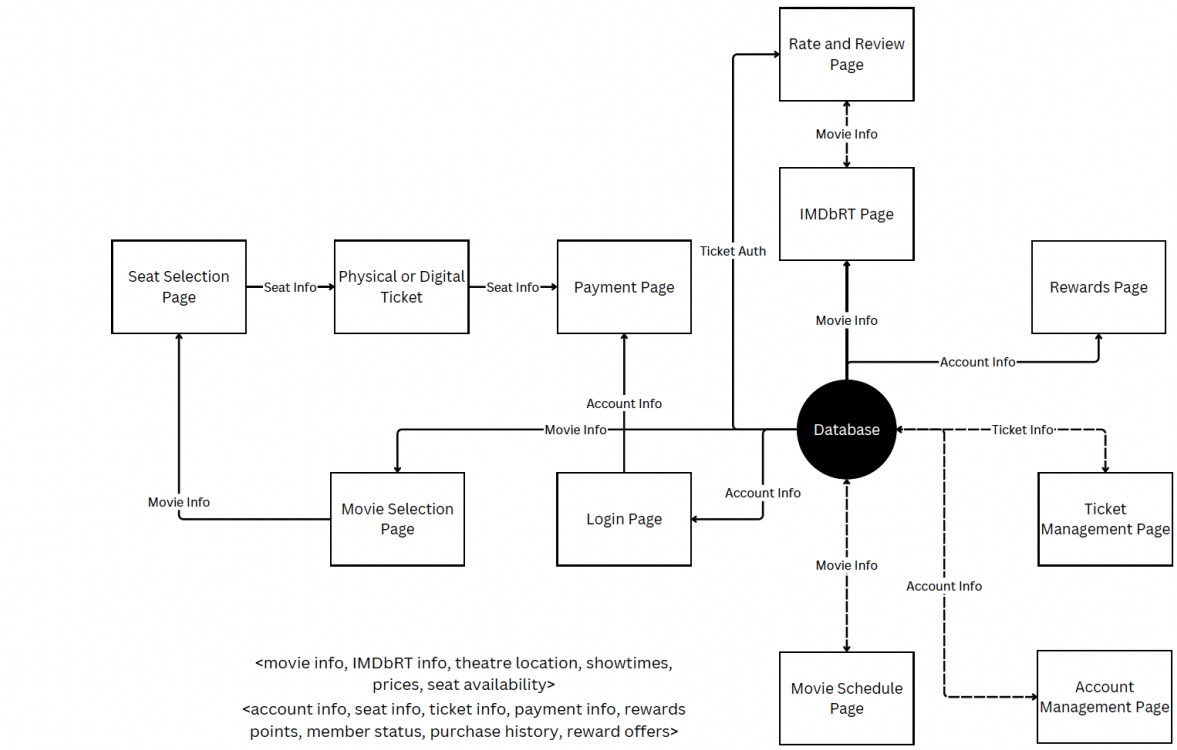
Design Management Strategy - CheepTIX



- **Updated diagram overall to be more intuitive**
- **Updated to include IMDbRT info, seat availability, and reward offers which are necessary for our design**
- **Ticket Management Page corrected to be a double arrow connected to the database, which reflects the user's ability to make changes to their tickets**
- **Movie Schedule Page no longer requires login auth, as users are able to freely view the page without a login**

Modified Software Architecture Diagram

Strategy:

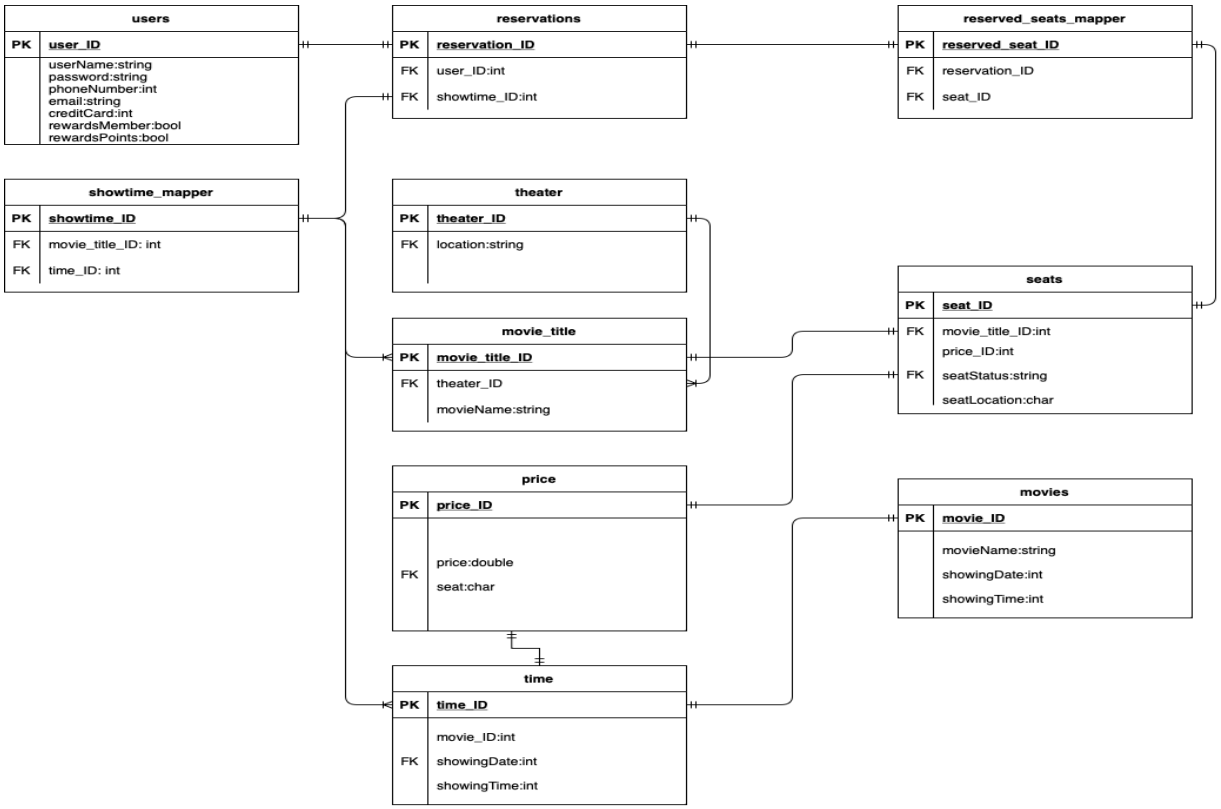
CheepTIX is adhering to a **SQL** data management strategy, as the movie ticketing system is largely table based in the form of rows and columns. Furthermore, the movie ticket data is not hierarchical, as the data does not contain parent and child relationships. If this were the case, SQL would not be well-suited. Furthermore, the ticketing system is itself a highly transaction-based application, in which to manage this facet and ensure smooth transactions of tickets, SQL is the preferred method.

Number of Databases:

We are utilizing one database, but will have multiple collections within that single database. The reason for this is to simplify deployment and management operations of CheepTIX, and to

reduce the latency for cross-collection data relationships. For example, this cross collection would occur in an instance after "**Authorization**" from the "**Login Page**", in which both "**Movie Info**" and "**Account Info**" are shortly called from the Database after one another to determine the "**Movie Schedule Page**".

Data Split and Logic:

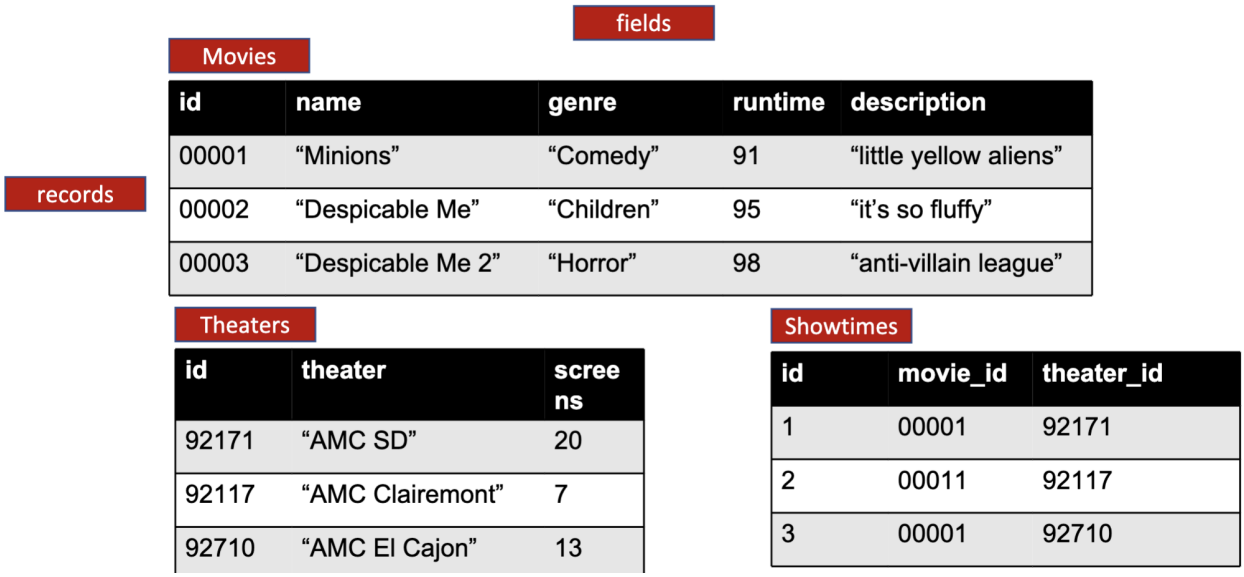


CheepTIX is splitting the data according to the entity diagram above, in which we have laid out the data that our application collects accordingly. The functions and classes within our program are titled above their respective primary key, and have the foreign key data values listed underneath to indicate their correlation. The diagram also notes the SQL relationship between entities, as double lines indicate one-to-one relationship, double line and arrow indicate one-to-many relationship, and arrows on both ends indicate many-to-many relationship. Despite some of the similar sounding classes, we have localized reserved_seats_mapper and showtime_mapper for instance to maintain the data split and preserve access speed.

Trade Off Discussion:

If we were to contrast from the SQL's ACID approach and use a NoSQL BASE alternative, our data would have to be stored as documents in the database as opposed to tables, ie. Movies Tables, Theaters Table vs. Movies as documents, Theaters as documents.

For instance, in the movie ticketing system, data is arranged as such in the database:



id	name	genre	runtime	description
00001	“Minions”	“Comedy”	91	“little yellow aliens”
00002	“Despicable Me”	“Children”	95	“it’s so fluffy”
00003	“Despicable Me 2”	“Horror”	98	“anti-villain league”

id	theater	screens
92171	“AMC SD”	20
92117	“AMC Clairemont”	7
92710	“AMC El Cajon”	13

id	movie_id	theater_id
1	00001	92171
2	00011	92117
3	00001	92710

This change is dependent on how data is stored and manipulated in each style, as NoSQL stores data in flexible, JSON-like documents. Furthermore, some of the tradeoffs for using SQL vs NoSQL in the following categories are that:

- Data Structure Flexibility
    - Since SQL requires predefined schema, this makes it less flexible and in changing the schema, this might require modifications to the database structure.
- Complex Queries
    - The SQL DB excels in handling complex queries that require joining multiple tables. As a result, SQL is advantageous for CheepTIX and other movie ticketing systems
- Data Consistency
    - SQL ensures strong data consistency and integrity through the aforementioned ACID transaction model. This is crucial for maintaining data accuracy in systems like CheepTIX where payments and books need to be precisely recorded for ticket buyers
- Scalability
    - SQL can handle moderate workloads moderately, but will most definitely require more complex solutions for scaling horizontally to accommodate high traffic and data growth. CheepTIX is only servicing San Diego County, so SQL is better suited albeit this insufficiency
- Development Ease
    - Relational databases are easier to work with for structured data with defined relationships, such as movie data and ticket data