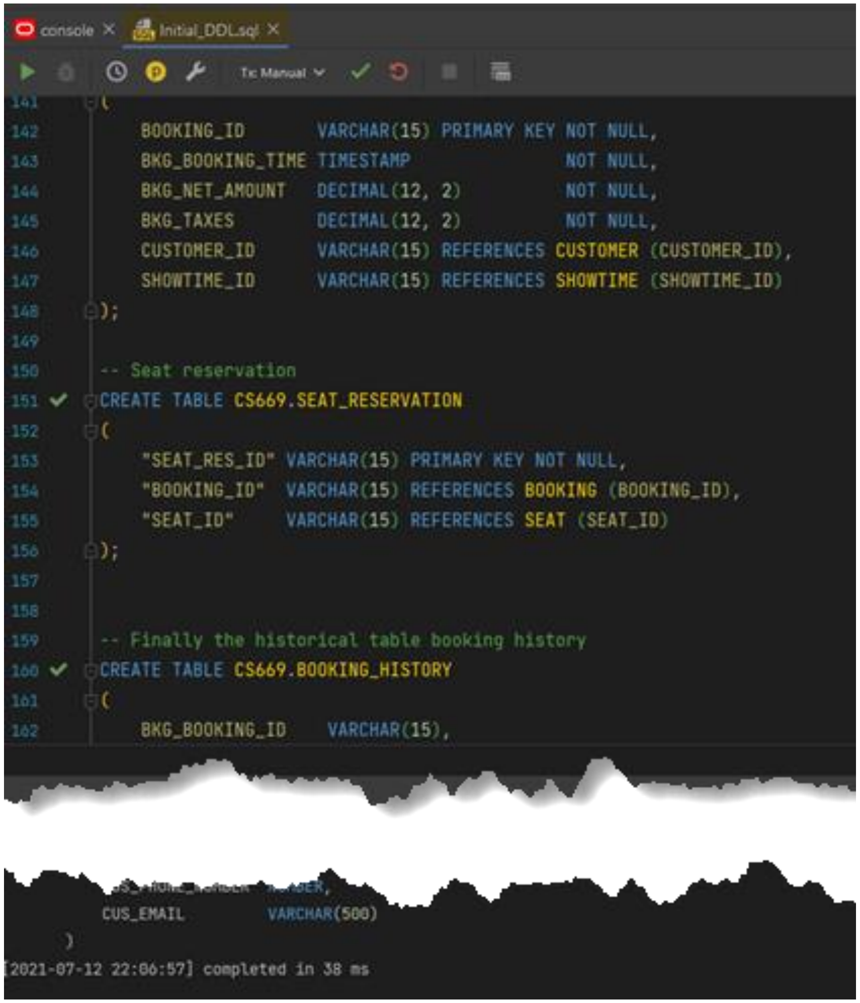
Here's my conceptual ERD after the above changes:



Implementation

Creating Initial Tables and Constraints

Here’s a screenshot of the executed DDL and the data diagram for the same – directly from the database. Please note I’ve used DataGrip for this execution.



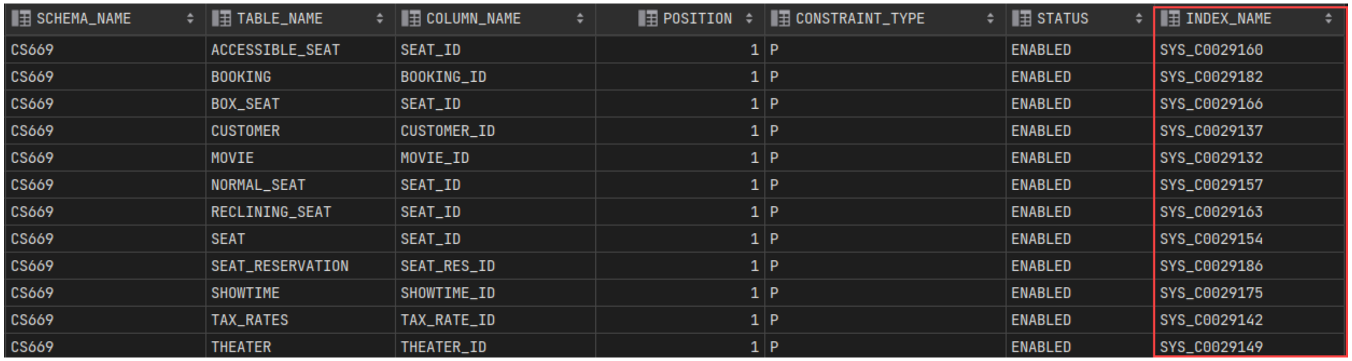
```
(
    BOOKING_ID       VARCHAR(15) PRIMARY KEY NOT NULL,
    BKG_BOOKING_TIME TIMESTAMP               NOT NULL,
    BKG_NET_AMOUNT   DECIMAL(12, 2)          NOT NULL,
    BKG_TAXES        DECIMAL(12, 2)          NOT NULL,
    CUSTOMER_ID      VARCHAR(15) REFERENCES CUSTOMER (CUSTOMER_ID),
    SHOWTIME_ID      VARCHAR(15) REFERENCES SHOWTIME (SHOWTIME_ID)
);

-- Seat reservation
CREATE TABLE CS669.SEAT_RESERVATION
(
    "SEAT_RES_ID" VARCHAR(15) PRIMARY KEY NOT NULL,
    "BOOKING_ID"  VARCHAR(15) REFERENCES BOOKING (BOOKING_ID),
    "SEAT_ID"     VARCHAR(15) REFERENCES SEAT (SEAT_ID)
);


-- Finally the historical table booking history
CREATE TABLE CS669.BOOKING_HISTORY
(
    BKG_BOOKING_ID    VARCHAR(15),
```

```
        CUS_PHONE_NUMBER    NUMBER,
        CUS_EMAIL           VARCHAR(500)
    )
[2021-07-12 22:06:57] completed in 38 ms
```

Identifying Indexes

- Primary Key Indexes – here is the list of the primary keys which are already indexed

- ACCESSIBLE_SEAT.SEAT_ID
- BOOKING.BOOKING_ID
- BOX_SEAT.SEAT_ID
- CUSTOMER.CUSTOMER_ID
- MOVIE.MOVIE_ID
- NORMAL_SEAT.SEAT_ID
- RECLINING_SEAT.SEAT_ID
- SEAT.SEAT_ID
- SEAT_RESERVATION.SEAT_RES_ID
- SHOWTIME.SHOWTIME_ID
- TAX_RATES.TAX_RATE_ID
- THEATER.THEATER_ID

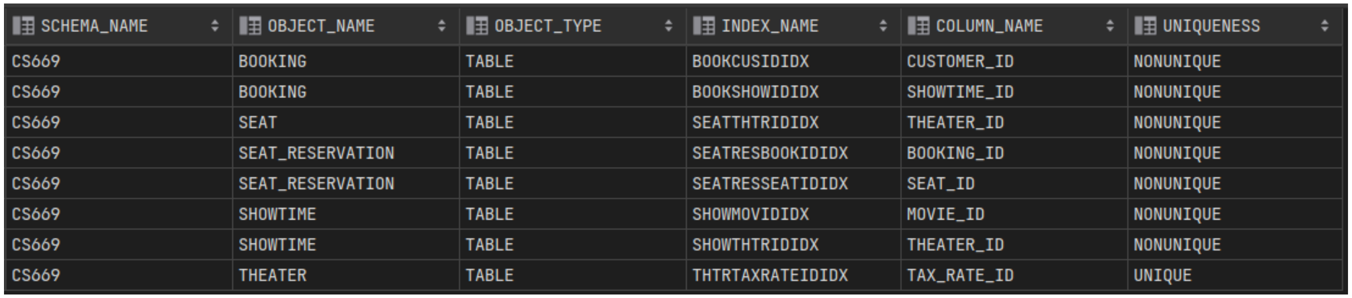


SCHEMA_NAME	TABLE_NAME	COLUMN_NAME	POSITION	CONSTRAINT_TYPE	STATUS	INDEX_NAME
CS669	ACCESSIBLE_SEAT	SEAT_ID	1	P	ENABLED	SYS_C0029160
CS669	BOOKING	BOOKING_ID	1	P	ENABLED	SYS_C0029182
CS669	BOX_SEAT	SEAT_ID	1	P	ENABLED	SYS_C0029166
CS669	CUSTOMER	CUSTOMER_ID	1	P	ENABLED	SYS_C0029137
CS669	MOVIE	MOVIE_ID	1	P	ENABLED	SYS_C0029132
CS669	NORMAL_SEAT	SEAT_ID	1	P	ENABLED	SYS_C0029157
CS669	RECLINING_SEAT	SEAT_ID	1	P	ENABLED	SYS_C0029163
CS669	SEAT	SEAT_ID	1	P	ENABLED	SYS_C0029154
CS669	SEAT_RESERVATION	SEAT_RES_ID	1	P	ENABLED	SYS_C0029186
CS669	SHOWTIME	SHOWTIME_ID	1	P	ENABLED	SYS_C0029175
CS669	TAX_RATES	TAX_RATE_ID	1	P	ENABLED	SYS_C0029142
CS669	THEATER	THEATER_ID	1	P	ENABLED	SYS_C0029149

- Foreign Key Indexes – As far as foreign keys, all of them will need an index.
Below is a table identifying each foreign key column, whether the index should be unique or not, and why:

Column	Unique?	Description
BOOKING.SHOWTIME_ID	Non-unique	Showtime entries in the Booking table are non-unique since there can be multiple showtime entries
BOOKING.CUSTOMER_ID	Non-unique	There can be multiple customers entries
SEAT.THEATER_ID	Non-unique	Theatre entries will repeat for each seat
SEAT_RESERVATION.SEAT_ID, SHOWTIME_ID	Unique	The same showtime cannot have the same seats booked - so unique
SEAT_RESERVATION.BOOKING_ID	Non-unique	A booking can have multiple seats
SHOWTIME.THEATER_ID	Non-unique	Theatre entries will repeat for showtimes in them
SHOWTIME.MOVIE_ID	Non-unique	A movie can be shown at multiple showtimes
THEATER.TAX_RATE_ID	Non-unique	There could be multiple theater entries for the same state

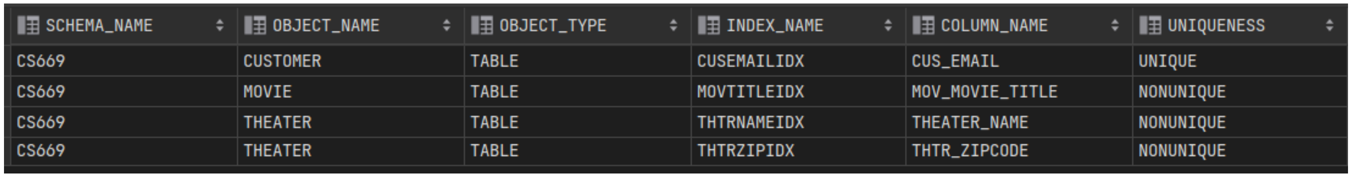
Please note that there is no need to create indexes on the subtype foreign keys (SeatID) since it's already indexed implicitly as the PK.



SCHEMA_NAME	OBJECT_NAME	OBJECT_TYPE	INDEX_NAME	COLUMN_NAME	UNIQUENESS
CS669	BOOKING	TABLE	BOOKCUSIDIDX	CUSTOMER_ID	NONUNIQUE
CS669	BOOKING	TABLE	BOOKSHOWIDIDX	SHOWTIME_ID	NONUNIQUE
CS669	SEAT	TABLE	SEATTHTRIDIDX	THEATER_ID	NONUNIQUE
CS669	SEAT_RESERVATION	TABLE	SEATRESBOOKIDIDX	BOOKING_ID	NONUNIQUE
CS669	SEAT_RESERVATION	TABLE	SEATRESSEATIDIDX	SEAT_ID	NONUNIQUE
CS669	SHOWTIME	TABLE	SHOWMOVIDIDX	MOVIE_ID	NONUNIQUE
CS669	SHOWTIME	TABLE	SHOWTHTRIDIDX	THEATER_ID	NONUNIQUE
CS669	THEATER	TABLE	THTRTAXRATEIDIDX	TAX_RATE_ID	UNIQUE

- Query driven Indexes – Here are the query driven indexes I was able to identify based on the use-cases for my app and visualizing how they would translate to queries by user interaction:

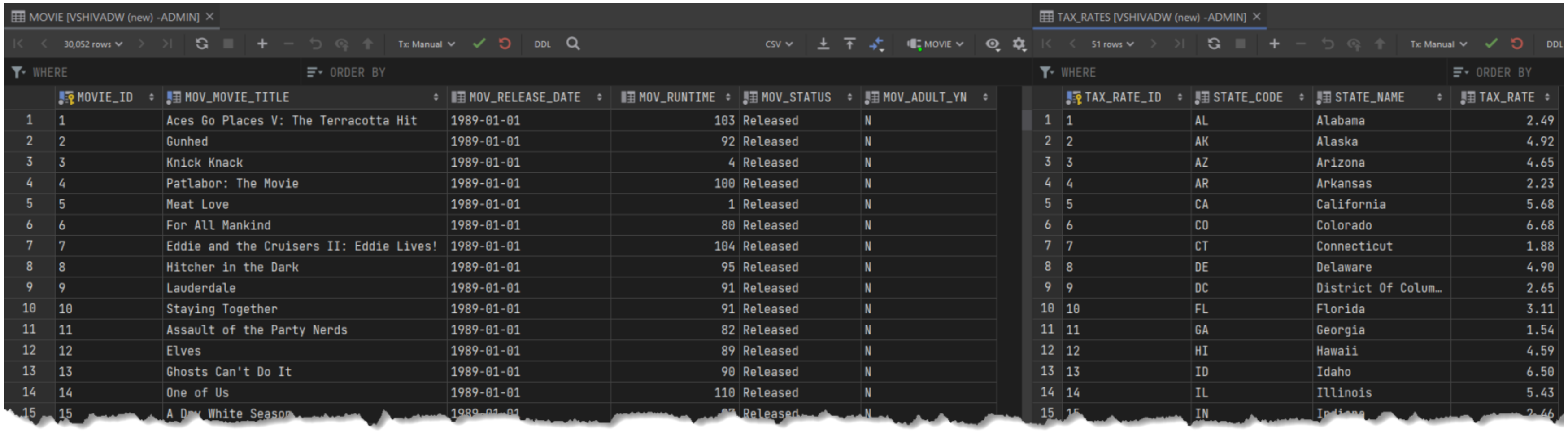
Column	Unique?	Description
CUSTOMER.CUS_EMAIL	Unique	A business rule states that the same email address cannot be used more than once to sign up
MOVIE.MOV_MOVIE_TITLE	Non-unique	Customers usually lookup movies on the app
THEATER.THEATER_NAME	Non-unique	There are chances that customers will look for a particular theater if they live in the area or liked their previous experience
THEATER.THEATER_ZIPCODE	Non-unique	Based on the customer's location, showtimes at that zipcode are shown on the app – this is a frequently used field to filter results



SCHEMA_NAME	OBJECT_NAME	OBJECT_TYPE	INDEX_NAME	COLUMN_NAME	UNIQUENESS
CS669	CUSTOMER	TABLE	CUSEMAILIDX	CUS_EMAIL	UNIQUE
CS669	MOVIE	TABLE	MOVTITLEIDX	MOV_MOVIE_TITLE	NONUNIQUE
CS669	THEATER	TABLE	THTRNAMEIDX	THEATER_NAME	NONUNIQUE
CS669	THEATER	TABLE	THTRZIPIDX	THTR_ZIPCODE	NONUNIQUE

Stored Procs to insert data

In this iteration (5), I'm writing reusable stored procedures to insert data into my database.
I start by inserting some raw data into MOVIE and TAX_RATE tables:



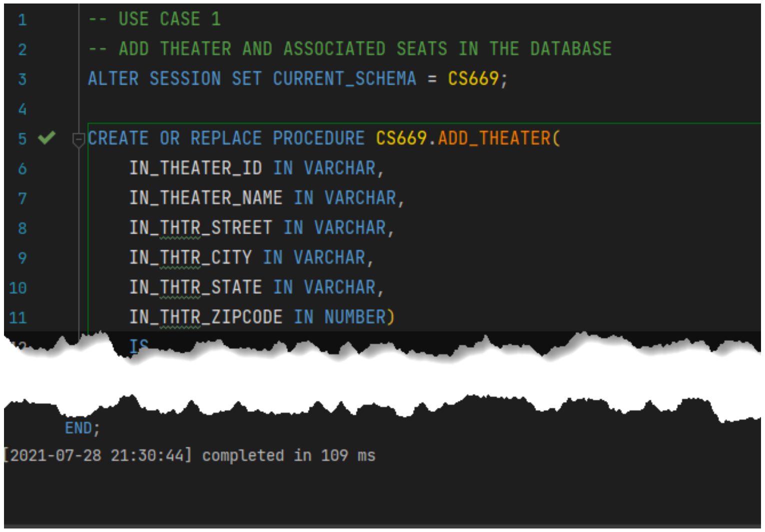
	MOVIE_ID	MOV_MOVIE_TITLE	MOV_RELEASE_DATE	MOV_RUNTIME	MOV_STATUS	MOV_ADULT_YN
1	1	Aces Go Places V: The Terracotta Hit	1989-01-01	103	Released	N
2	2	Gunhed	1989-01-01	92	Released	N
3	3	Knick Knack	1989-01-01	4	Released	N
4	4	Patlabor: The Movie	1989-01-01	100	Released	N
5	5	Meat Love	1989-01-01	1	Released	N
6	6	For All Mankind	1989-01-01	80	Released	N
7	7	Eddie and the Cruisers II: Eddie Lives!	1989-01-01	104	Released	N
8	8	Hitcher in the Dark	1989-01-01	95	Released	N
9	9	Lauderdale	1989-01-01	91	Released	N
10	10	Staying Together	1989-01-01	91	Released	N
11	11	Assault of the Party Nerds	1989-01-01	82	Released	N
12	12	Elves	1989-01-01	89	Released	N
13	13	Ghosts Can't Do It	1989-01-01	90	Released	N
14	14	One of Us	1989-01-01	110	Released	N
15	15	A Dry White Season	1989-01-01	97	Released	N

	TAX_RATE_ID	STATE_CODE	STATE_NAME	TAX_RATE
1	1	AL	Alabama	2.49
2	2	AK	Alaska	4.92
3	3	AZ	Arizona	4.65
4	4	AR	Arkansas	2.23
5	5	CA	California	5.68
6	6	CO	Colorado	6.68
7	7	CT	Connecticut	1.88
8	8	DE	Delaware	4.90
9	9	DC	District Of Colum…	2.65
10	10	FL	Florida	3.11
11	11	GA	Georgia	1.54
12	12	HI	Hawaii	4.59
13	13	ID	Idaho	6.50
14	14	IL	Illinois	5.43
15	15	IN	Indiana	2.46

<u>Use Case 1</u>

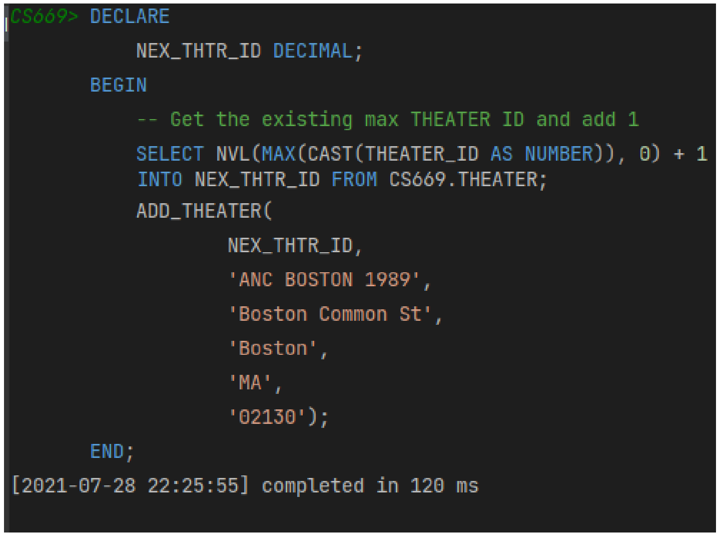
Inserting data into THEATER; also insert into SEAT for that theater

A new theater has been licensed to show movies, and the app admin needs to make a new entry into the database, with details of the same, including the details of the seats in the theater. For the sake of simplicity, I assume that the new theaters being inserted don't have any specialized seats – accessible/box/reclining.

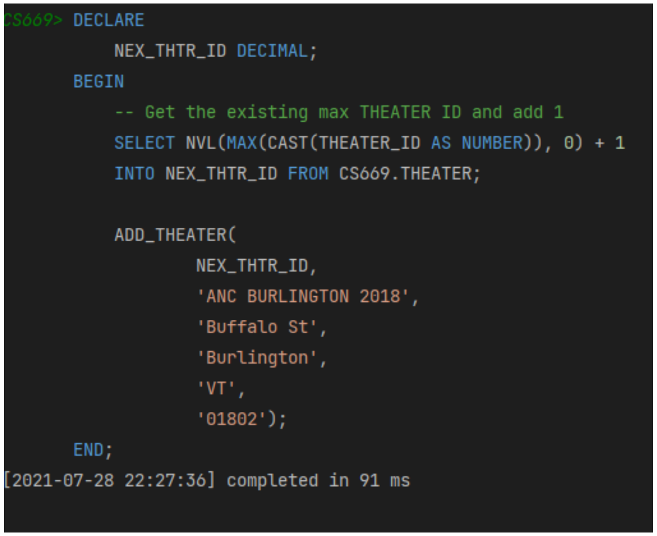


```
-- USE CASE 1
-- ADD THEATER AND ASSOCIATED SEATS IN THE DATABASE
ALTER SESSION SET CURRENT_SCHEMA = CS669;

CREATE OR REPLACE PROCEDURE CS669.ADD_THEATER(
    IN_THEATER_ID IN VARCHAR,
    IN_THEATER_NAME IN VARCHAR,
    IN_THTR_STREET IN VARCHAR,
    IN_THTR_CITY IN VARCHAR,
    IN_THTR_STATE IN VARCHAR,
    IN_THTR_ZIPCODE IN NUMBER)
    IS
```

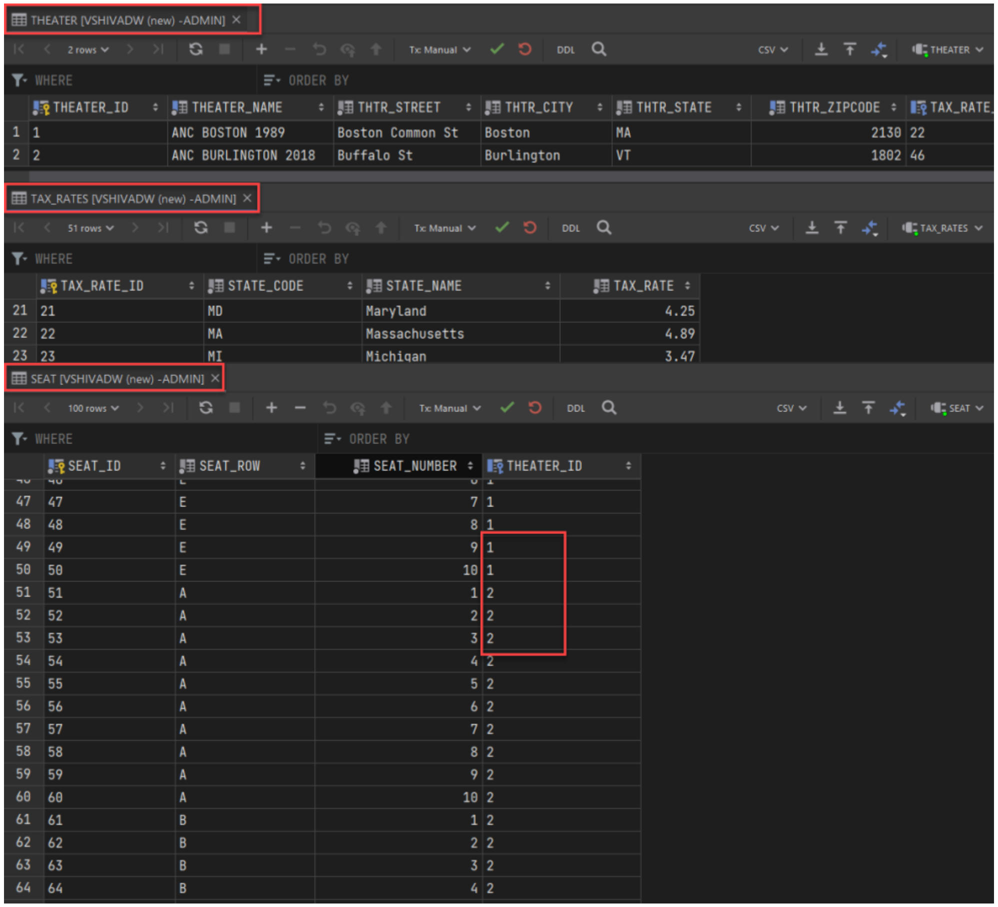
```
      END;
[2021-07-28 21:30:44] completed in 109 ms
```



```
CS669> DECLARE
           NEX_THTR_ID DECIMAL;
       BEGIN
           -- Get the existing max THEATER ID and add 1
           SELECT NVL(MAX(CAST(THEATER_ID AS NUMBER)), 0) + 1
           INTO NEX_THTR_ID FROM CS669.THEATER;
           ADD_THEATER(
                   NEX_THTR_ID,
                   'ANC BOSTON 1989',
                   'Boston Common St',
                   'Boston',
                   'MA',
                   '02130');
       END;
[2021-07-28 22:25:55] completed in 120 ms
```



```
CS669> DECLARE
           NEX_THTR_ID DECIMAL;
       BEGIN
           -- Get the existing max THEATER ID and add 1
           SELECT NVL(MAX(CAST(THEATER_ID AS NUMBER)), 0) + 1
           INTO NEX_THTR_ID FROM CS669.THEATER;

           ADD_THEATER(
                   NEX_THTR_ID,
                   'ANC BURLINGTON 2018',
                   'Buffalo St',
                   'Burlington',
                   'VT',
                   '01802');
       END;
[2021-07-28 22:27:36] completed in 91 ms
```



THEATER [VSHIVADW (new) -ADMIN]

	THEATER_ID	THEATER_NAME	THTR_STREET	THTR_CITY	THTR_STATE	THTR_ZIPCODE	TAX_RATE_
1	1	ANC BOSTON 1989	Boston Common St	Boston	MA	2130	22
2	2	ANC BURLINGTON 2018	Buffalo St	Burlington	VT	1802	46

TAX_RATES [VSHIVADW (new) -ADMIN]

	TAX_RATE_ID	STATE_CODE	STATE_NAME	TAX_RATE
21	21	MD	Maryland	4.25
22	22	MA	Massachusetts	4.89
23	23	MI	Michigan	3.47

SEAT [VSHIVADW (new) -ADMIN]

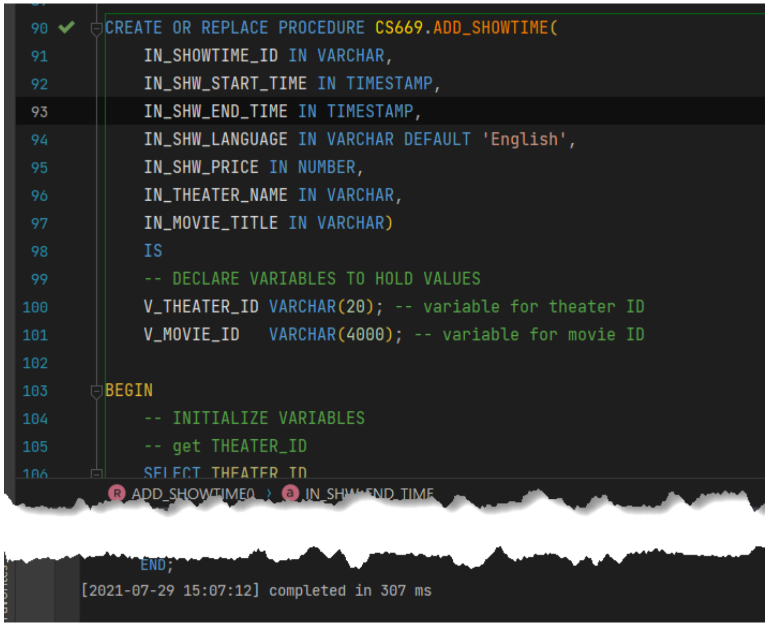
	SEAT_ID	SEAT_ROW	SEAT_NUMBER	THEATER_ID
47	47	E	7	1
48	48	E	8	1
49	49	E	9	1
50	50	E	10	1
51	51	A	1	2
52	52	A	2	2
53	53	A	3	2
54	54	A	4	2
55	55	A	5	2
56	56	A	6	2
57	57	A	7	2
58	58	A	8	2
59	59	A	9	2
60	60	A	10	2
61	61	B	1	2
62	62	B	2	2
63	63	B	3	2
64	64	B	4	2

Use Case 2

Inserting data into SHOWTIME for a movie that's playing in a theater.

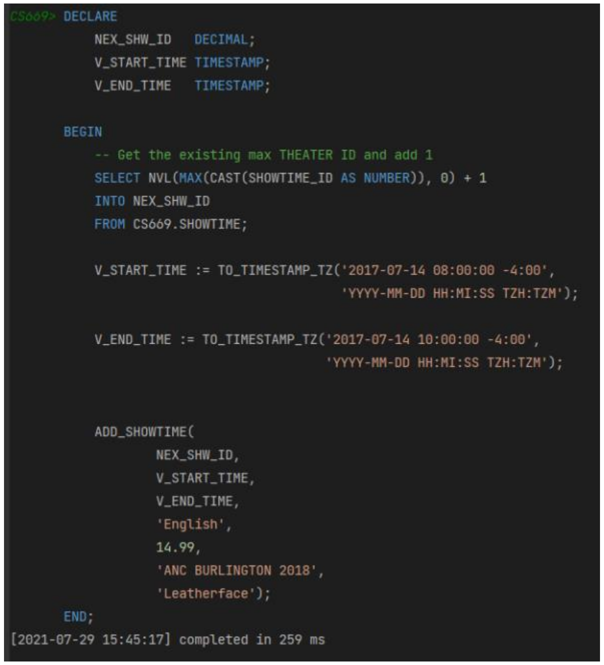
The theater admin, based on the movies that are released, updates showtimes on the app.
This would be available on the SHOWTIME table.

1. START_DATE and END_DATE are programmatically derived from START_TIME and END_TIME.
   Each entry signifies that day's start and end times
   It is assumed that if data exists on this table, the showtime is active – so the admin would also have to remove inactive showtimes.
   The removal is not in scope for this use case.

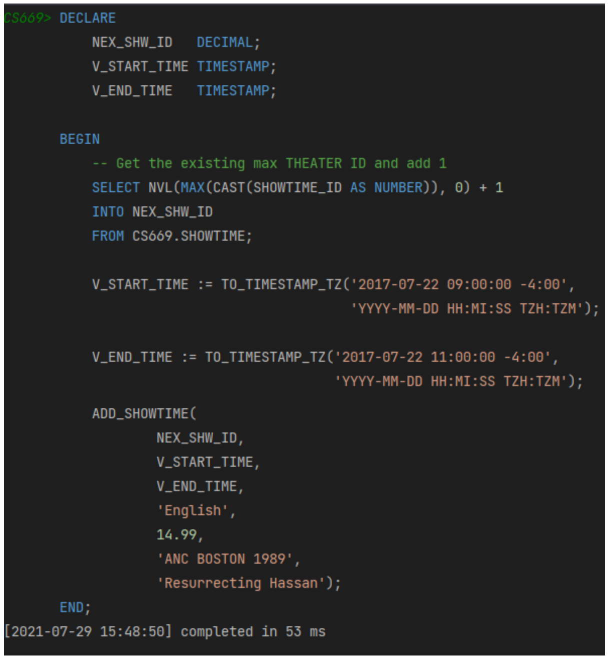


```
CREATE OR REPLACE PROCEDURE CS669.ADD_SHOWTIME(
    IN_SHOWTIME_ID IN VARCHAR,
    IN_SHW_START_TIME IN TIMESTAMP,
    IN_SHW_END_TIME IN TIMESTAMP,
    IN_SHW_LANGUAGE IN VARCHAR DEFAULT 'English',
    IN_SHW_PRICE IN NUMBER,
    IN_THEATER_NAME IN VARCHAR,
    IN_MOVIE_TITLE IN VARCHAR)
    IS
    -- DECLARE VARIABLES TO HOLD VALUES
    V_THEATER_ID VARCHAR(20); -- variable for theater ID
    V_MOVIE_ID   VARCHAR(4000); -- variable for movie ID

BEGIN
    -- INITIALIZE VARIABLES
    -- get THEATER_ID
    SELECT THEATER_ID
```

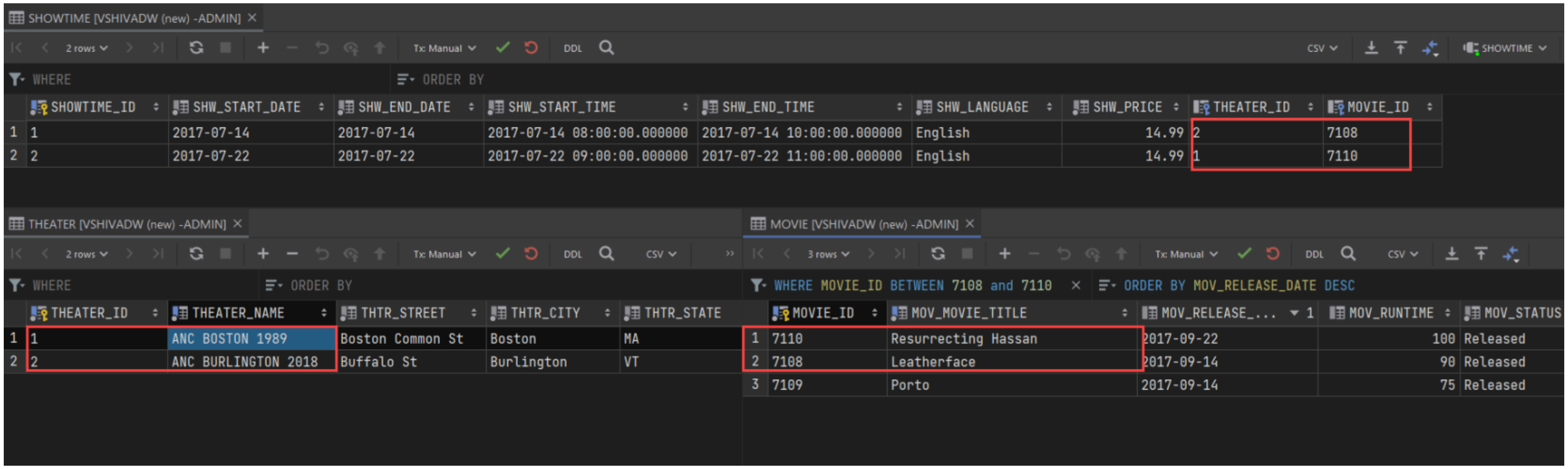
```
    END;
[2021-07-29 15:07:12] completed in 307 ms
```



```
CS669> DECLARE
           NEX_SHW_ID   DECIMAL;
           V_START_TIME TIMESTAMP;
           V_END_TIME   TIMESTAMP;

       BEGIN
           -- Get the existing max THEATER ID and add 1
           SELECT NVL(MAX(CAST(SHOWTIME_ID AS NUMBER)), 0) + 1
           INTO NEX_SHW_ID
           FROM CS669.SHOWTIME;

           V_START_TIME := TO_TIMESTAMP_TZ('2017-07-14 08:00:00 -4:00',
                                           'YYYY-MM-DD HH:MI:SS TZH:TZM');

           V_END_TIME := TO_TIMESTAMP_TZ('2017-07-14 10:00:00 -4:00',
                                         'YYYY-MM-DD HH:MI:SS TZH:TZM');


           ADD_SHOWTIME(
                   NEX_SHW_ID,
                   V_START_TIME,
                   V_END_TIME,
                   'English',
                   14.99,
                   'ANC BURLINGTON 2018',
                   'Leatherface');
       END;
[2021-07-29 15:45:17] completed in 259 ms
```



```
CS669> DECLARE
           NEX_SHW_ID   DECIMAL;
           V_START_TIME TIMESTAMP;
           V_END_TIME   TIMESTAMP;

       BEGIN
           -- Get the existing max THEATER ID and add 1
           SELECT NVL(MAX(CAST(SHOWTIME_ID AS NUMBER)), 0) + 1
           INTO NEX_SHW_ID
           FROM CS669.SHOWTIME;

           V_START_TIME := TO_TIMESTAMP_TZ('2017-07-22 09:00:00 -4:00',
                                           'YYYY-MM-DD HH:MI:SS TZH:TZM');

           V_END_TIME := TO_TIMESTAMP_TZ('2017-07-22 11:00:00 -4:00',
                                         'YYYY-MM-DD HH:MI:SS TZH:TZM');

           ADD_SHOWTIME(
                   NEX_SHW_ID,
                   V_START_TIME,
                   V_END_TIME,
                   'English',
                   14.99,
                   'ANC BOSTON 1989',
                   'Resurrecting Hassan');
       END;
[2021-07-29 15:48:50] completed in 53 ms
```



SHOWTIME [VSHIVADW (new) -ADMIN]

	SHOWTIME_ID	SHW_START_DATE	SHW_END_DATE	SHW_START_TIME	SHW_END_TIME	SHW_LANGUAGE	SHW_PRICE	THEATER_ID	MOVIE_ID
1	1	2017-07-14	2017-07-14	2017-07-14 08:00:00.000000	2017-07-14 10:00:00.000000	English	14.99	2	7108
2	2	2017-07-22	2017-07-22	2017-07-22 09:00:00.000000	2017-07-22 11:00:00.000000	English	14.99	1	7110

THEATER [VSHIVADW (new) -ADMIN]

	THEATER_ID	THEATER_NAME	THTR_STREET	THTR_CITY	THTR_STATE
1	1	ANC BOSTON 1989	Boston Common St	Boston	MA
2	2	ANC BURLINGTON 2018	Buffalo St	Burlington	VT

MOVIE [VSHIVADW (new) -ADMIN]

WHERE MOVIE_ID BETWEEN 7108 and 7110

ORDER BY MOV_RELEASE_DATE DESC

	MOVIE_ID	MOV_MOVIE_TITLE	MOV_RELEASE_...	MOV_RUNTIME	MOV_STATUS
1	7110	Resurrecting Hassan	2017-09-22	100	Released
2	7108	Leatherface	2017-09-14	90	Released
3	7109	Porto	2017-09-14	75	Released

Use Case 3

Customer makes a booking, reserves a seat for a movie showtime at a theater

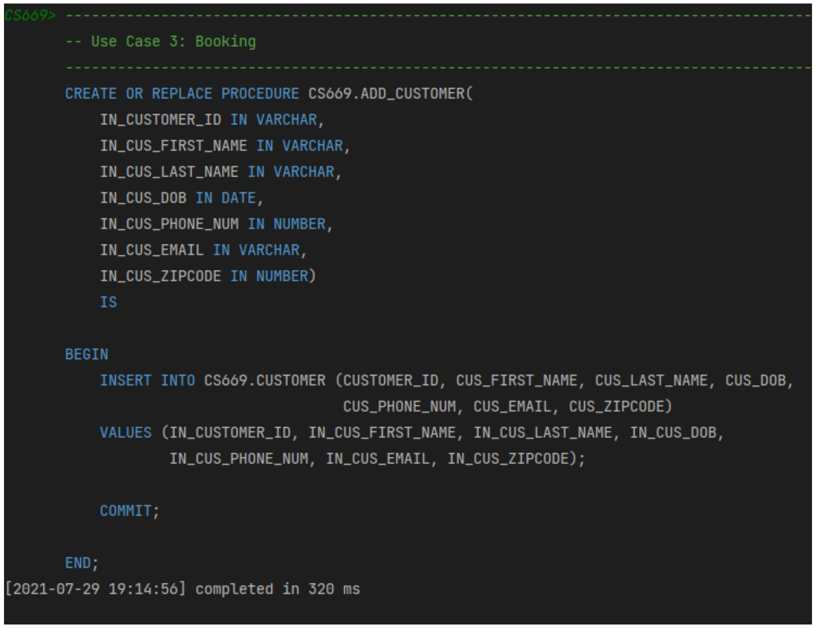
This use case has two PL/SQL procs since it's more of a process unlike the above one-off insert use cases

1. Through the booking process, my app captures customer information and places it in CUSTOMER table – ADD_CUSTOMER
2. MAKE_BOOKING then captures customer, showtime & seat information and inserts into BOOKING and SEAT_RESERVATION

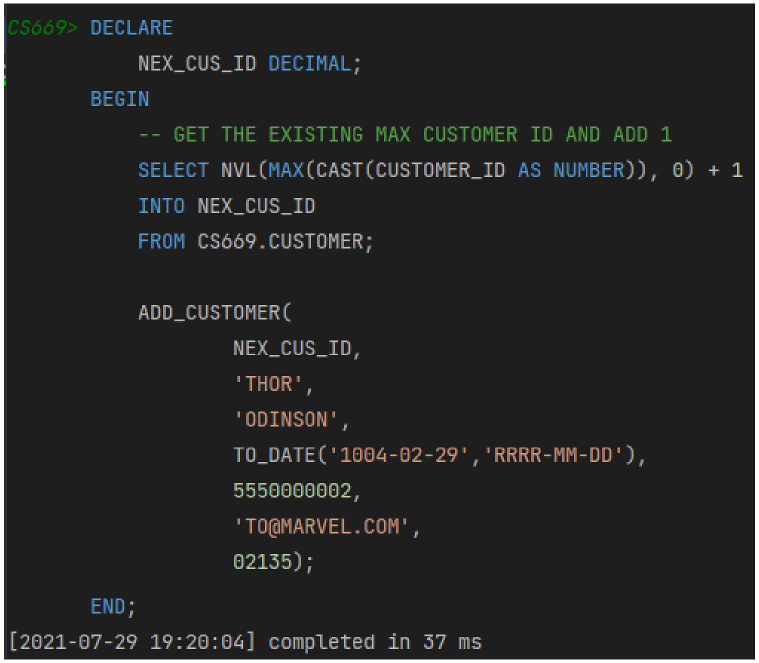
This use case also assumes that if there is an active booking, the table SEAT_RESERVATION will have entries for the seats booked. If the booking gets cancelled, the entries would be deleted.

Deletions and historical data capture are not in scope for this use case

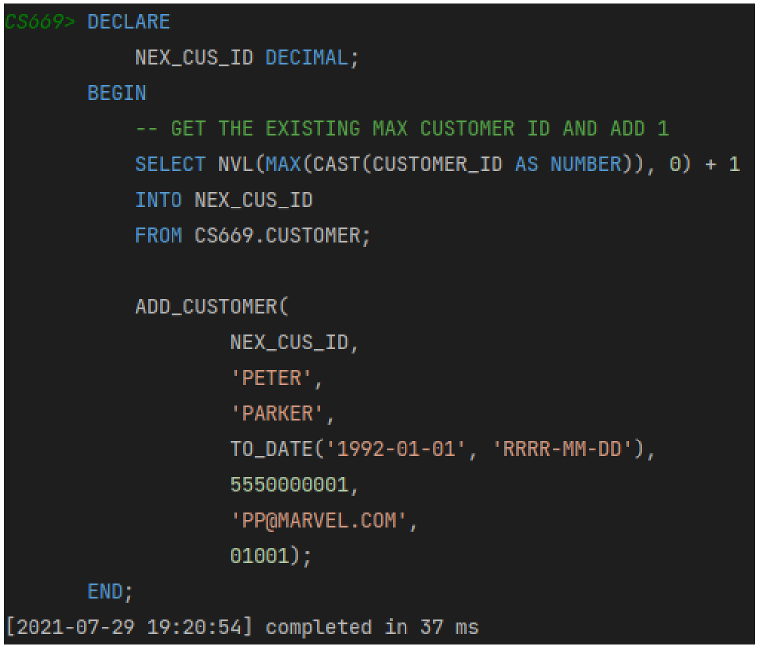
ADD_CUSTOMER:



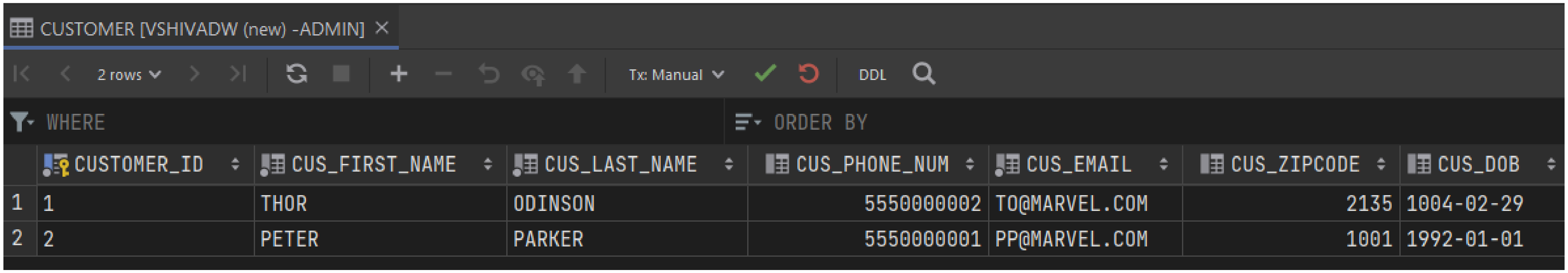
```
CS669> ------------------------------------------------------------------------------------------
       -- Use Case 3: Booking
       ------------------------------------------------------------------------------------------
       CREATE OR REPLACE PROCEDURE CS669.ADD_CUSTOMER(
           IN_CUSTOMER_ID IN VARCHAR,
           IN_CUS_FIRST_NAME IN VARCHAR,
           IN_CUS_LAST_NAME IN VARCHAR,
           IN_CUS_DOB IN DATE,
           IN_CUS_PHONE_NUM IN NUMBER,
           IN_CUS_EMAIL IN VARCHAR,
           IN_CUS_ZIPCODE IN NUMBER)
           IS

       BEGIN
           INSERT INTO CS669.CUSTOMER (CUSTOMER_ID, CUS_FIRST_NAME, CUS_LAST_NAME, CUS_DOB,
                                       CUS_PHONE_NUM, CUS_EMAIL, CUS_ZIPCODE)
           VALUES (IN_CUSTOMER_ID, IN_CUS_FIRST_NAME, IN_CUS_LAST_NAME, IN_CUS_DOB,
                   IN_CUS_PHONE_NUM, IN_CUS_EMAIL, IN_CUS_ZIPCODE);

           COMMIT;

       END;
[2021-07-29 19:14:56] completed in 320 ms
```



```
CS669> DECLARE
           NEX_CUS_ID DECIMAL;
       BEGIN
           -- GET THE EXISTING MAX CUSTOMER ID AND ADD 1
           SELECT NVL(MAX(CAST(CUSTOMER_ID AS NUMBER)), 0) + 1
           INTO NEX_CUS_ID
           FROM CS669.CUSTOMER;

           ADD_CUSTOMER(
                   NEX_CUS_ID,
                   'THOR',
                   'ODINSON',
                   TO_DATE('1004-02-29','RRRR-MM-DD'),
                   5550000002,
                   'TO@MARVEL.COM',
                   02135);

       END;
[2021-07-29 19:20:04] completed in 37 ms
```

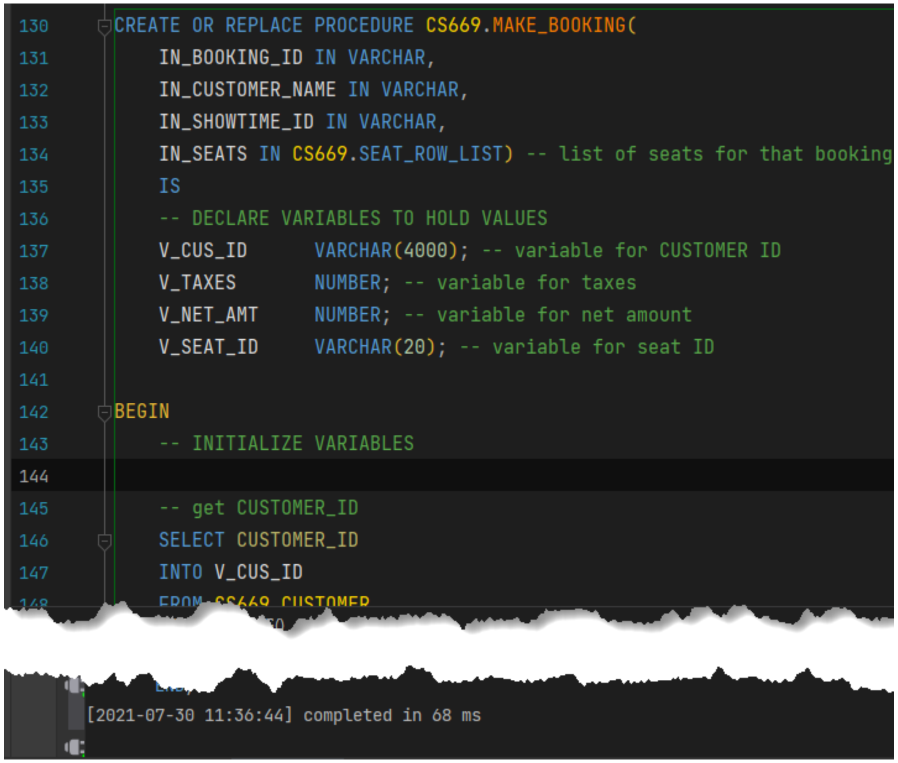


```
CS669> DECLARE
           NEX_CUS_ID DECIMAL;
       BEGIN
           -- GET THE EXISTING MAX CUSTOMER ID AND ADD 1
           SELECT NVL(MAX(CAST(CUSTOMER_ID AS NUMBER)), 0) + 1
           INTO NEX_CUS_ID
           FROM CS669.CUSTOMER;

           ADD_CUSTOMER(
                   NEX_CUS_ID,
                   'PETER',
                   'PARKER',
                   TO_DATE('1992-01-01', 'RRRR-MM-DD'),
                   5550000001,
                   'PP@MARVEL.COM',
                   01001);
       END;
[2021-07-29 19:20:54] completed in 37 ms
```



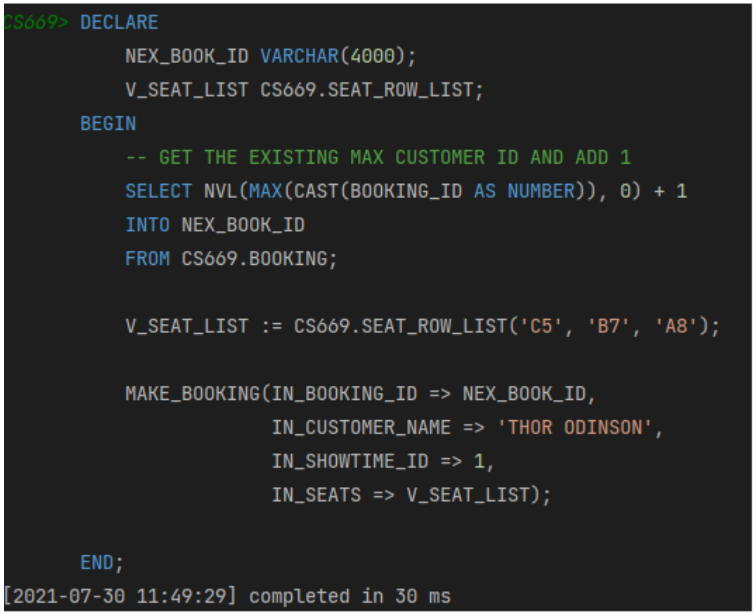
	CUSTOMER_ID	CUS_FIRST_NAME	CUS_LAST_NAME	CUS_PHONE_NUM	CUS_EMAIL	CUS_ZIPCODE	CUS_DOB
1	1	THOR	ODINSON	5550000002	TO@MARVEL.COM	2135	1004-02-29
2	2	PETER	PARKER	5550000001	PP@MARVEL.COM	1001	1992-01-01

MAKE_BOOKING:

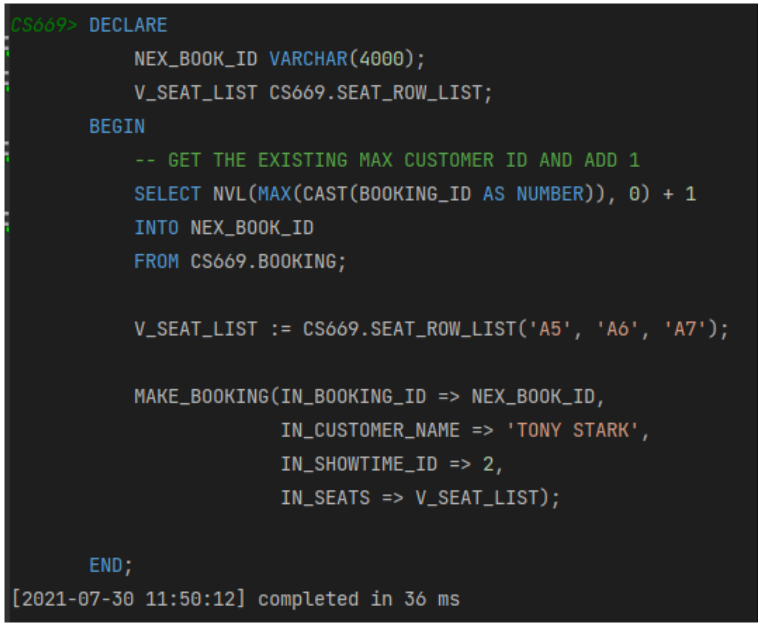


```
CREATE OR REPLACE PROCEDURE CS669.MAKE_BOOKING(
    IN_BOOKING_ID IN VARCHAR,
    IN_CUSTOMER_NAME IN VARCHAR,
    IN_SHOWTIME_ID IN VARCHAR,
    IN_SEATS IN CS669.SEAT_ROW_LIST) -- list of seats for that booking
    IS
    -- DECLARE VARIABLES TO HOLD VALUES
    V_CUS_ID      VARCHAR(4000); -- variable for CUSTOMER ID
    V_TAXES       NUMBER; -- variable for taxes
    V_NET_AMT     NUMBER; -- variable for net amount
    V_SEAT_ID     VARCHAR(20); -- variable for seat ID

BEGIN
    -- INITIALIZE VARIABLES

    -- get CUSTOMER_ID
    SELECT CUSTOMER_ID
    INTO V_CUS_ID
    FROM CS669.CUSTOMER
```

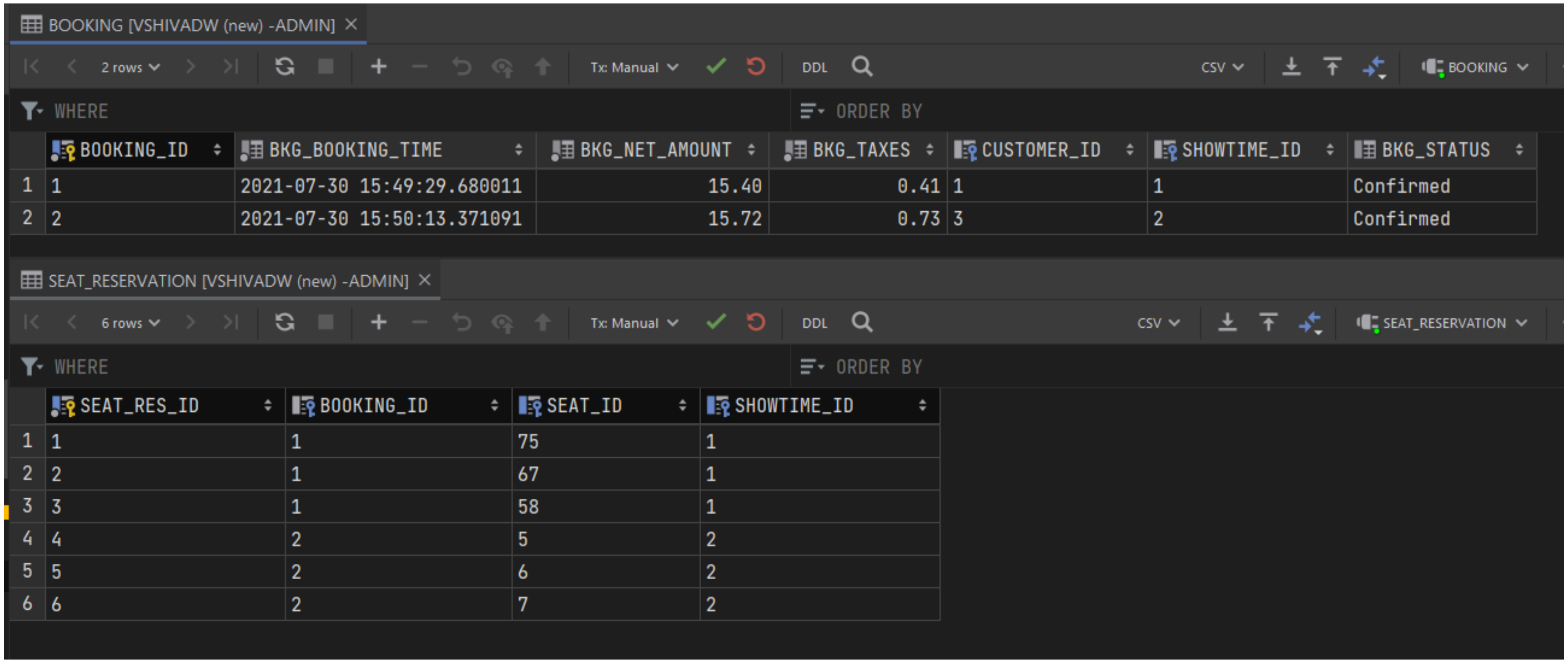
```
[2021-07-30 11:36:44] completed in 68 ms
```



```
CS669> DECLARE
           NEX_BOOK_ID VARCHAR(4000);
           V_SEAT_LIST CS669.SEAT_ROW_LIST;
       BEGIN
           -- GET THE EXISTING MAX CUSTOMER ID AND ADD 1
           SELECT NVL(MAX(CAST(BOOKING_ID AS NUMBER)), 0) + 1
           INTO NEX_BOOK_ID
           FROM CS669.BOOKING;

           V_SEAT_LIST := CS669.SEAT_ROW_LIST('C5', 'B7', 'A8');

           MAKE_BOOKING(IN_BOOKING_ID => NEX_BOOK_ID,
                        IN_CUSTOMER_NAME => 'THOR ODINSON',
                        IN_SHOWTIME_ID => 1,
                        IN_SEATS => V_SEAT_LIST);

       END;
[2021-07-30 11:49:29] completed in 30 ms
```



```
CS669> DECLARE
           NEX_BOOK_ID VARCHAR(4000);
           V_SEAT_LIST CS669.SEAT_ROW_LIST;
       BEGIN
           -- GET THE EXISTING MAX CUSTOMER ID AND ADD 1
           SELECT NVL(MAX(CAST(BOOKING_ID AS NUMBER)), 0) + 1
           INTO NEX_BOOK_ID
           FROM CS669.BOOKING;

           V_SEAT_LIST := CS669.SEAT_ROW_LIST('A5', 'A6', 'A7');

           MAKE_BOOKING(IN_BOOKING_ID => NEX_BOOK_ID,
                        IN_CUSTOMER_NAME => 'TONY STARK',
                        IN_SHOWTIME_ID => 2,
                        IN_SEATS => V_SEAT_LIST);

       END;
[2021-07-30 11:50:12] completed in 36 ms
```



	BOOKING_ID	BKG_BOOKING_TIME	BKG_NET_AMOUNT	BKG_TAXES	CUSTOMER_ID	SHOWTIME_ID	BKG_STATUS
1	1	2021-07-30 15:49:29.680011	15.40	0.41	1	1	Confirmed
2	2	2021-07-30 15:50:13.371091	15.72	0.73	3	2	Confirmed

	SEAT_RES_ID	BOOKING_ID	SEAT_ID	SHOWTIME_ID
1	1	1	75	1
2	2	1	67	1
3	3	1	58	1
4	4	2	5	2
5	5	2	6	2
6	6	2	7	2

Triggers

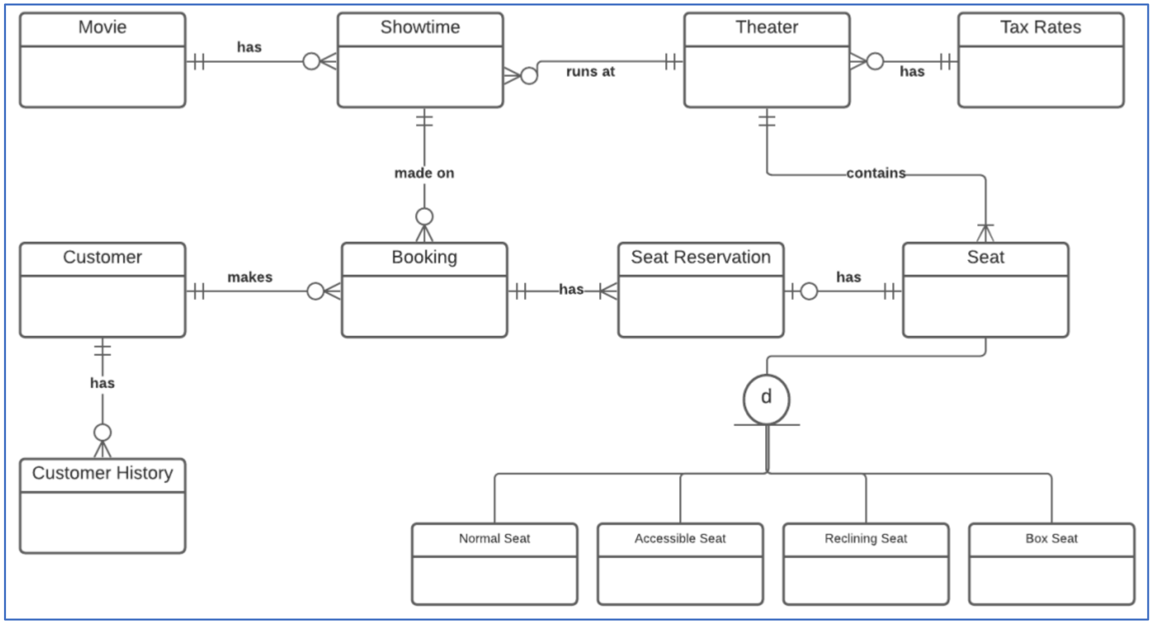
Maintain Historical Data - Customer

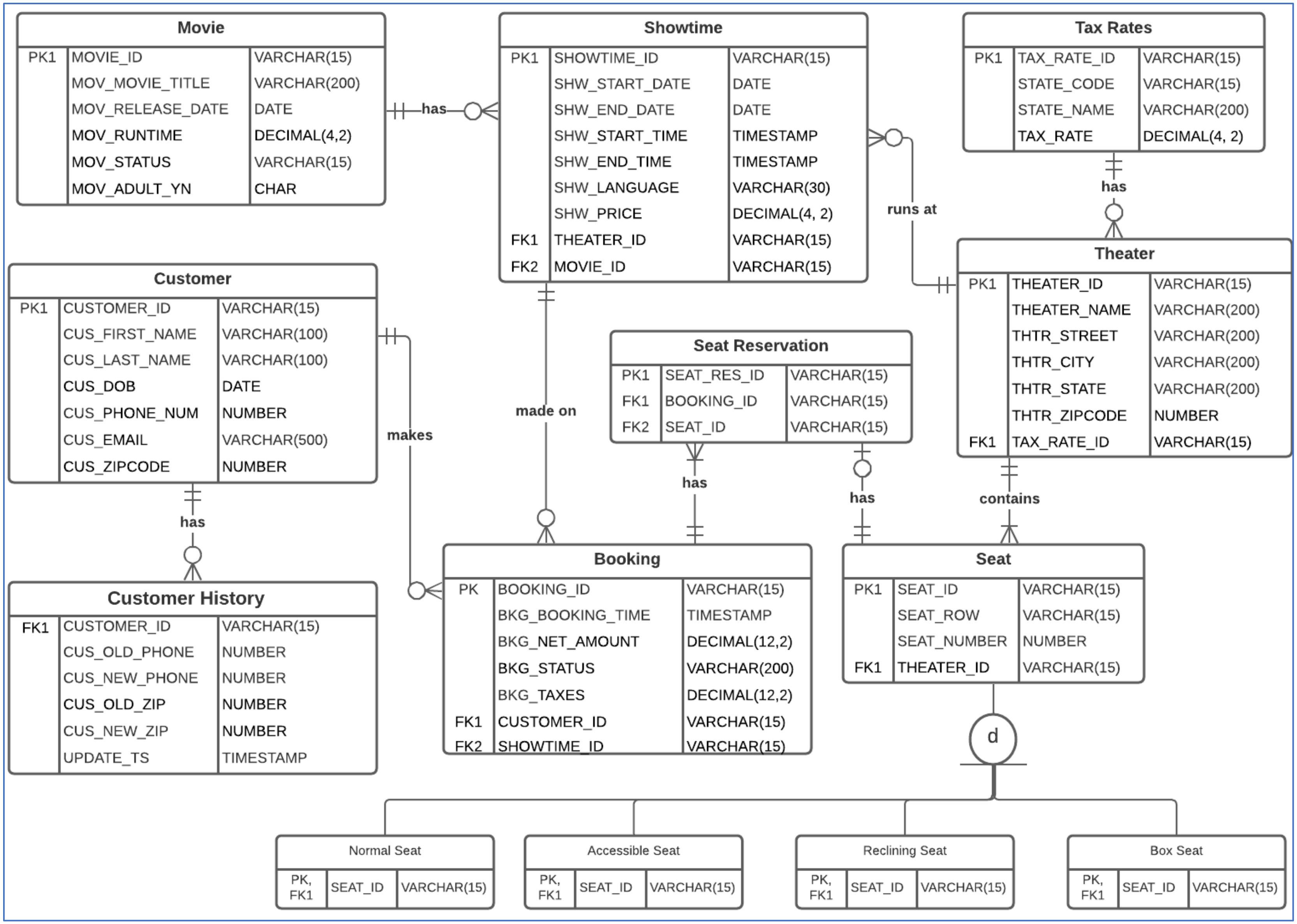
Most important aspects of my app and database are to keep a history of the bookings made and customer information. There could be others – like tracking past showtimes or movie statuses, but for now, I'm keeping a history of only these two important tables.
As for Booking history, the main table Booking itself contains the current and history – if a customer wants to lookup past bookings, it can be retrieved from the same table.
With that, here are is the new table CUSTOMER_HISTORY:

Table: **Customer History**

Field	What it Stores
CUSTOMER_ID	Foreign key to the unique customer ID in CUSTOMER
CUS_OLD_PHONE	First Name of the Customer
CUS_NEW_PHONE	Last Name of the Customer
CUS_OLD_ZIP	Zip code of the customer
CUS_OLD_ZIP	Operation made – INSERT/UPDATE
UPDATE_TS	Timestamp of the operation

And here are the new ERDs:


Movie
has
Showtime
runs at
Theater
has
Tax Rates
made on
contains
Customer
makes
Booking
has
Seat Reservation
has
Seat
has
d
Customer History
Normal Seat
Accessible Seat
Reclining Seat
Box Seat



Movie

PK1	MOVIE_ID	VARCHAR(15)
	MOV_MOVIE_TITLE	VARCHAR(200)
	MOV_RELEASE_DATE	DATE
	MOV_RUNTIME	DECIMAL(4,2)
	MOV_STATUS	VARCHAR(15)
	MOV_ADULT_YN	CHAR

Showtime

PK1	SHOWTIME_ID	VARCHAR(15)
	SHW_START_DATE	DATE
	SHW_END_DATE	DATE
	SHW_START_TIME	TIMESTAMP
	SHW_END_TIME	TIMESTAMP
	SHW_LANGUAGE	VARCHAR(30)
	SHW_PRICE	DECIMAL(4, 2)
FK1	THEATER_ID	VARCHAR(15)
FK2	MOVIE_ID	VARCHAR(15)

Tax Rates

PK1	TAX_RATE_ID	VARCHAR(15)
	STATE_CODE	VARCHAR(15)
	STATE_NAME	VARCHAR(200)
	TAX_RATE	DECIMAL(4, 2)

Customer

PK1	CUSTOMER_ID	VARCHAR(15)
	CUS_FIRST_NAME	VARCHAR(100)
	CUS_LAST_NAME	VARCHAR(100)
	CUS_DOB	DATE
	CUS_PHONE_NUM	NUMBER
	CUS_EMAIL	VARCHAR(500)
	CUS_ZIPCODE	NUMBER

Seat Reservation

PK1	SEAT_RES_ID	VARCHAR(15)
FK1	BOOKING_ID	VARCHAR(15)
FK2	SEAT_ID	VARCHAR(15)

Theater

PK1	THEATER_ID	VARCHAR(15)
	THEATER_NAME	VARCHAR(200)
	THTR_STREET	VARCHAR(200)
	THTR_CITY	VARCHAR(200)
	THTR_STATE	VARCHAR(200)
	THTR_ZIPCODE	NUMBER
FK1	TAX_RATE_ID	VARCHAR(15)

Customer History

FK1	CUSTOMER_ID	VARCHAR(15)
	CUS_OLD_PHONE	NUMBER
	CUS_NEW_PHONE	NUMBER
	CUS_OLD_ZIP	NUMBER
	CUS_NEW_ZIP	NUMBER
	UPDATE_TS	TIMESTAMP

Booking

PK	BOOKING_ID	VARCHAR(15)
	BKG_BOOKING_TIME	TIMESTAMP
	BKG_NET_AMOUNT	DECIMAL(12,2)
	BKG_STATUS	VARCHAR(200)
	BKG_TAXES	DECIMAL(12,2)
FK1	CUSTOMER_ID	VARCHAR(15)
FK2	SHOWTIME_ID	VARCHAR(15)

Seat

PK1	SEAT_ID	VARCHAR(15)
	SEAT_ROW	VARCHAR(15)
	SEAT_NUMBER	NUMBER
FK1	THEATER_ID	VARCHAR(15)

Normal Seat

PK, FK1	SEAT_ID	VARCHAR(15)

Accessible Seat

PK, FK1	SEAT_ID	VARCHAR(15)

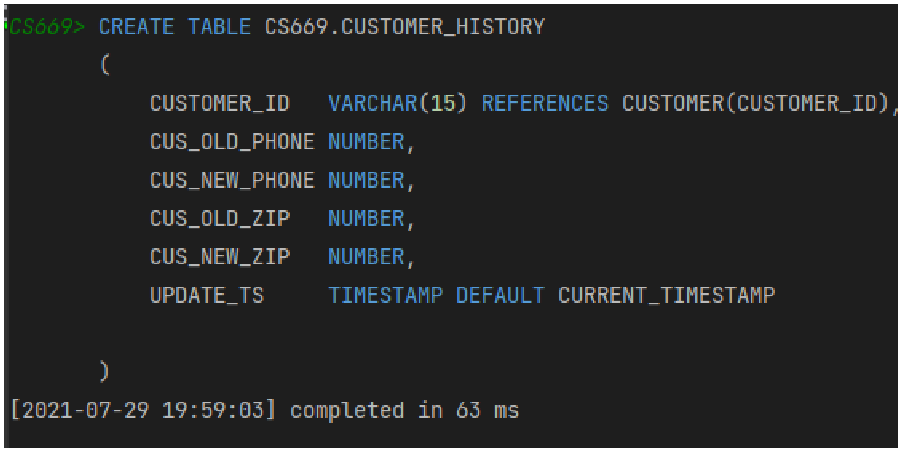
Reclining Seat

PK, FK1	SEAT_ID	VARCHAR(15)

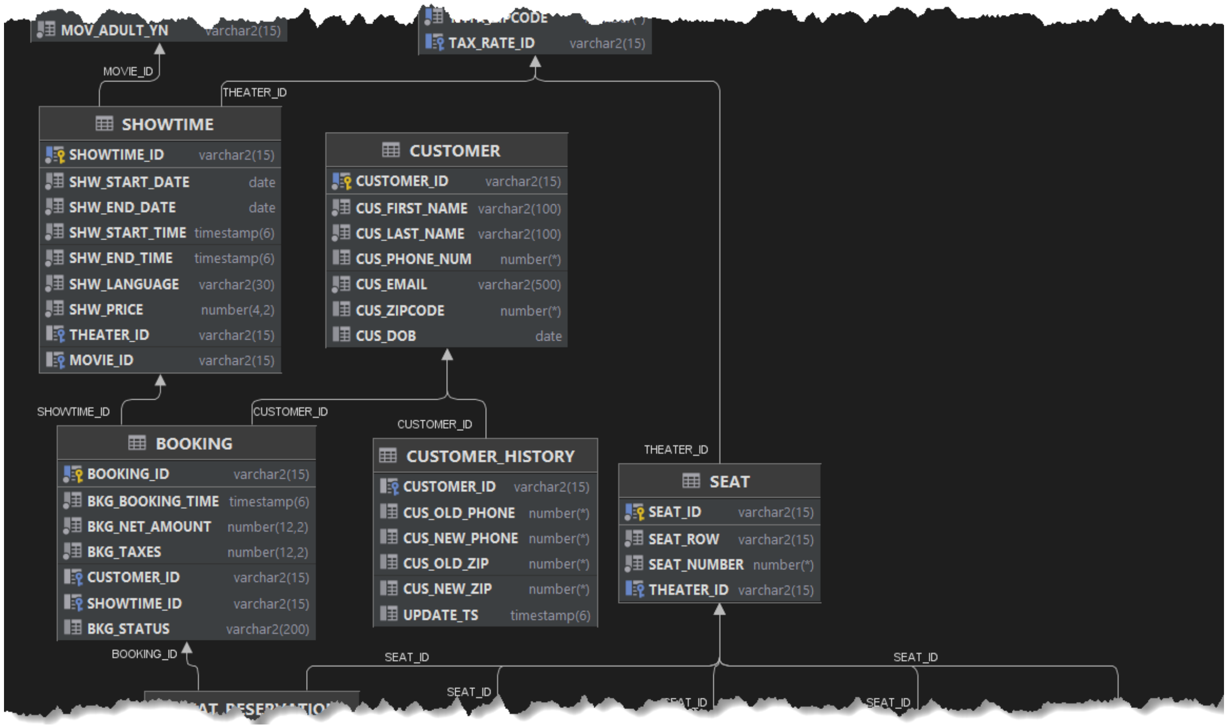
Box Seat

PK, FK1	SEAT_ID	VARCHAR(15)

Here are the screenshots of the table creation, updated physical diagram from the database:



```
CS669> CREATE TABLE CS669.CUSTOMER_HISTORY
       (
           CUSTOMER_ID   VARCHAR(15) REFERENCES CUSTOMER(CUSTOMER_ID),
           CUS_OLD_PHONE NUMBER,
           CUS_NEW_PHONE NUMBER,
           CUS_OLD_ZIP   NUMBER,
           CUS_NEW_ZIP   NUMBER,
           UPDATE_TS     TIMESTAMP DEFAULT CURRENT_TIMESTAMP

       )
[2021-07-29 19:59:03] completed in 63 ms
```



SHOWTIME	
SHOWTIME_ID	varchar2(15)
SHW_START_DATE	date
SHW_END_DATE	date
SHW_START_TIME	timestamp(6)
SHW_END_TIME	timestamp(6)
SHW_LANGUAGE	varchar2(30)
SHW_PRICE	number(4,2)
THEATER_ID	varchar2(15)
MOVIE_ID	varchar2(15)

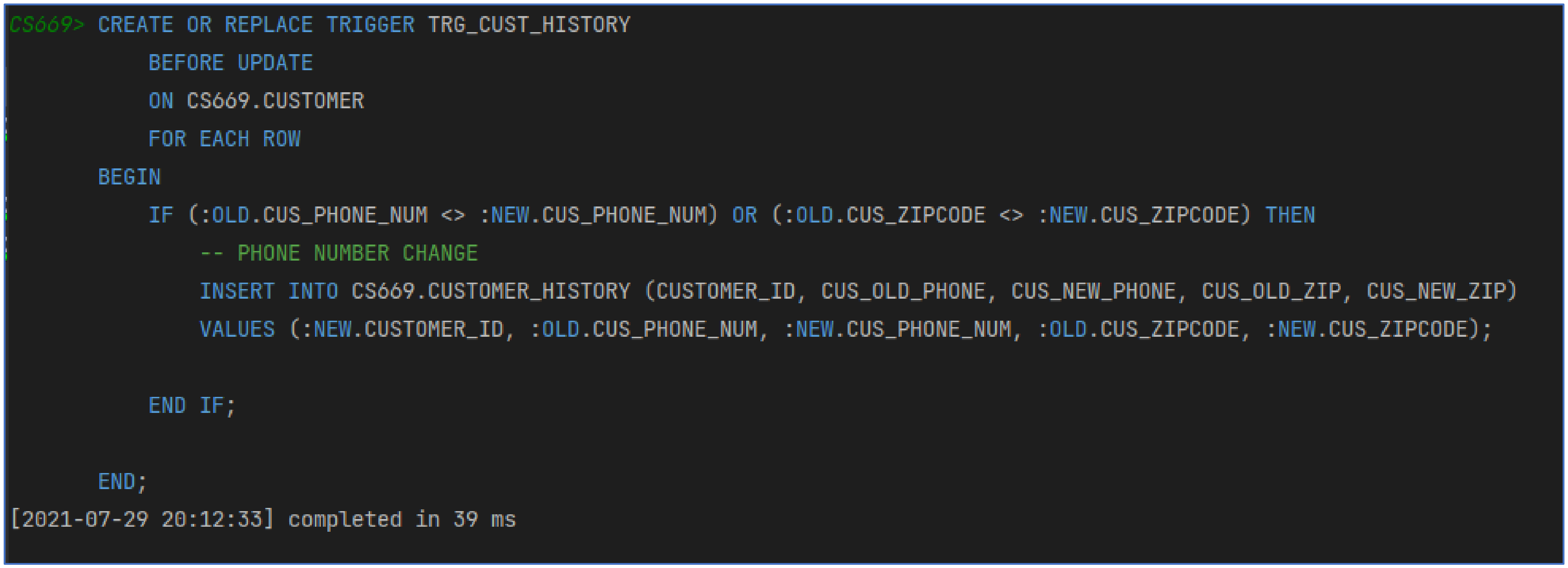
CUSTOMER	
CUSTOMER_ID	varchar2(15)
CUS_FIRST_NAME	varchar2(100)
CUS_LAST_NAME	varchar2(100)
CUS_PHONE_NUM	number(*)
CUS_EMAIL	varchar2(500)
CUS_ZIPCODE	number(*)
CUS_DOB	date

BOOKING	
BOOKING_ID	varchar2(15)
BKG_BOOKING_TIME	timestamp(6)
BKG_NET_AMOUNT	number(12,2)
BKG_TAXES	number(12,2)
CUSTOMER_ID	varchar2(15)
SHOWTIME_ID	varchar2(15)
BKG_STATUS	varchar2(200)

CUSTOMER_HISTORY	
CUSTOMER_ID	varchar2(15)
CUS_OLD_PHONE	number(*)
CUS_NEW_PHONE	number(*)
CUS_OLD_ZIP	number(*)
CUS_NEW_ZIP	number(*)
UPDATE_TS	timestamp(6)

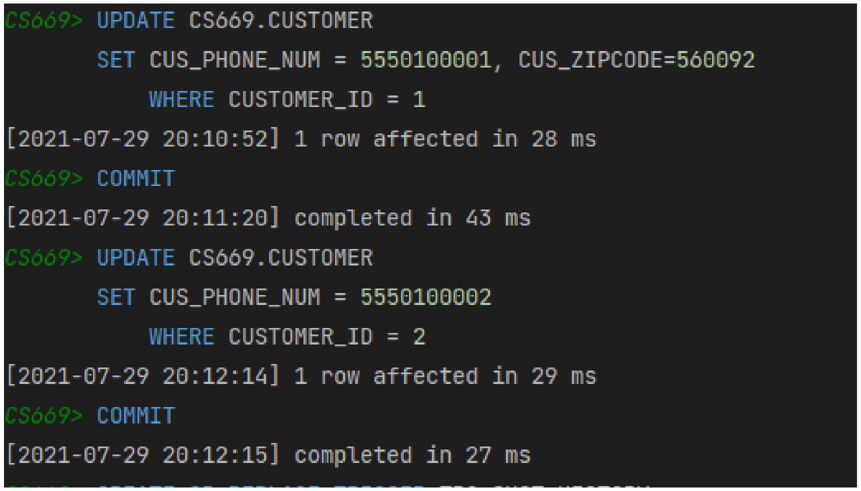
SEAT	
SEAT_ID	varchar2(15)
SEAT_ROW	varchar2(15)
SEAT_NUMBER	number(*)
THEATER_ID	varchar2(15)

Here is the trigger that will track changes to phone number and zip code:

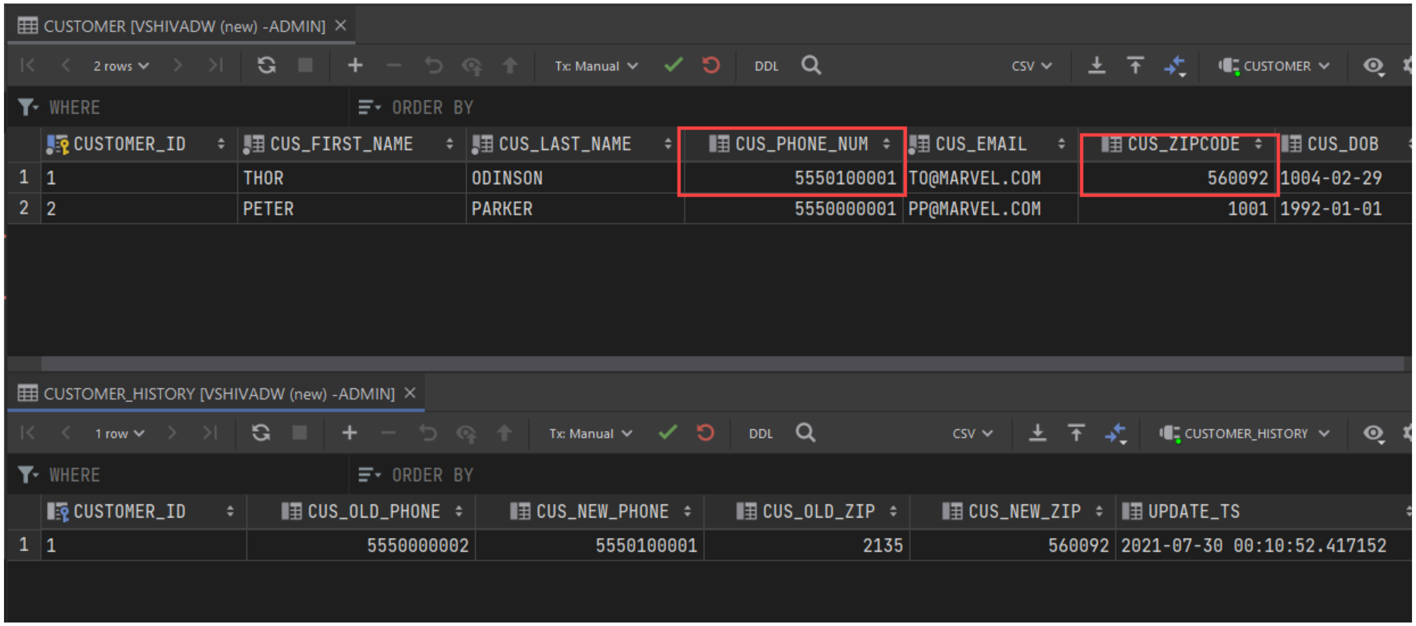


```
CS669> CREATE OR REPLACE TRIGGER TRG_CUST_HISTORY
           BEFORE UPDATE
           ON CS669.CUSTOMER
           FOR EACH ROW
       BEGIN
           IF (:OLD.CUS_PHONE_NUM <> :NEW.CUS_PHONE_NUM) OR (:OLD.CUS_ZIPCODE <> :NEW.CUS_ZIPCODE) THEN
               -- PHONE NUMBER CHANGE
               INSERT INTO CS669.CUSTOMER_HISTORY (CUSTOMER_ID, CUS_OLD_PHONE, CUS_NEW_PHONE, CUS_OLD_ZIP, CUS_NEW_ZIP)
               VALUES (:NEW.CUSTOMER_ID, :OLD.CUS_PHONE_NUM, :NEW.CUS_PHONE_NUM, :OLD.CUS_ZIPCODE, :NEW.CUS_ZIPCODE);

           END IF;

       END;
[2021-07-29 20:12:33] completed in 39 ms
```

A couple of updates to check:



```
CS669> UPDATE CS669.CUSTOMER
       SET CUS_PHONE_NUM = 5550100001, CUS_ZIPCODE=560092
           WHERE CUSTOMER_ID = 1
[2021-07-29 20:10:52] 1 row affected in 28 ms
CS669> COMMIT
[2021-07-29 20:11:20] completed in 43 ms
CS669> UPDATE CS669.CUSTOMER
       SET CUS_PHONE_NUM = 5550100002
           WHERE CUSTOMER_ID = 2
[2021-07-29 20:12:14] 1 row affected in 29 ms
CS669> COMMIT
[2021-07-29 20:12:15] completed in 27 ms
```

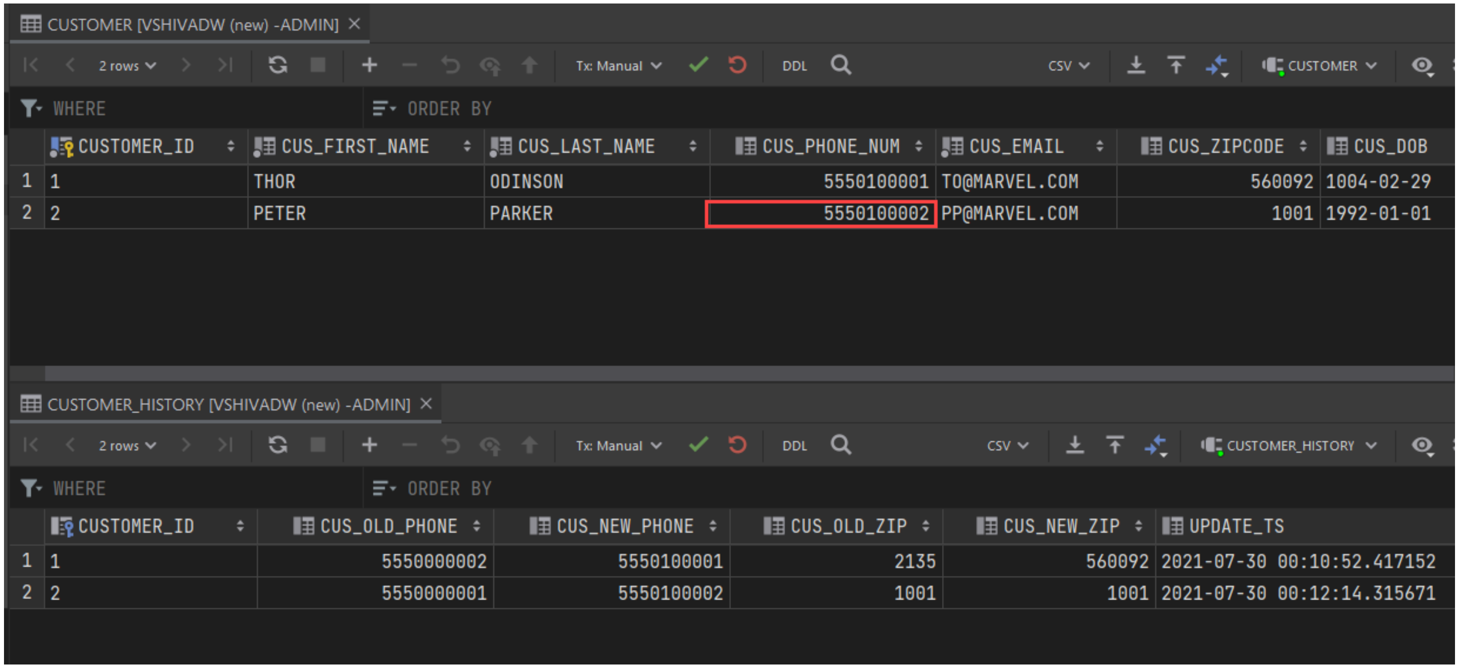


CUSTOMER [VSHIVADW (new) -ADMIN]

	CUSTOMER_ID	CUS_FIRST_NAME	CUS_LAST_NAME	CUS_PHONE_NUM	CUS_EMAIL	CUS_ZIPCODE	CUS_DOB
1	1	THOR	ODINSON	5550100001	TO@MARVEL.COM	560092	1004-02-29
2	2	PETER	PARKER	5550000001	PP@MARVEL.COM	1001	1992-01-01

CUSTOMER_HISTORY [VSHIVADW (new) -ADMIN]

	CUSTOMER_ID	CUS_OLD_PHONE	CUS_NEW_PHONE	CUS_OLD_ZIP	CUS_NEW_ZIP	UPDATE_TS
1	1	5550000002	5550100001	2135	560092	2021-07-30 00:10:52.417152



CUSTOMER [VSHIVADW (new) -ADMIN]

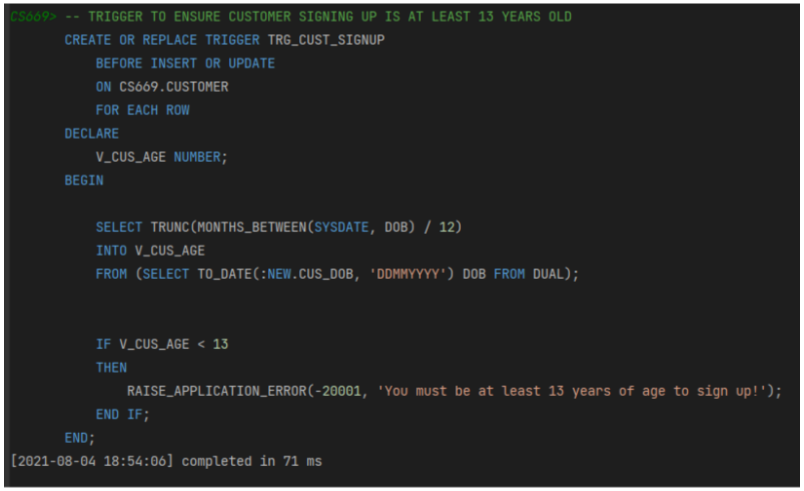
	CUSTOMER_ID	CUS_FIRST_NAME	CUS_LAST_NAME	CUS_PHONE_NUM	CUS_EMAIL	CUS_ZIPCODE	CUS_DOB
1	1	THOR	ODINSON	5550100001	TO@MARVEL.COM	560092	1004-02-29
2	2	PETER	PARKER	5550100002	PP@MARVEL.COM	1001	1992-01-01

CUSTOMER_HISTORY [VSHIVADW (new) -ADMIN]

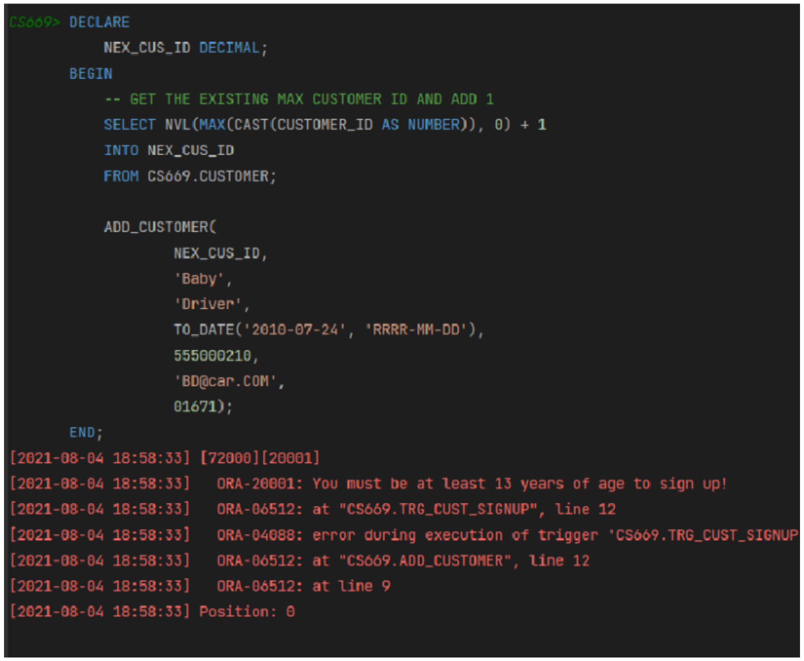
	CUSTOMER_ID	CUS_OLD_PHONE	CUS_NEW_PHONE	CUS_OLD_ZIP	CUS_NEW_ZIP	UPDATE_TS
1	1	5550000002	5550100001	2135	560092	2021-07-30 00:10:52.417152
2	2	5550000001	5550100002	1001	1001	2021-07-30 00:12:14.315671

Additional Triggers for Business Rules

1. Customer must be at least 13 years of age to sign up:

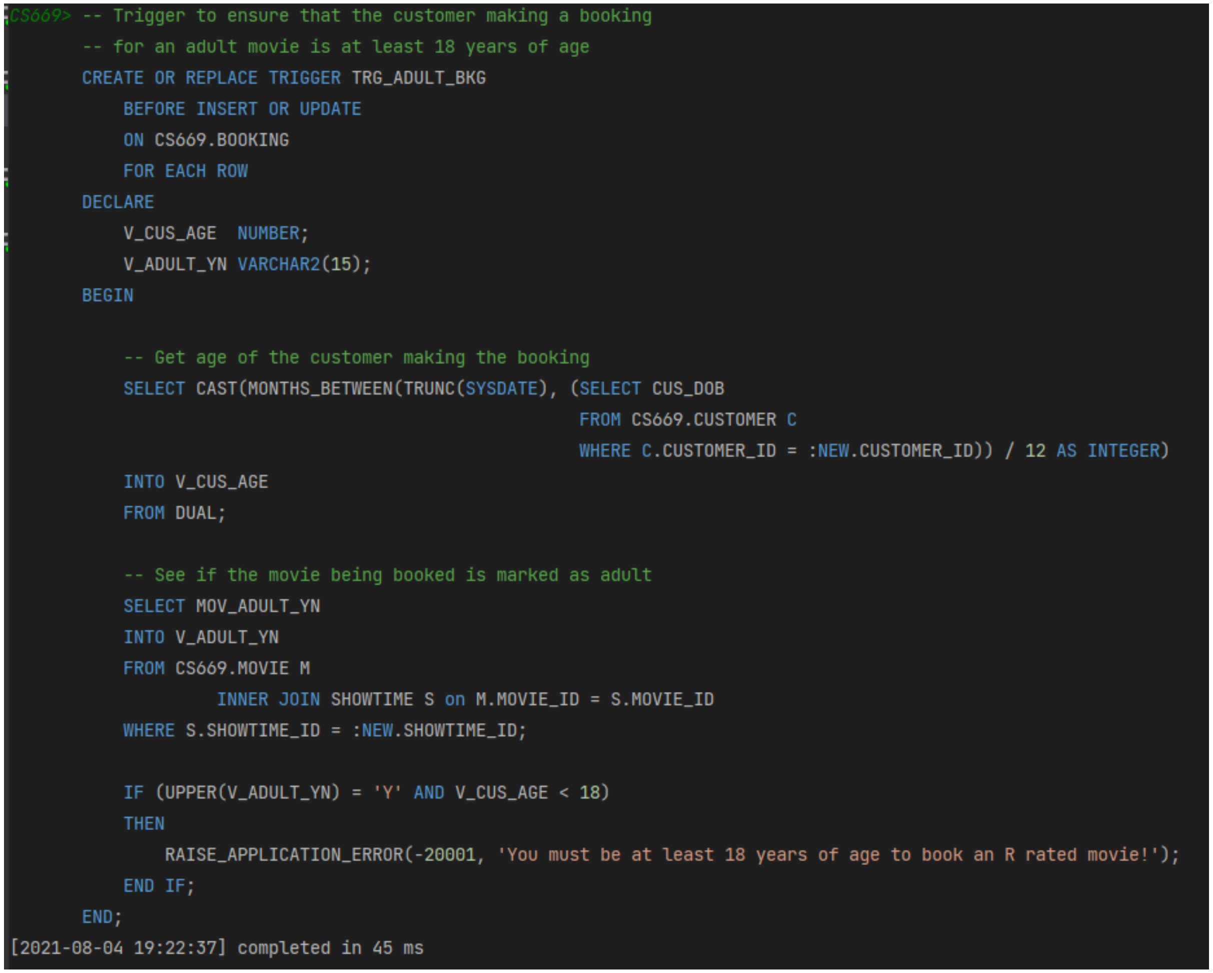


```
CS669> -- TRIGGER TO ENSURE CUSTOMER SIGNING UP IS AT LEAST 13 YEARS OLD
       CREATE OR REPLACE TRIGGER TRG_CUST_SIGNUP
           BEFORE INSERT OR UPDATE
           ON CS669.CUSTOMER
           FOR EACH ROW
       DECLARE
           V_CUS_AGE NUMBER;
       BEGIN

           SELECT TRUNC(MONTHS_BETWEEN(SYSDATE, DOB) / 12)
           INTO V_CUS_AGE
           FROM (SELECT TO_DATE(:NEW.CUS_DOB, 'DDMMYYYY') DOB FROM DUAL);


           IF V_CUS_AGE < 13
           THEN
               RAISE_APPLICATION_ERROR(-20001, 'You must be at least 13 years of age to sign up!');
           END IF;
       END;
[2021-08-04 18:54:06] completed in 71 ms
```

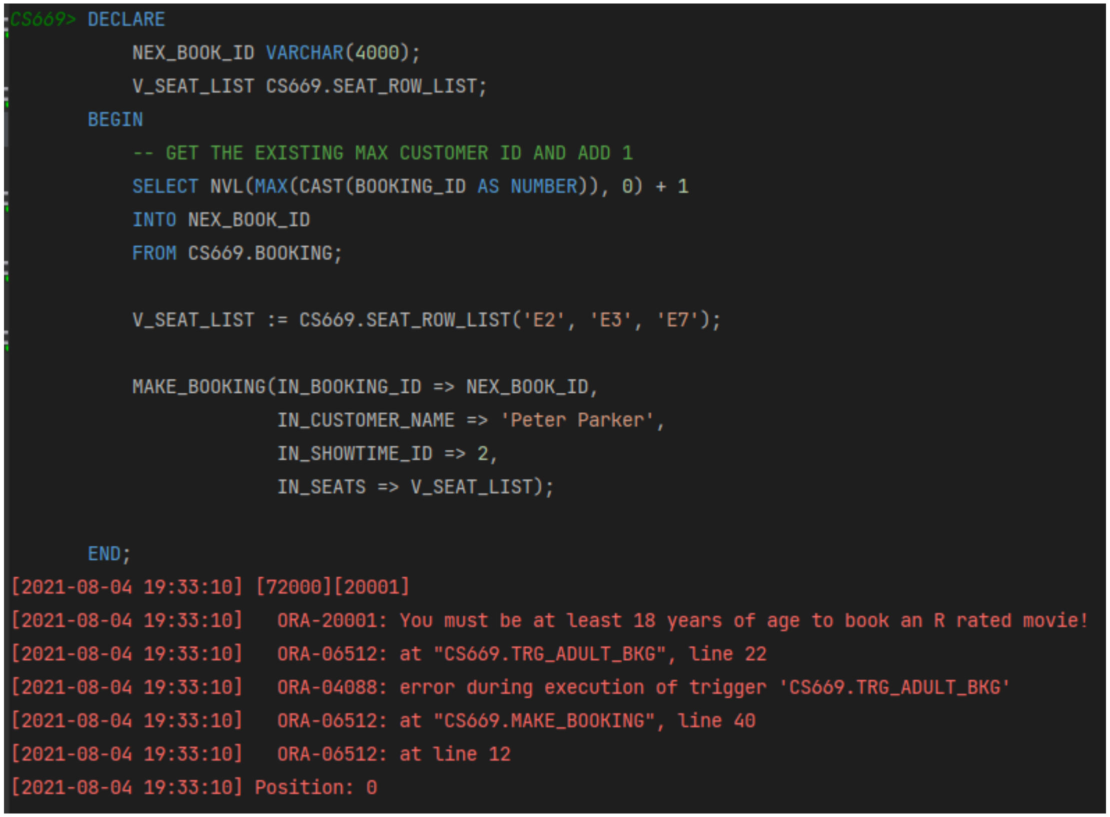


```
CS669> DECLARE
           NEX_CUS_ID DECIMAL;
       BEGIN
           -- GET THE EXISTING MAX CUSTOMER ID AND ADD 1
           SELECT NVL(MAX(CAST(CUSTOMER_ID AS NUMBER)), 0) + 1
           INTO NEX_CUS_ID
           FROM CS669.CUSTOMER;

           ADD_CUSTOMER(
                   NEX_CUS_ID,
                   'Baby',
                   'Driver',
                   TO_DATE('2010-07-24', 'RRRR-MM-DD'),
                   555000210,
                   'BD@car.COM',
                   01671);
       END;
[2021-08-04 18:58:33] [72000][20001]
[2021-08-04 18:58:33]   ORA-20001: You must be at least 13 years of age to sign up!
[2021-08-04 18:58:33]   ORA-06512: at "CS669.TRG_CUST_SIGNUP", line 12
[2021-08-04 18:58:33]   ORA-04088: error during execution of trigger 'CS669.TRG_CUST_SIGNUP
[2021-08-04 18:58:33]   ORA-06512: at "CS669.ADD_CUSTOMER", line 12
[2021-08-04 18:58:33]   ORA-06512: at line 9
[2021-08-04 18:58:33] Position: 0
```

2. Customer must be at least 18 years of age to book an R rated movie



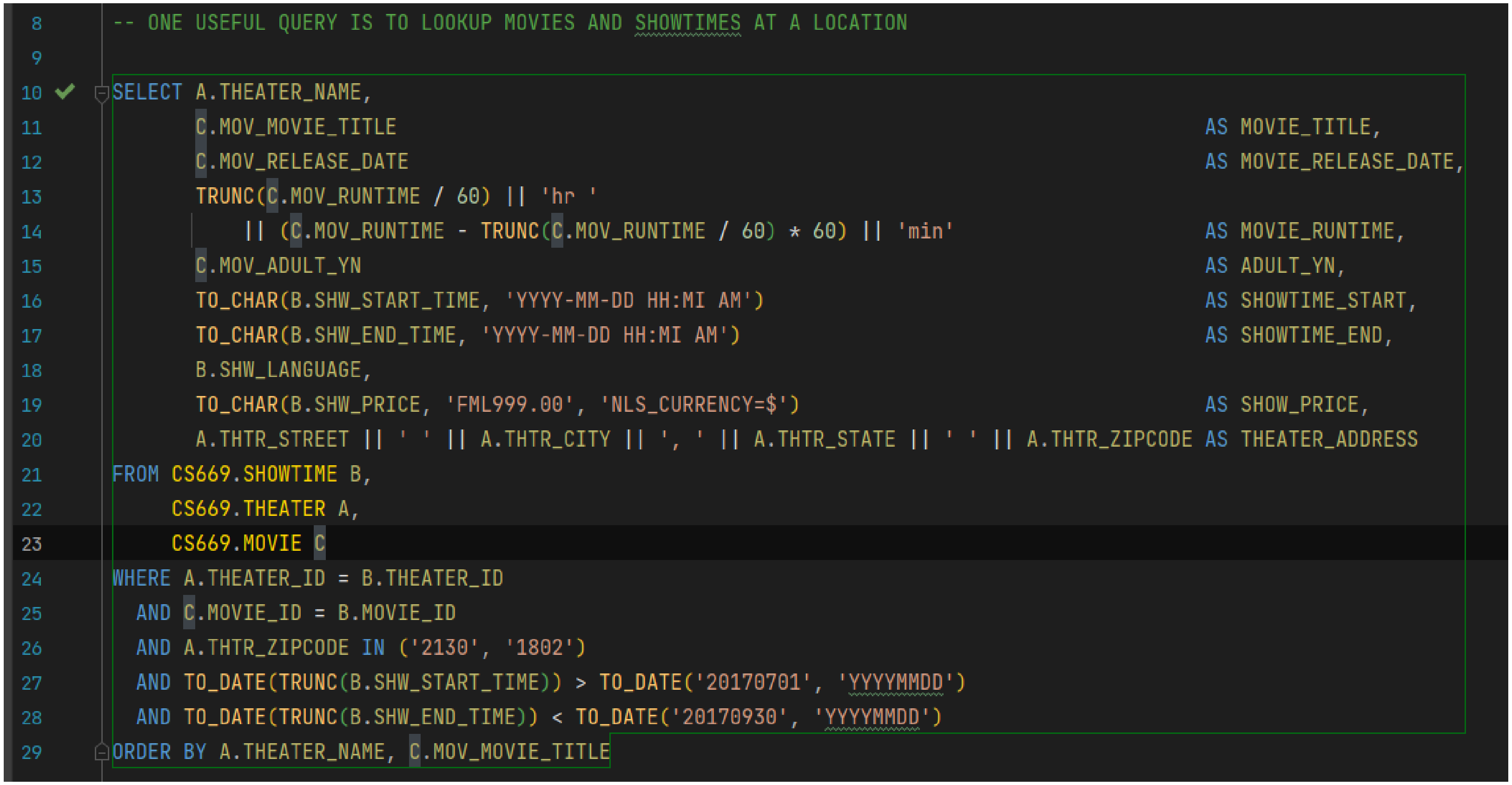
```
CS669> -- Trigger to ensure that the customer making a booking
       -- for an adult movie is at least 18 years of age
       CREATE OR REPLACE TRIGGER TRG_ADULT_BKG
           BEFORE INSERT OR UPDATE
           ON CS669.BOOKING
           FOR EACH ROW
       DECLARE
           V_CUS_AGE  NUMBER;
           V_ADULT_YN VARCHAR2(15);
       BEGIN

           -- Get age of the customer making the booking
           SELECT CAST(MONTHS_BETWEEN(TRUNC(SYSDATE), (SELECT CUS_DOB
                                                       FROM CS669.CUSTOMER C
                                                       WHERE C.CUSTOMER_ID = :NEW.CUSTOMER_ID)) / 12 AS INTEGER)
           INTO V_CUS_AGE
           FROM DUAL;

           -- See if the movie being booked is marked as adult
           SELECT MOV_ADULT_YN
           INTO V_ADULT_YN
           FROM CS669.MOVIE M
                    INNER JOIN SHOWTIME S on M.MOVIE_ID = S.MOVIE_ID
           WHERE S.SHOWTIME_ID = :NEW.SHOWTIME_ID;

           IF (UPPER(V_ADULT_YN) = 'Y' AND V_CUS_AGE < 18)
           THEN
               RAISE_APPLICATION_ERROR(-20001, 'You must be at least 18 years of age to book an R rated movie!');
           END IF;
       END;
[2021-08-04 19:22:37] completed in 45 ms
```



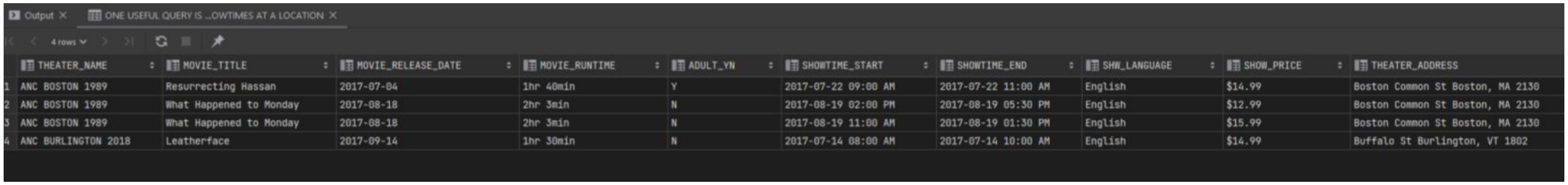
```
CS669> DECLARE
           NEX_BOOK_ID VARCHAR(4000);
           V_SEAT_LIST CS669.SEAT_ROW_LIST;
       BEGIN
           -- GET THE EXISTING MAX CUSTOMER ID AND ADD 1
           SELECT NVL(MAX(CAST(BOOKING_ID AS NUMBER)), 0) + 1
           INTO NEX_BOOK_ID
           FROM CS669.BOOKING;

           V_SEAT_LIST := CS669.SEAT_ROW_LIST('E2', 'E3', 'E7');

           MAKE_BOOKING(IN_BOOKING_ID => NEX_BOOK_ID,
                        IN_CUSTOMER_NAME => 'Peter Parker',
                        IN_SHOWTIME_ID => 2,
                        IN_SEATS => V_SEAT_LIST);

       END;
[2021-08-04 19:33:10] [72000][20001]
[2021-08-04 19:33:10]   ORA-20001: You must be at least 18 years of age to book an R rated movie!
[2021-08-04 19:33:10]   ORA-06512: at "CS669.TRG_ADULT_BKG", line 22
[2021-08-04 19:33:10]   ORA-04088: error during execution of trigger 'CS669.TRG_ADULT_BKG'
[2021-08-04 19:33:10]   ORA-06512: at "CS669.MAKE_BOOKING", line 40
[2021-08-04 19:33:10]   ORA-06512: at line 12
[2021-08-04 19:33:10] Position: 0
```

Organization-Driven Queries

Query 1: bookmytalkies app’s fundamental starting point is to show movies and their showtimes at a particular location based on where the customer’s located. Imagining that the front-end app would retrieve this data from the database, it would fire a query similar to the below – which lists the theater names, address, the movie name, runtime, showtimes for that movie – start and end times and the respective price.



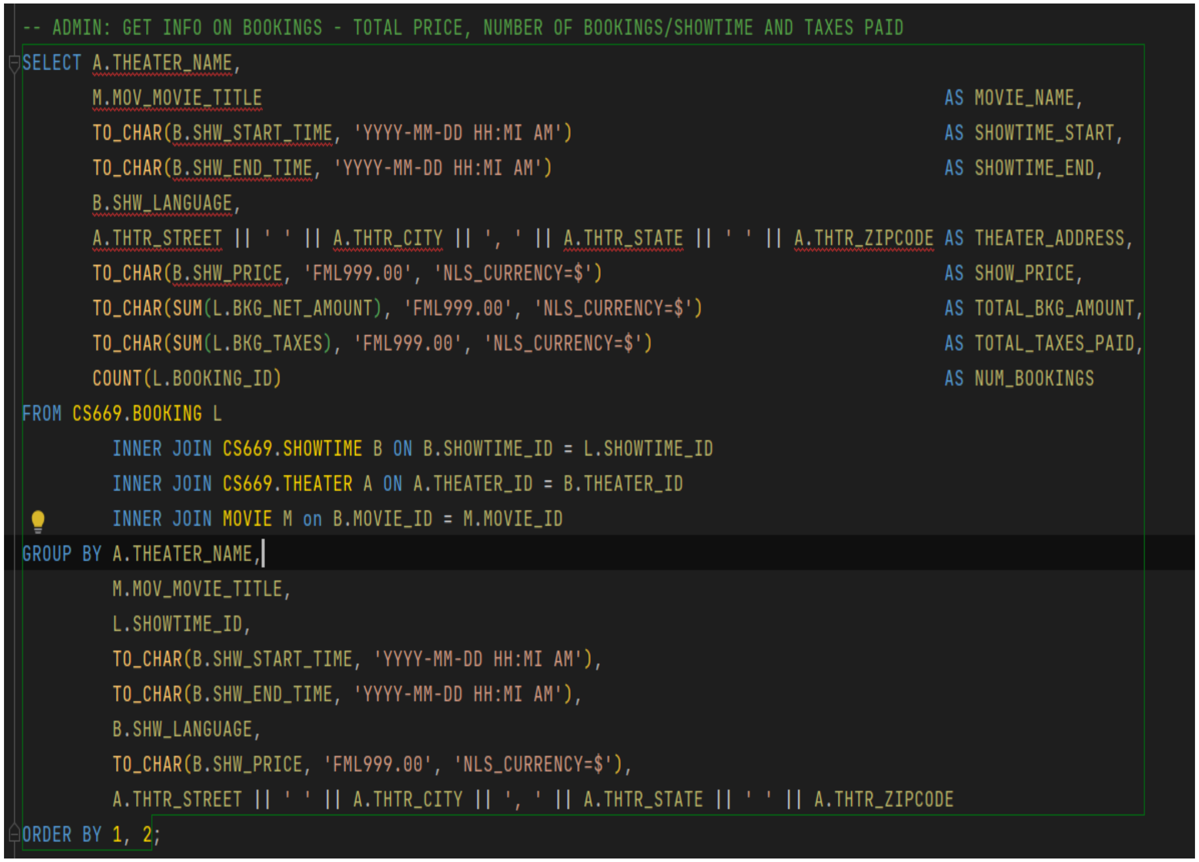
```sql
-- ONE USEFUL QUERY IS TO LOOKUP MOVIES AND SHOWTIMES AT A LOCATION

SELECT A.THEATER_NAME,
       C.MOV_MOVIE_TITLE                                                          AS MOVIE_TITLE,
       C.MOV_RELEASE_DATE                                                         AS MOVIE_RELEASE_DATE,
       TRUNC(C.MOV_RUNTIME / 60) || 'hr '
           || (C.MOV_RUNTIME - TRUNC(C.MOV_RUNTIME / 60) * 60) || 'min'           AS MOVIE_RUNTIME,
       C.MOV_ADULT_YN                                                             AS ADULT_YN,
       TO_CHAR(B.SHW_START_TIME, 'YYYY-MM-DD HH:MI AM')                           AS SHOWTIME_START,
       TO_CHAR(B.SHW_END_TIME, 'YYYY-MM-DD HH:MI AM')                             AS SHOWTIME_END,
       B.SHW_LANGUAGE,
       TO_CHAR(B.SHW_PRICE, 'FML999.00', 'NLS_CURRENCY=$')                        AS SHOW_PRICE,
       A.THTR_STREET || ' ' || A.THTR_CITY || ', ' || A.THTR_STATE || ' ' || A.THTR_ZIPCODE AS THEATER_ADDRESS
FROM CS669.SHOWTIME B,
     CS669.THEATER A,
     CS669.MOVIE C
WHERE A.THEATER_ID = B.THEATER_ID
  AND C.MOVIE_ID = B.MOVIE_ID
  AND A.THTR_ZIPCODE IN ('2130', '1802')
  AND TO_DATE(TRUNC(B.SHW_START_TIME)) > TO_DATE('20170701', 'YYYYMMDD')
  AND TO_DATE(TRUNC(B.SHW_END_TIME)) < TO_DATE('20170930', 'YYYYMMDD')
ORDER BY A.THEATER_NAME, C.MOV_MOVIE_TITLE
```



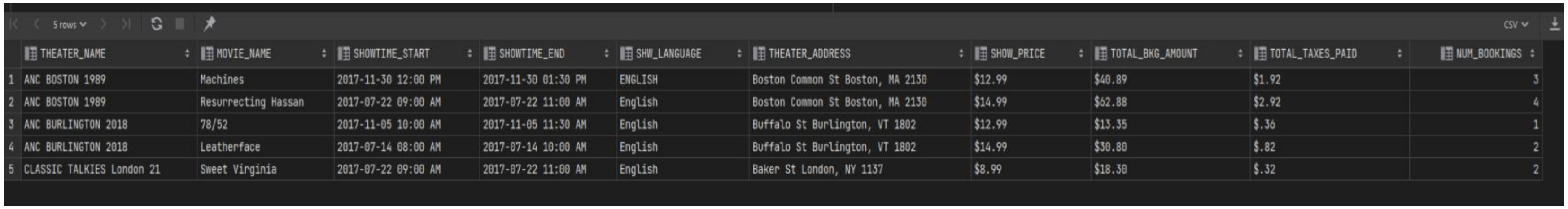
	THEATER_NAME	MOVIE_TITLE	MOVIE_RELEASE_DATE	MOVIE_RUNTIME	ADULT_YN	SHOWTIME_START	SHOWTIME_END	SHW_LANGUAGE	SHOW_PRICE	THEATER_ADDRESS
1	ANC BOSTON 1989	Resurrecting Hassan	2017-07-04	1hr 40min	Y	2017-07-22 09:00 AM	2017-07-22 11:00 AM	English	$14.99	Boston Common St Boston, MA 2130
2	ANC BOSTON 1989	What Happened to Monday	2017-08-18	2hr 3min	N	2017-08-19 02:00 PM	2017-08-19 05:30 PM	English	$12.99	Boston Common St Boston, MA 2130
3	ANC BOSTON 1989	What Happened to Monday	2017-08-18	2hr 3min	N	2017-08-19 11:00 AM	2017-08-19 01:30 PM	English	$15.99	Boston Common St Boston, MA 2130
4	ANC BURLINGTON 2018	Leatherface	2017-09-14	1hr 30min	N	2017-07-14 08:00 AM	2017-07-14 10:00 AM	English	$14.99	Buffalo St Burlington, VT 1802

Query 2: From an administration or analytics standpoint, it would be useful to see how many bookings have been made for a showtime – respective to that movie and theater. Although this is a simple use case/query, it can be extended further to analyze revenue/profits over time.

The query along with the result set is shown below – includes details of theaters, movie, showtimes – times, price, tickets booked and total amount in dollars, included taxes paid.

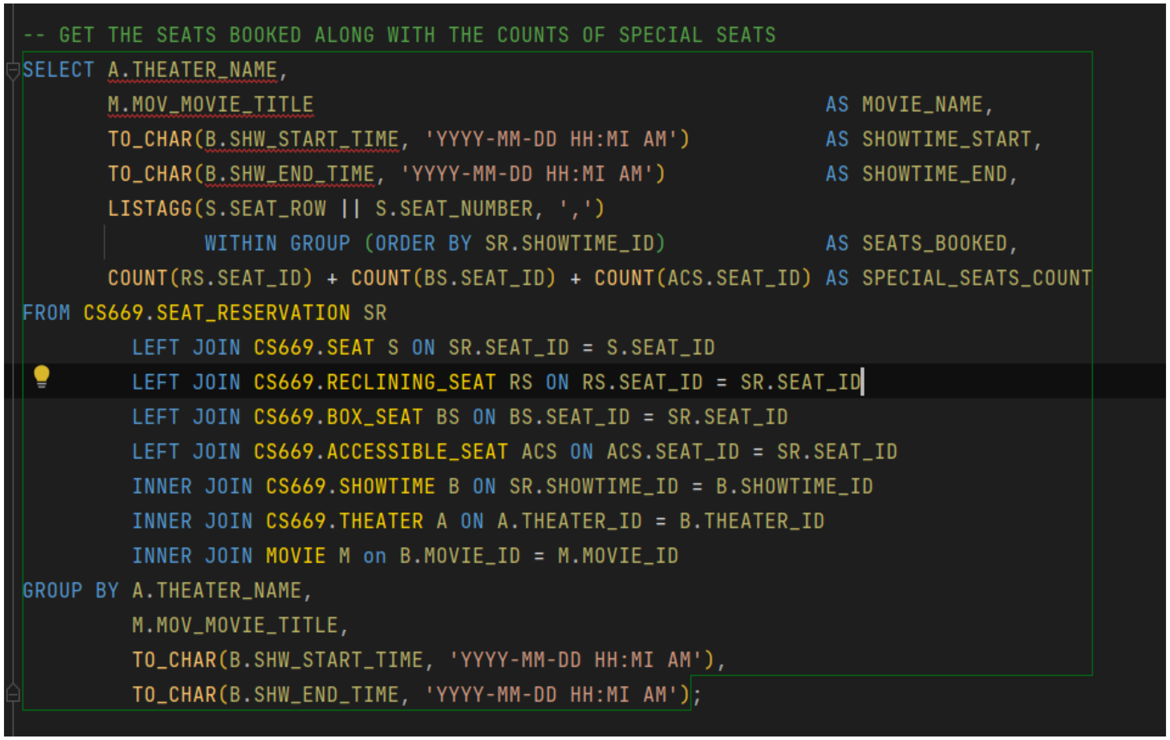


```
-- ADMIN: GET INFO ON BOOKINGS - TOTAL PRICE, NUMBER OF BOOKINGS/SHOWTIME AND TAXES PAID
SELECT A.THEATER_NAME,
       M.MOV_MOVIE_TITLE                                                                  AS MOVIE_NAME,
       TO_CHAR(B.SHW_START_TIME, 'YYYY-MM-DD HH:MI AM')                                   AS SHOWTIME_START,
       TO_CHAR(B.SHW_END_TIME, 'YYYY-MM-DD HH:MI AM')                                     AS SHOWTIME_END,
       B.SHW_LANGUAGE,
       A.THTR_STREET || ' ' || A.THTR_CITY || ', ' || A.THTR_STATE || ' ' || A.THTR_ZIPCODE AS THEATER_ADDRESS,
       TO_CHAR(B.SHW_PRICE, 'FML999.00', 'NLS_CURRENCY=$')                                AS SHOW_PRICE,
       TO_CHAR(SUM(L.BKG_NET_AMOUNT), 'FML999.00', 'NLS_CURRENCY=$')                      AS TOTAL_BKG_AMOUNT,
       TO_CHAR(SUM(L.BKG_TAXES), 'FML999.00', 'NLS_CURRENCY=$')                           AS TOTAL_TAXES_PAID,
       COUNT(L.BOOKING_ID)                                                                AS NUM_BOOKINGS
FROM CS669.BOOKING L
         INNER JOIN CS669.SHOWTIME B ON B.SHOWTIME_ID = L.SHOWTIME_ID
         INNER JOIN CS669.THEATER A ON A.THEATER_ID = B.THEATER_ID
         INNER JOIN MOVIE M on B.MOVIE_ID = M.MOVIE_ID
GROUP BY A.THEATER_NAME,
         M.MOV_MOVIE_TITLE,
         L.SHOWTIME_ID,
         TO_CHAR(B.SHW_START_TIME, 'YYYY-MM-DD HH:MI AM'),
         TO_CHAR(B.SHW_END_TIME, 'YYYY-MM-DD HH:MI AM'),
         B.SHW_LANGUAGE,
         TO_CHAR(B.SHW_PRICE, 'FML999.00', 'NLS_CURRENCY=$'),
         A.THTR_STREET || ' ' || A.THTR_CITY || ', ' || A.THTR_STATE || ' ' || A.THTR_ZIPCODE
ORDER BY 1, 2;
```

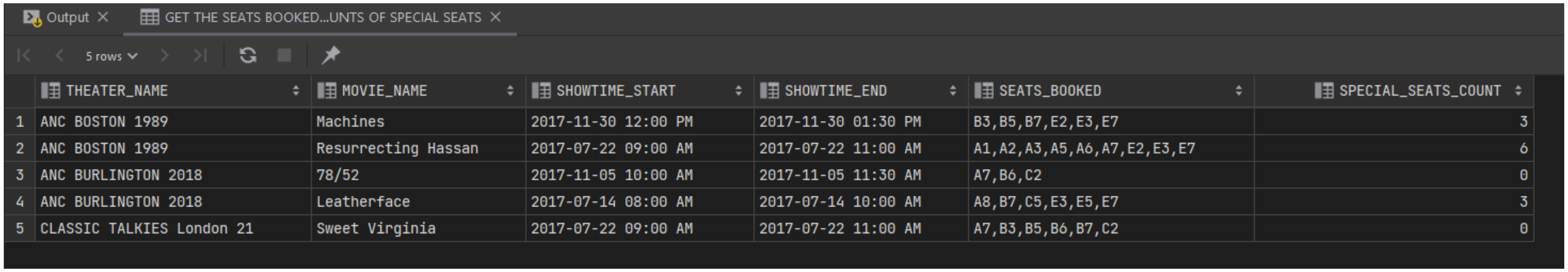


	THEATER_NAME	MOVIE_NAME	SHOWTIME_START	SHOWTIME_END	SHW_LANGUAGE	THEATER_ADDRESS	SHOW_PRICE	TOTAL_BKG_AMOUNT	TOTAL_TAXES_PAID	NUM_BOOKINGS
1	ANC BOSTON 1989	Machines	2017-11-30 12:00 PM	2017-11-30 01:30 PM	ENGLISH	Boston Common St Boston, MA 2130	$12.99	$40.89	$1.92	3
2	ANC BOSTON 1989	Resurrecting Hassan	2017-07-22 09:00 AM	2017-07-22 11:00 AM	English	Boston Common St Boston, MA 2130	$14.99	$62.88	$2.92	4
3	ANC BURLINGTON 2018	78/52	2017-11-05 10:00 AM	2017-11-05 11:30 AM	English	Buffalo St Burlington, VT 1802	$12.99	$13.35	$.36	1
4	ANC BURLINGTON 2018	Leatherface	2017-07-14 08:00 AM	2017-07-14 10:00 AM	English	Buffalo St Burlington, VT 1802	$14.99	$30.80	$.82	2
5	CLASSIC TALKIES London 21	Sweet Virginia	2017-07-22 09:00 AM	2017-07-22 11:00 AM	English	Baker St London, NY 1137	$8.99	$18.30	$.32	2

Query 3: Another analytics use case would be to understand the trends of "specialized" seat bookings – assuming theaters offer reclining, box and accessible seats along with normal seats, getting the count of such bookings would help understand how customers book such special seats – perhaps on special occasions or for certain types of movies or even families/groups all booking one type of seat.
Beyond this query, one could drill down to understand the details of such bookings, price differences etc which are not in scope here.



```sql
-- GET THE SEATS BOOKED ALONG WITH THE COUNTS OF SPECIAL SEATS
SELECT A.THEATER_NAME,
       M.MOV_MOVIE_TITLE                                     AS MOVIE_NAME,
       TO_CHAR(B.SHW_START_TIME, 'YYYY-MM-DD HH:MI AM')      AS SHOWTIME_START,
       TO_CHAR(B.SHW_END_TIME, 'YYYY-MM-DD HH:MI AM')        AS SHOWTIME_END,
       LISTAGG(S.SEAT_ROW || S.SEAT_NUMBER, ',')
               WITHIN GROUP (ORDER BY SR.SHOWTIME_ID)        AS SEATS_BOOKED,
       COUNT(RS.SEAT_ID) + COUNT(BS.SEAT_ID) + COUNT(ACS.SEAT_ID) AS SPECIAL_SEATS_COUNT
FROM CS669.SEAT_RESERVATION SR
         LEFT JOIN CS669.SEAT S ON SR.SEAT_ID = S.SEAT_ID
         LEFT JOIN CS669.RECLINING_SEAT RS ON RS.SEAT_ID = SR.SEAT_ID
         LEFT JOIN CS669.BOX_SEAT BS ON BS.SEAT_ID = SR.SEAT_ID
         LEFT JOIN CS669.ACCESSIBLE_SEAT ACS ON ACS.SEAT_ID = SR.SEAT_ID
         INNER JOIN CS669.SHOWTIME B ON SR.SHOWTIME_ID = B.SHOWTIME_ID
         INNER JOIN CS669.THEATER A ON A.THEATER_ID = B.THEATER_ID
         INNER JOIN MOVIE M on B.MOVIE_ID = M.MOVIE_ID
GROUP BY A.THEATER_NAME,
         M.MOV_MOVIE_TITLE,
         TO_CHAR(B.SHW_START_TIME, 'YYYY-MM-DD HH:MI AM'),
         TO_CHAR(B.SHW_END_TIME, 'YYYY-MM-DD HH:MI AM');
```



	THEATER_NAME	MOVIE_NAME	SHOWTIME_START	SHOWTIME_END	SEATS_BOOKED	SPECIAL_SEATS_COUNT
1	ANC BOSTON 1989	Machines	2017-11-30 12:00 PM	2017-11-30 01:30 PM	B3,B5,B7,E2,E3,E7	3
2	ANC BOSTON 1989	Resurrecting Hassan	2017-07-22 09:00 AM	2017-07-22 11:00 AM	A1,A2,A3,A5,A6,A7,E2,E3,E7	6
3	ANC BURLINGTON 2018	78/52	2017-11-05 10:00 AM	2017-11-05 11:30 AM	A7,B6,C2	0
4	ANC BURLINGTON 2018	Leatherface	2017-07-14 08:00 AM	2017-07-14 10:00 AM	A8,B7,C5,E3,E5,E7	3
5	CLASSIC TALKIES London 21	Sweet Virginia	2017-07-22 09:00 AM	2017-07-22 11:00 AM	A7,B3,B5,B6,B7,C2	0