4/19/2023

Isaiah Sarria, Randy Koliha, Simon Pringle, Damian Cutie, Christopher Aguiar

Central FL Bus System Database Design

# **Database Description**

Our project covers the database design of a Central Florida bus system. The database handles the backend processes that relate to route scheduling, bus stops, ticket prices, drivers, passengers, tickets, passes, and passenger user accounts.

Users: Passengers have limited access to the database using the online Lynx portal. Passengers can access information regarding ticket price, bus routes, and arrival and departure times.

Administrators: The Central Florida Transportation Authority, also known as LYNX, manages and provides the public transportation services for Orange, Seminole, and Osceola counties. In this case, we are only designing based off the public transportation services in Orlando. These administrators can fully access and manipulate the database system to update.

# **Business Rules:**

1. Many passengers can ride on one bus
2. One bus can only hold many passengers
3. A route can have many stops
4. Many stops can be a part of one route
5. Each passenger can only have one ticket
6. One ticket can be purchased by one passenger
7. One city can have many routes
8. Many routes may be in one city
9. Each bus has one route
10. One route may have many buses
11. Each passenger can sign up for one user account
12. One user account can only have one passenger associated
13. One city can have many buses
14. Many buses can be in one city
15. Many passengers can only purchase one pass.
16. A pass can be purchased by many passengers
17. Many tickets can be bought by one user account
18. One user account can buy many tickets
19. One pass can be registered to one user account
20. One user account can have one pass registered to it

# **Sources:**

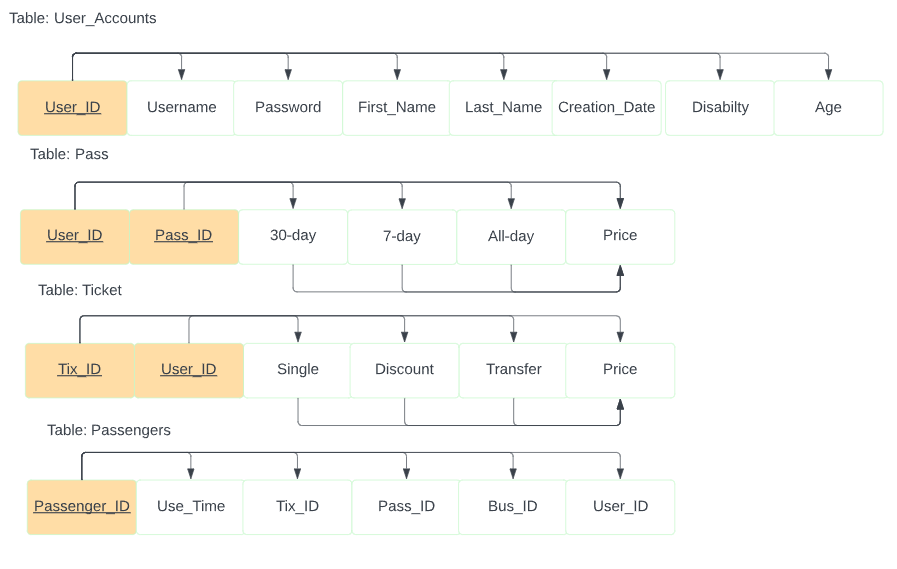
* City of Orlando’s Website about Public Transit and LYNX
  + <https://www.orlando.gov/Parking-Transportation/Public-Transit>
* A detailed Transport Hub Database Model for our reference
  + <https://vertabelo.com/blog/traveling-by-bus-or-train-a-transport-hub-database-model/>
* A Personal Blog with a brief and simple ERD model for our reference for a bus company
  + <https://afrincse.blogspot.com/2019/02/scenario-to-erd_96.html>
* A YouTube video detailing a bus system database system
  + <https://www.youtube.com/watch?v=ipIoNB5ghYo&ab_channel=AleenaKuriakose>
* Simple breakdown of Lynx system structure
  + <https://en.wikipedia.org/wiki/Lynx_(Orlando>)

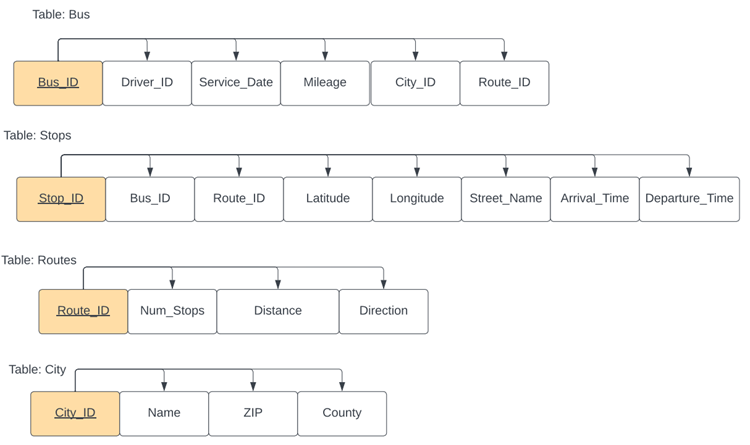
# **Data Dictionary**



The data dictionary groups all the attributes with its respective table and provides more details for each attribute. These details include the type of data the attribute can hold, its formatting, its accepted range of values, indicates if it is required, if it is a primary key and/or foreign key, and if it is a foreign key the data dictionary lists the table the foreign key references.

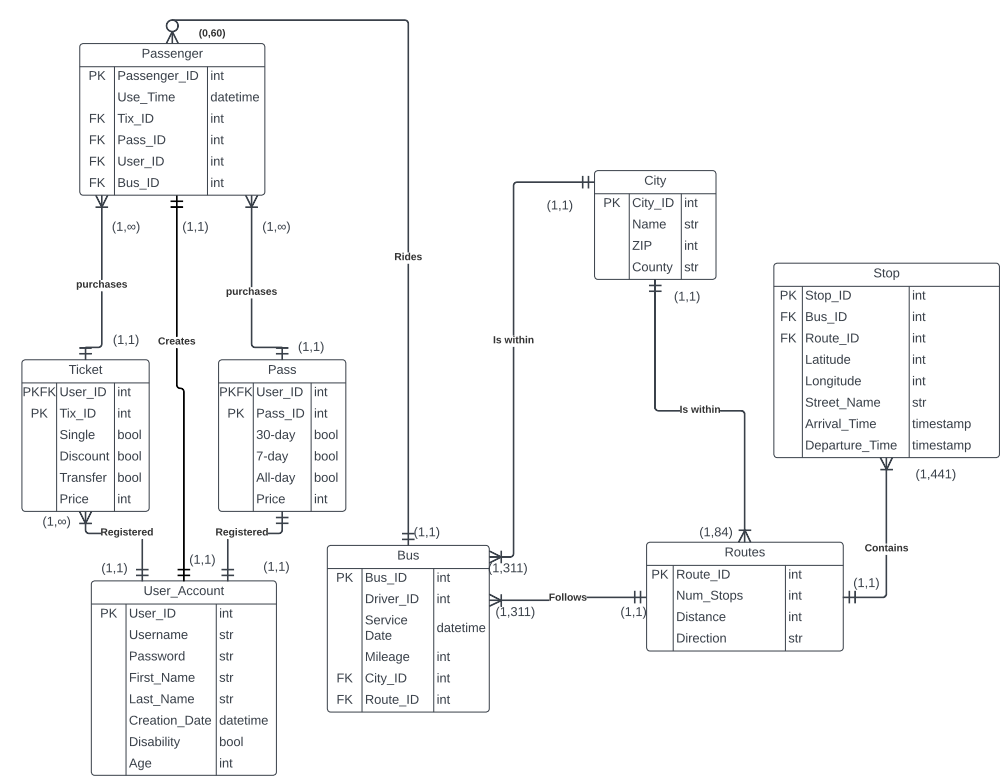
# **Dependency Diagrams**





Except for two tables, there are no other transitive or partial dependencies for any of the tables for performance purposes by preventing data redundancies and anomalies for. The tables Pass and Ticket we made them in 2NF because if the tables were reconstructed to be in 3NF there would be too many redundant tables. For instance, we would have to create 3 tables that show the dependence on the price. So, the transitive dependency is necessary to show the dependence on the price. This same logic can be applied to the Ticket table.

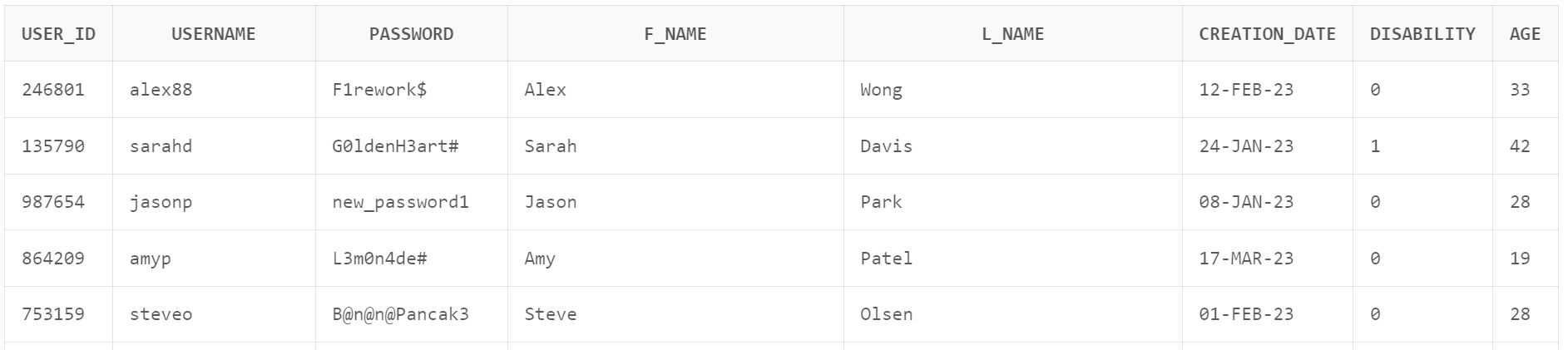
# **Final ERD**



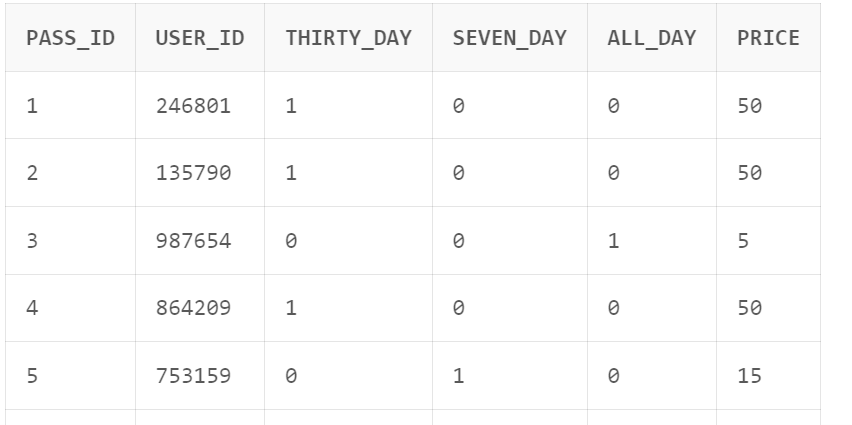
The Passenger table shows when and where as well as how passengers have been scanned into a bus. The Tickets table shows what types of tickets the users bought. The Pass table shows what kind of pass was bought by users. The User Account table holds information about user accounts to purchase tickets or passes. The Bus table holds information about the bus's location and service. The City table holds information about where each city is located. The Routes table shows the direction and distance as well as how many stops are in each route. The Stop table shows what bus stop at a stop location and where the stops are located as well as when buses arrive and depart.

# **First Five Rows of Created Tables**

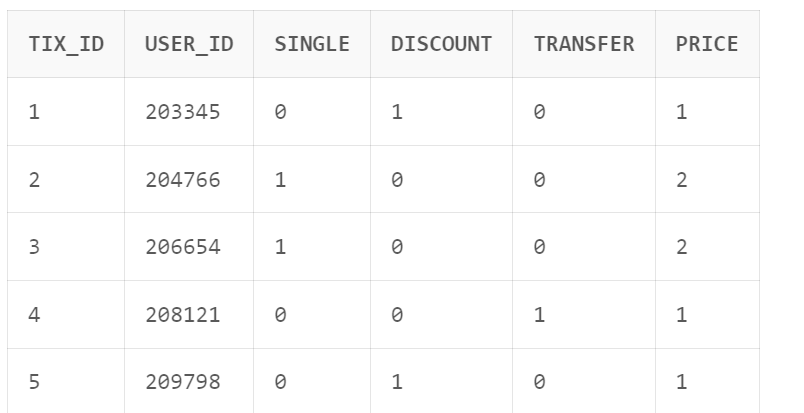
## User Accounts



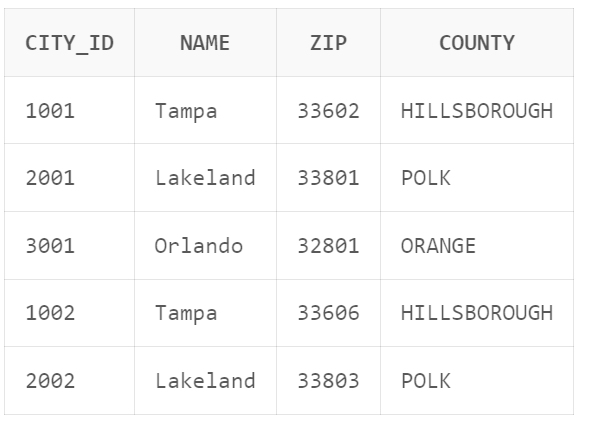
## Pass



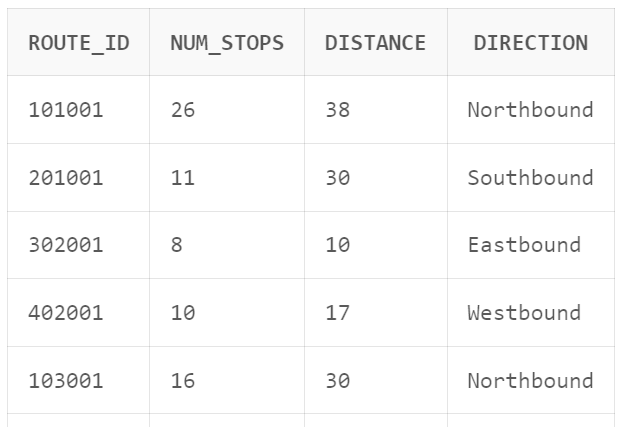
## Ticket

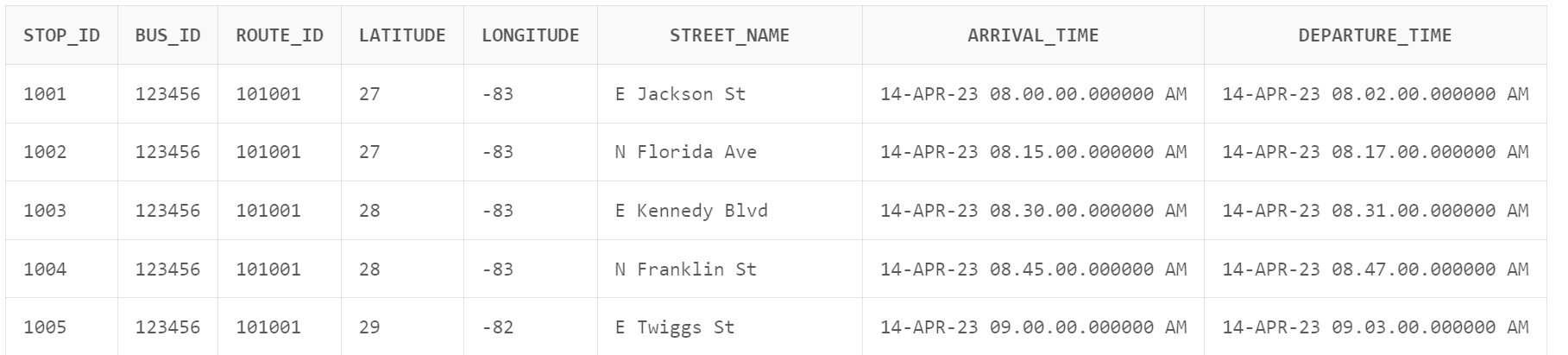
****

## City

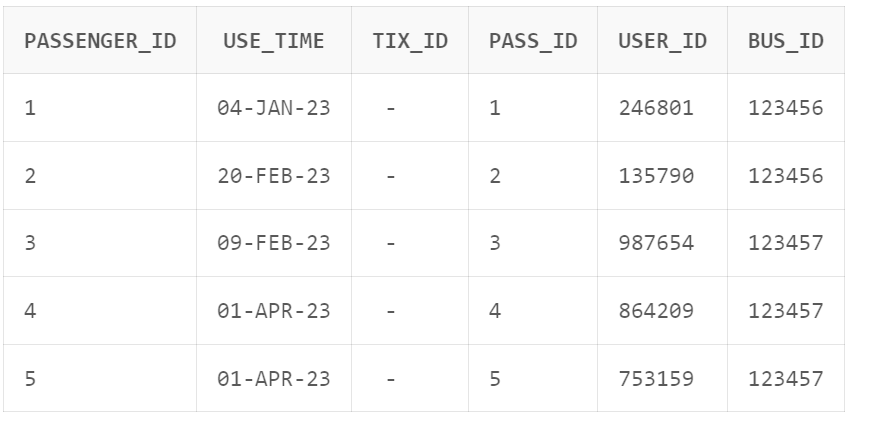


## Route



Stops

Passengers



# **Updates**

Note: All updates have a select statement before completing the update to show the differences with the update.

1.

This code selects the User ID that equal 987654 before and after the update to show the change. Then we updated this user’s password to ‘new\_password1’ and replaced the age to age 28.

SELECT \* FROM USER\_ACCOUNT WHERE USER\_ID = 987654;

UPDATE USER\_ACCOUNT SET PASSWORD = 'new\_password1' WHERE USER\_ID = 987654;

UPDATE USER\_ACCOUNT SET AGE = 28 WHERE USER\_ID = 987654;

SELECT \* FROM USER\_ACCOUNT WHERE USER\_ID = 987654;





2.

This code changes a particular user’s pass, 135790, from all day to thirty day and adjusts the price of the pass since it was upgraded.

SELECT \* FROM PASS WHERE USER\_ID = 135790;

UPDATE PASS SET PRICE = 50 WHERE USER\_ID = 135790;

UPDATE PASS SET THIRTY\_DAY = 1, ALL\_DAY = 0 WHERE USER\_ID = 135790;

SELECT \* FROM PASS WHERE USER\_ID = 135790;

Table, calendar

Description automatically generated

Table

Description automatically generated

3.

This update is applying a discount (DISCOUNT = 1, SINGLE = 0 means there’s a discount to the single ticket and vice versa) where the user’s disability status (from USER\_ACCOUNT) is subqueried to then find their user ID from USER\_ACCOUNT. Once that’s selected, the price for those with a disability that purchased a single ticked changes from 2 to 1 and then the whole ticket table is selected.

SELECT \* FROM TICKET;

UPDATE TICKET SET DISCOUNT = 1, SINGLE = 0 WHERE USER\_ID IN (SELECT USER\_ID FROM USER\_ACCOUNT WHERE DISABILITY = 1);

UPDATE TICKET SET PRICE = 1 WHERE DISCOUNT = 1;

SELECT \* FROM TICKET;

Table

Description automatically generated with medium confidence

Table

Description automatically generated

4.

This code is updating the CITY table by setting where the city name is Tampa, the county is set to Hillsborough and where the CITY\_ID is 02002 the ZIP is set to 33803.

SELECT \* FROM CITY;

UPDATE CITY SET COUNTY = 'HILLSBOROUGH' WHERE NAME = 'Tampa';

UPDATE CITY SET ZIP = 33803 WHERE CITY\_ID = 002002;

SELECT \* FROM CITY;

Table

Description automatically generated

Table

Description automatically generated

5.

This updates two rows in the routes table. It updates route, 302001’s number of stops. It also updates the distance and direction of any route that has less than 25 stops. Finally, it shows this these changes by selecting routes 101001 and 302001.

SELECT \* FROM ROUTES WHERE ROUTE\_ID = 101001 OR ROUTE\_ID = 302001;

UPDATE ROUTES SET NUM\_STOPS = 8 WHERE ROUTE\_ID = 302001;

UPDATE ROUTES SET DISTANCE = 38, DIRECTION = 'Northbound' WHERE NUM\_STOPS > 25;

SELECT \* FROM ROUTES WHERE ROUTE\_ID = 101001 OR ROUTE\_ID = 302001;

Table

Description automatically generated

Table

Description automatically generated

6.

Here we update bus 123456’s service date to 04-FEB-23. Also we update the mileage to 95,000 and the city ID to 001001 for the bus which is driven by driver 000001

SELECT \* FROM BUS WHERE BUS\_ID = 123456;

UPDATE BUS SET SERVICE\_DATE = '04-FEB-23' WHERE BUS\_ID = 123456;

UPDATE BUS SET MILEAGE = 95000, CITY\_ID = 001001 WHERE DRIVER\_ID = 000001;

SELECT \* FROM BUS WHERE BUS\_ID = 123456;

Table

Description automatically generated

Table

Description automatically generated

7.

This code selects all the columns of the row where STOP\_ID is equal to 002011. It then updated the Arrival\_Time and Departure\_Time of those rows. Then, the code updates the longitude and latitude of the columns where STOP\_ID is equal to 002011.

SELECT \* FROM STOPS WHERE STOP\_ID = 002011;

UPDATE STOPS SET Arrival\_Time = '03-MAY-23 8:01:00 AM', Departure\_Time = '03-MAY-23 8:04:00 AM' WHERE STOP\_ID = 002011;

UPDATE STOPS SET LATITUDE = '28.042535', LONGITUDE = '-81.955435' WHERE STOP\_ID = 002011;

SELECT \* FROM STOPS WHERE STOP\_ID = 002011;





8.

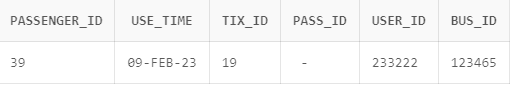
This code selects the passengers that have the passenger id of 39 and sets the time use\_time to be '03-MAY-23' as well as the bus id to 123464 of said passenger compared to what it was before.

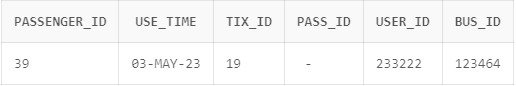
SELECT \* FROM PASSENGER WHERE PASSENGER\_ID = 39;

UPDATE PASSENGER SET USE\_TIME = '03-MAY-23' WHERE PASSENGER\_ID = 39;

UPDATE PASSENGER SET BUS\_ID= 123464 WHERE PASSENGER\_ID = 39;

SELECT \* FROM PASSENGER WHERE PASSENGER\_ID = 39;





# **Constraint Checks**

-- Referential Constraint

INSERT INTO TICKET (TIX\_ID, USER\_ID, SINGLE, DISCOUNT, TRANSFER, PRICE)

VALUES (1, 1, 1, 0, 0, 2);

The referential constraint in the code is occurring because of the foreign key constraint defined in the TICKET table. The FOREIGN KEY constraint ensures that the values in the USER\_ID column of the TICKET table match the values in the USER\_ID column of the USER\_ACCOUNT table.

In this case, the INSERT statement is trying to insert a record with a USER\_ID value of 1 in the TICKET table. However, there must be a corresponding record with a USER\_ID value of 1 in the USER\_ACCOUNT table, otherwise the foreign key constraint will be violated, and the INSERT statement will fail.

-- Check Constraint

INSERT INTO USER\_ACCOUNT (USER\_ID, USERNAME, PASSWORD, F\_NAME, L\_NAME, CREATION\_DATE, DISABILITY, AGE)

VALUES (1, 'johndoe', 'password', 'J0hn', 'Doe', '2022-01-01', 0, 30);

The check constraint in the code is occurring because of the PASSWORD column constraint in the USER\_ACCOUNT table. The CHECK constraint ensures that the PASSWORD value meets certain complexity requirements, including being at least 8 characters long and containing at least one special character.

In this case, the INSERT statement is trying to insert a record with a PASSWORD value of 'password' in the USER\_ACCOUNT table. However, this value does not meet the complexity requirements defined in the CHECK constraint, and therefore the INSERT statement will fail.

-- Not Null Constraint

INSERT INTO CITY(CITY\_ID, NAME, ZIP, COUNTY)

VALUES(4, NULL, ‘33647’, ‘Hillsborough’);

This query is trying to insert data into the CITY table, and the NOT NULL constraint is being applied to the NAME column to ensure that it is not NULL. However, the insert statement is trying to insert a NULL value for the NAME column, which violates the NOT NULL constraint, and hence, the insert statement will fail with an error.

# **Queries**

These are the questions we will query in our database. In general, these questions can be used for predictive analysis in the future to determine required capacities of the bus system.

1. How many users sign up each month?
2. How many disabled passengers have ridden each bus?
3. How many buses are in each city?
4. What is the average age of all users?
5. How many disabled passengers have ridden each bus?
6. How many people have passes and which pass do they have?
7. How many passengers does the route with the most stops have?
8. What is the total count of buses with passengers and the total number of passengers on those buses in each city?​
9. How many users sign up each month?

This could be useful to identify seasonality trends in bus usage​.

This code extracts the month from the CREATION\_DATE attribute then saves it into Month\_Created. It then also extracts the number of accounts created in those months and sorts them by descending order.

SELECT EXTRACT(MONTH FROM CREATION\_DATE) AS month\_created, COUNT(\*) AS num\_accounts\_created

FROM USER\_ACCOUNT

GROUP BY EXTRACT(MONTH FROM CREATION\_DATE)

ORDER BY num\_accounts\_created DESC;

**Output:**

Table

Description automatically generated

1. How many disabled passengers have ridden each bus?

This could be used to estimate which routes are popular among disabled passengers to better accommodate them.

This code selects the Passengers Bus ID's and counts them if the passenger is disabled. It then joins tables and displays all the busses that have disabled passengers that have ridden on them.

SELECT p.BUS\_ID, COUNT(CASE WHEN u.DISABILITY = 1 THEN 1 ELSE NULL END) AS DISABLED\_PASSENGERS

FROM PASSENGER p

JOIN USER\_ACCOUNT u ON p.USER\_ID = u.USER\_ID

JOIN BUS b ON p.BUS\_ID = b.BUS\_ID

GROUP BY p.BUS\_ID

ORDER BY DISABLED\_PASSENGERS DESC;

**Output:**

Table

Description automatically generated

1. How many buses are in each city?

This could be used to estimate costs associated with buses in each city such as labor, gas, bus purchases, etc.

This code selects the name of cities and counts how many unique busses are assigned to that city. The code then joins tables and groups them by city name. The code then displays cities and how many busses belong to them.

SELECT CITY.NAME, COUNT(BUS.BUS\_ID) AS BUS\_COUNT

FROM CITY

JOIN BUS ON CITY.CITY\_ID = BUS.CITY\_ID

GROUP BY CITY.NAME

ORDER BY BUS\_COUNT DESC;

**Output:**

Table

Description automatically generated

1. What is the average age of all users?

It is good to know the demographics of the users in the transportation system to know the basic understanding of the community of bus riders to notice changes and trends in the future.

This averages the ages listed in the User Account table.

SELECT AVG(age) AS avg\_age FROM user\_account;

**Output:**

Calendar

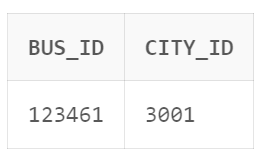
Description automatically generated with medium confidence

SELECT BUS\_ID, CITY\_ID

FROM BUS

WHERE MILEAGE = (SELECT MAX(MILEAGE) FROM BUS);

**Output:**



1. How many people have passes and which pass do they have?

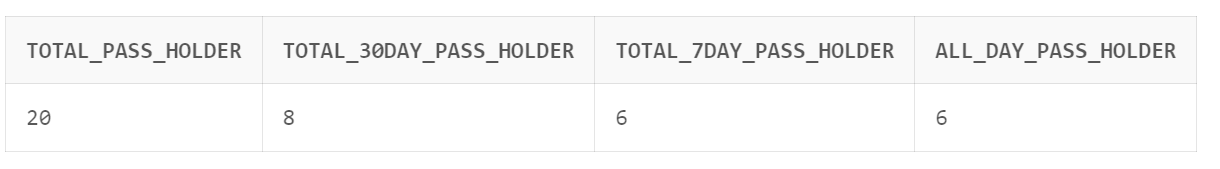
Passes and tickets are the bus system’s way of making money. It is important to analysis the amount of people who purchase passes and to see which passes are popular to create better strategies to increase revenue.

This query selects the count of all passes, and then sums the different types of passes individually from the Pass table.

SELECT COUNT(P.USER\_ID) AS TOTAL\_PASS\_HOLDER, SUM(P.THIRTY\_DAY) AS TOTAL\_30DAY\_PASS\_HOLDER, SUM(P.SEVEN\_DAY) AS TOTAL\_7DAY\_PASS\_HOLDER, SUM(P.ALL\_DAY)

AS ALL\_DAY\_PASS\_HOLDER FROM PASS P;

**Output:**

****

1. How many passengers does the route with the most stops have?

Discovering how many disabled passengers have ridden each bus, could be used to better accommodate to passengers for this specific route.

The code counts the number of passengers. The tables are joined and selects the table there the number of routes is maximum then displays the passenger count.

SELECT R.ROUTE\_ID, COUNT(\*) AS num\_passengers FROM PASSENGER P JOIN BUS B ON P.BUS\_ID = B.BUS\_ID

JOIN ROUTES R ON R.ROUTE\_ID = B.ROUTE\_ID WHERE R.NUM\_STOPS = (SELECT MAX(NUM\_STOPS) FROM ROUTES) GROUP BY R.ROUTE\_ID, R.NUM\_STOPS;

**Output:**

Table, calendar

Description automatically generated

1. What is the total count of buses with passengers and the total number of passengers on those buses in each city?​

The shown query will help with determining the use of our busses and whether we can expand or not, whether that be the number of buses or the capacity of our buses.

The query selects the distinct count of Bus IDs and counts the total number of passengers among those buses per city. These were selected from four joined tables where the City\_ID is matched with the Bus and City table based off the needed link where the Bus\_ID matched between the Bus and Passenger table which is then finally grouped by city name.

SELECT CITY.NAME, COUNT(DISTINCT(BUS.BUS\_ID)) AS NUM\_BUSES , COUNT(PASSENGER.PASSENGER\_ID) AS NUM\_PASSENGERS

FROM CITY JOIN BUS ON CITY.CITY\_ID = BUS.CITY\_ID JOIN ROUTES ON BUS.ROUTE\_ID = ROUTES.ROUTE\_ID

JOIN PASSENGER ON BUS.BUS\_ID = PASSENGER.BUS\_ID GROUP BY CITY.NAME;

**Output:**

Table

Description automatically generated

# **Conclusion**

In this project we attempted to create a functional bus transportation system database. Through this project we were able to find the most important pieces when it comes to storing data for a transportation system like this one. Those pieces being those attributes of who are riding your transportation system and the vehicles you are distributing throughout multiple cities, routes, and stops. Using complex queries and diagrams to showcase the frame for our designed system, we believe we came up with a good starting point for a system that can be used in the real-world making data collection for any bus transportation system seamless.