The Introduction and Use of SQL Databases

Student Name

Institutional Affiliation

Professor's Name

Date

The Introduction and Use of SQL Databases

Storage systems are essential in computer systems that store and process data over time. In the current technological era, different applications are being developed as solutions to different problems around the globe. These include mobile applications to grant people access to services, banking applications for the storage of transactions and to allow users access to their accounts, and many more. Machine learning and AI-driven applications process large amounts of data to come up with patterns that are used for the prediction of certain events. Therefore, data storage systems that enable the efficient storage and retrieval of data have evolved over time to facilitate these use cases.

Data storage systems have evolved over time. The first computer-based storage system consisted of punch cards developed by Basile Bouchon in 1725. The same technology was proposed by Charles Babbage in 1837 to create the Analytical Engine, which was used as a primitive calculator. The technology has thus advanced from punch cards to the solid-state drives, data silos, and cloud data storage used today (Foote, 2023).

Database Design and Implementation

Types of databases

Before writing an application, the developer has to think about how data will be represented (data modeling). The most commonly known and widely used models utilize Structured Query Language (SQL), based on the relational model proposed by Edgar Codd in 1970 (Kleppmann, 2021). SQL is a standardized programming language used to manage relational databases. Not Only SQL (NoSQL) is the alternative to relational-based models. However, NoSQL deals with non-relational models.

The two approaches are suitable in different applications depending on the use case scenario. The key difference between SQL and NoSQL is the use of a schema. A schema is a structure of how the data in different entities is related. SQL-based databases will keep a schema of how the different entities are linked. For instance, the “client” entity is linked to the “sales” one via the client_id foreign key reference in the sales table. Thus, when storing data in the sales table, you have to associate it with the client making the sales.

SQL databases can be classified as transactional or historical. Transactions are primarily used in our daily activities for processing data. This includes data from bank transactions where a client makes a deposit. On the other hand, a historical database stores large quantities of data over a long period. For instance, weather data stored for 10 or 20 years can be used for forecasting. Therefore, historical databases are mainly used in data mining and knowledge-discovery applications. Specific patterns can be retrieved from data and used to predict what will happen if such patterns reappear (Kleppmann, 2021). Historical databases generate patterns that are mostly used in machine learning and AI training, while transactional databases are used in applications that perform daily operations.

Types of SQL systems available

There are different relational database management systems (RDBMS) that support the use of SQL. These mainly include MySQL, Microsoft SQL, Oracle Database, PostgreSQL and IBM Db2. These RDBMS have similar SQL structures, with little variation in the syntax. The code snippets shown in this document will be from a PostgreSQL database running on a Linux operating system. The RDBMS are cross-platform and can work on several available operating systems, including Windows and Mac.

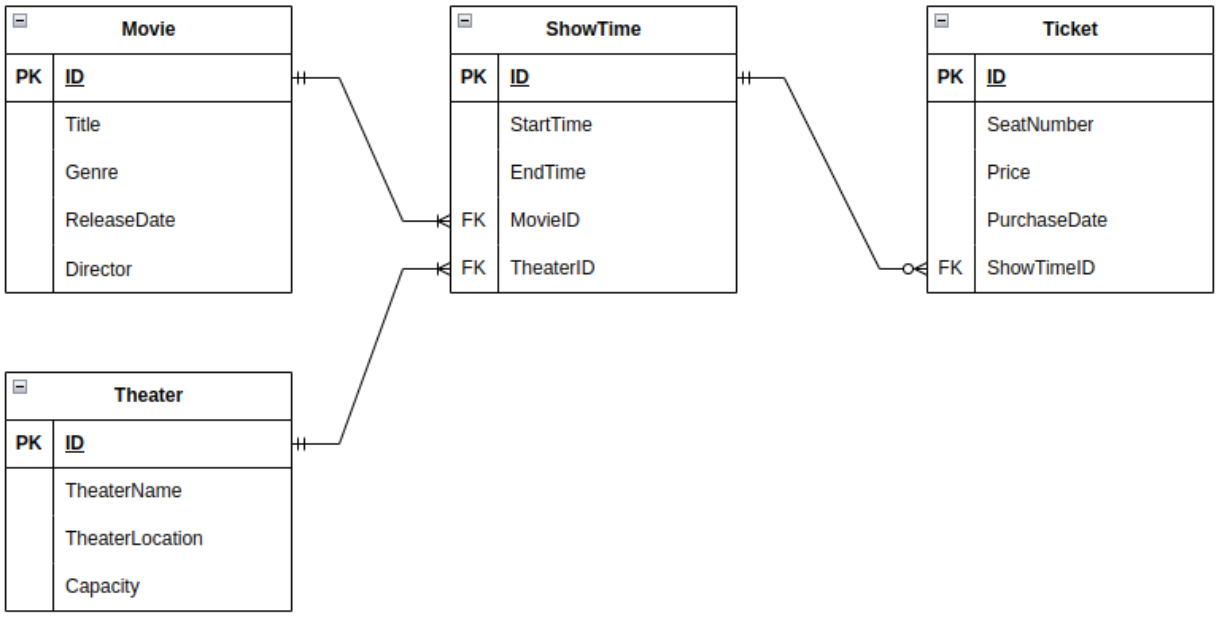
Creating databases and tables

The first order of operation when creating a database is to log into the Postgres system. After that, you enter the interactive mode where you can write the SQL commands. Next, a database is needed to hold the table and data being stored. The use of a database enables the system to isolate and containerize different pieces of data being stored in the system. This enables system administrators to regulate the people who can access the stored data.

```
-- Delete database if it does not exit
DROP DATABASE IF EXISTS MovieTicket;

-- Create the MovieTicket database
CREATE DATABASE MovieTicket;

-- Use the MovieTicket database with the following operations
-- USE MovieTicket;
-- Use the MovieTicket database in postgres
\c movieticket
```

The database is dropped to prevent conflicts when recreating the same. It is important to note that the information within a dropped database cannot be recovered. After creating the database, switch to the new DB to be able to create tables or entities and store data in it.

It is essential to have a schema when dealing with a relational database. The schema guides the database on how the tables are related. Below is the Entity Relation Diagram (ERD) showing the structure of the system.



In this diagram, the entities have been linked with different types of relations. The Showtime table can have one or many movies, which is achieved via a one-to-many foreign key relation. Theater is related to the showtime via a one-to-many foreign key. Showtime can have none, one, or more tickets linked to it. Foreign keys help to enforce relationships and data integrity in SQL databases.

The code snippet below shows the SQL code needed to implement the ERD above. When creating a table, a primary key is a unique value that will hold the position of a record in the table. The primary key is also an index, meaning that it is stored in a different table to reference the location of the stored record. Indices improve performance by enabling the system to look through a small dataset and access specific records rather than sequentially searching through all of them. Foreign keys are used to link and reference records in separate entities (Campbell &

Majors, 2018). Thus, foreign key needs to be of the same data type as the referenced key.

```
-- Create the Theater relation
CREATE TABLE Theater (
    ID SERIAL PRIMARY KEY,
    TheaterName VARCHAR(255),
    TheaterLocation VARCHAR(255),
    Capacity INTEGER DEFAULT(0)
);

-- Create the Movie relation
CREATE TABLE Movie (
    ID SERIAL PRIMARY KEY,
    Title VARCHAR(255),
    Genre VARCHAR(255),
    ReleaseDate DATE NULL,
    Director VARCHAR(255) NULL
);

-- Create the showtime relation
CREATE TABLE Showtime (
    ID SERIAL PRIMARY KEY,
    MovieID INTEGER REFERENCES Movie(ID),
    TheaterID INTEGER REFERENCES Theater(ID),
    StartTime TIMESTAMP NULL,
    EndTime TIMESTAMP NULL,
    CONSTRAINT FK_ShowtimeMovie FOREIGN KEY (MovieID) REFERENCES Movie(ID),
    CONSTRAINT FK_ShowtimeTheater FOREIGN KEY (TheaterID) REFERENCES Theater(ID)
);

-- Create the Ticket relation
CREATE TABLE Ticket (
    ID SERIAL PRIMARY KEY,
    ShowtimeID INTEGER REFERENCES Showtime(ID),
    SeatNumber VARCHAR(255),
    Price DECIMAL(5, 2) DEFAULT(0),
    CONSTRAINT FK_TicketShowtime FOREIGN KEY (ShowtimeID) REFERENCES Showtime(ID)
);
```

The insertion and querying of data

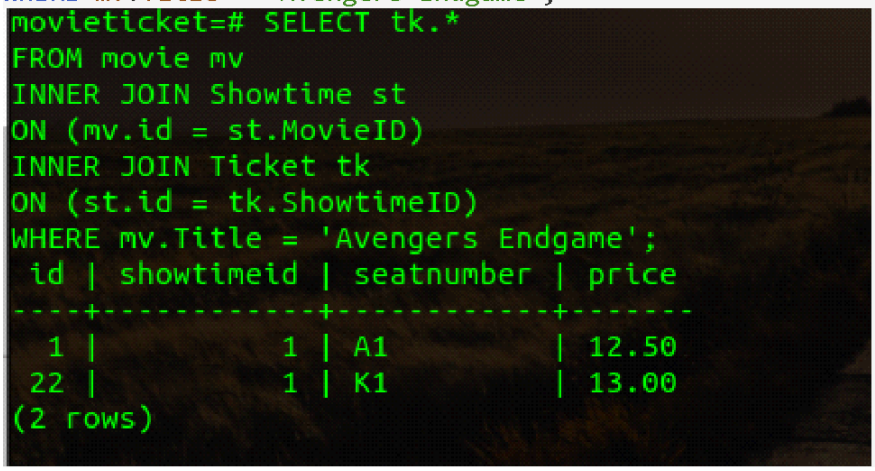
The following commands show how to insert data in the tables created. The insert command takes in the table name, column names, and values to be inserted.

```
INSERT INTO Theater (TheaterName, TheaterLocation, Capacity)
VALUES ('Aston Villa', 'City Center', 200), ('Silver Screen', 'Downtown', 150),
('Prestige Cinemas', 'Suburbia', 180), ('Starlight Multiplex', 'City Mall', 220);

INSERT INTO Movie (Title, Genre, ReleaseDate, Director)
VALUES
('Avengers Endgame', 'Action, Adventure', '2019-04-26', 'Anthony and Joe Russo'),
('The Shawshank Redemption', 'Drama', '1994-09-23', 'Frank Darabont'),
('The Godfather', 'Crime, Drama', '1972-03-15', 'Francis Ford Coppola'),
('The Dark Knight', 'Action, Crime, Drama', '2008-07-18', 'Christopher Nolan'),
('Pulp Fiction', 'Crime, Drama', '1994-10-14', 'Quentin Tarantino'),
('Schindler''s List', 'Biography, Drama, History', '1993-12-15', 'Steven Spiel-
berg'),
('Forrest Gump', 'Drama, Romance', '1994-07-06', 'Robert Zemeckis'),
('The Matrix', 'Action, Sci-Fi', '1999-03-31', 'Lana and Lilly Wachowski'),
('Inception', 'Action, Adventure, Sci-Fi', '2010-07-16', 'Christopher Nolan'),
('The Silence of the Lambs', 'Crime, Drama, Thriller', '1991-02-14', 'Jonathan
Demme'),
('The Shawshank Redemption', 'Drama', '1994-09-23', 'Frank Darabont');

INSERT INTO Showtime (MovieID, TheaterID, StartTime, EndTime)
VALUES (1, 1, '2023-01-15 15:00:00', '2023-01-15 17:00:00'),
(2, 2, '2023-01-15 18:00:00', '2023-01-15 20:00:00'),
(3, 3, '2023-01-16 14:30:00', '2023-01-16 16:30:00'),
(4, 4, '2023-01-16 17:30:00', '2023-01-16 19:30:00'),
(5, 1, '2023-01-17 19:00:00', '2023-01-17 21:00:00'),
(6, 2, '2023-01-18 16:00:00', '2023-01-18 18:00:00'),
(7, 3, '2023-01-18 20:30:00', '2023-01-18 22:30:00'),
(8, 4, '2023-01-19 14:00:00', '2023-01-19 16:00:00'),
(9, 1, '2023-01-19 18:30:00', '2023-01-19 20:30:00'),
(10, 2, '2023-01-20 17:30:00', '2023-01-20 19:30:00'),
(11, 3, '2023-01-20 20:00:00', '2023-01-20 22:00:00');
```

```
INSERT INTO Ticket (ShowtimeID, SeatNumber, Price)
VALUES (1, 'A1', 12.50), (2, 'B3', 10.00), (3, 'C5', 15.00), (4, 'D2', 11.50),
(5, 'E4', 13.00), (6, 'F1', 14.50), (7, 'G6', 12.00), (8, 'H8', 10.50),
(9, 'I3', 16.00), (10, 'J5', 11.00), (11, 'K2', 14.50), (2, 'A3', 13.50),
(3, 'B5', 11.00), (4, 'C2', 14.00), (5, 'D4', 12.50), (6, 'E1', 15.00),
(7, 'F6', 13.50), (8, 'G8', 11.00), (9, 'H3', 16.50), (10, 'I5', 12.00),
(11, 'J2', 15.50), (1, 'K1', 13.00), (2, 'A4', 10.50), (3, 'B6', 15.00),
(4, 'C3', 11.50), (5, 'D5', 14.00), (6, 'E2', 12.50), (7, 'F7', 13.00),
(8, 'G1', 10.50), (9, 'H4', 15.50), (10, 'I6', 11.00);
```

SQL can be used to retrieve the data stored in the table to create different reports. “Select statement” retrieves data from a particular table. Different types of ‘joins’ can be used to get data from more than one table. Foreign keys can be used to record data in different tables, since the keys are common in the linked tables.

```
-- Get all seats booked for a given movie
SELECT tk.* FROM movie mv
INNER JOIN Showtime st ON (mv.id = st.MovieID)
INNER JOIN Ticket tk ON (st.id = tk.ShowtimeID)
WHERE mv.Title = 'Avengers Endgame';
```



```
movieticket=# SELECT tk.*
FROM movie mv
INNER JOIN Showtime st
ON (mv.id = st.MovieID)
INNER JOIN Ticket tk
ON (st.id = tk.ShowtimeID)
WHERE mv.Title = 'Avengers Endgame';
 id | showtimeid | seatnumber | price
----+------------+------------+-------
  1 |          1 | A1         | 12.50
 22 |          1 | K1         | 13.00
(2 rows)
```

```sql
-- Find the movies displayed at 'Prestige Cinemas'
SELECT mv.* FROM Movie mv
INNER JOIN Showtime st ON (mv.ID = st.MovieID)
INNER JOIN Theater th ON (th.id = st.TheaterID)
WHERE th.TheaterName = 'Prestige Cinemas';
```

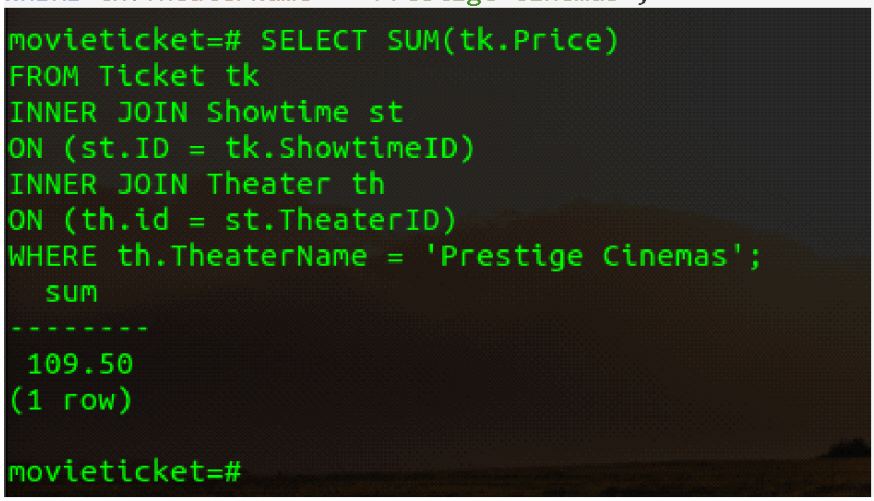
```
movieticket=# SELECT mv.*
FROM Movie mv
INNER JOIN Showtime st
ON (mv.ID = st.MovieID)
INNER JOIN Theater th
ON (th.id = st.TheaterID)
WHERE th.TheaterName = 'Prestige Cinemas';
 id |          title           |     genre      | releasedate |       director
----+--------------------------+----------------+-------------+----------------------
  3 | The Godfather            | Crime, Drama   | 1972-03-15  | Francis Ford Coppola
  7 | Forrest Gump             | Drama, Romance | 1994-07-06  | Robert Zemeckis
 11 | The Shawshank Redemption | Drama          | 1994-09-23  | Frank Darabont
(3 rows)
```

SQL supports the use of aggregations to generate the desired reports. Aggregation is helpful when dealing with calculations. Many reports usually require metrics, which can be generated from SQL tables with the use of these aggregations. The snippets below show how to perform basic aggregations on an SQL database.

```sql
-- Find the number of movies shown at 'Prestige Cinemas'
SELECT COUNT(mv.id) FROM Movie mv
INNER JOIN Showtime st ON (mv.ID = st.MovieID)
INNER JOIN Theater th ON (th.id = st.TheaterID)
WHERE th.TheaterName = 'Prestige Cinemas';
```

```
movieticket=# SELECT COUNT(mv.id)
FROM Movie mv
INNER JOIN Showtime st
ON (mv.ID = st.MovieID)
INNER JOIN Theater th
ON (th.id = st.TheaterID)
WHERE th.TheaterName = 'Prestige Cinemas';
 count
-------
     3
(1 row)

movieticket=#
```

```sql
-- Total income at 'Prestige Cinemas'
SELECT SUM(tk.Price) FROM Ticket tk
INNER JOIN Showtime st ON (st.ID = tk.ShowtimeID)
INNER JOIN Theater th ON (th.id = st.TheaterID)
WHERE th.TheaterName = 'Prestige Cinemas';
```



```
movieticket=# SELECT SUM(tk.Price)
FROM Ticket tk
INNER JOIN Showtime st
ON (st.ID = tk.ShowtimeID)
INNER JOIN Theater th
ON (th.id = st.TheaterID)
WHERE th.TheaterName = 'Prestige Cinemas';
  sum
--------
 109.50
(1 row)

movieticket=#
```

In summary, SQL-based databases have proven to be useful in daily applications. They have enabled the easier and more efficient storage, retrieval and manipulation of data for users and systems. Therefore, SQL databases have proven to be useful and are currently being utilized in several applications.

References

Campbell, L., & Majors, C. (2018). Data Storage, Indexing, and Replication. In *Database Reliability Engineering: Designing and Operating Resilient Database Systems* (pp. 181–190). essay, O'Reilly Media.

Foote, K. D. (2023, January 4). *A brief history of data storage*. DATAVERSITY. https://www.dataversity.net/brief-history-data-storage/

Kleppmann, M. (2021). Data Models and Query Language. In *Designing data-intensive applications: The big ideas behind reliable, scalable, and maintainable systems* (pp. 27–38). essay, O'Reilly.