Philipps Universität Marburg

Datenintegration – 2nd. Phase

Johannes Gesk

Niklas Standop

UCC Discovery Steps

Step 1

In [185]:
```python
# step 1: join actual trips and stop_times
r1 = pd.merge(actual_data['stop_times'], actual_data['trips'], how='left', left_on=['Tr
r1
```

Out[185]:

	TripId	StopId	StopSequence	ArrivalDelay	ArrivalTime	DepartureDelay	DepartureTime
0	254163638	9057862	NaN	NaN	NaN	0.0	NaN
1	282351984	9013478	NaN	NaN	NaN	0.0	NaN
2	282385362	9058008	5.0	-60.0	NaN	0.0	NaN
3	264505093	9050640	11.0	30.0	NaN	30.0	NaN
4	282384857	9049320	NaN	NaN	NaN	0.0	NaN
...	...	...	...	...	...	...	...
2527	282384687	9044660	NaN	NaN	NaN	0.0	NaN
2528	282181087	9046671	NaN	NaN	NaN	0.0	NaN
2529	282351593	8000049	NaN	NaN	NaN	0.0	NaN
2530	282181685						
2531	282181070						

2532 rows × 12 colu

Step 2/3

In [192]:
```python
# step 2a: join routes and agency (target data)
r2 = pd.merge(target_data['routes'], target_data['agency'], how='left', left_on=['agenc
r2.head(3)
```

Out[192]:

	route_id	agency_id	route_short_name	route_type	agency_name
0	71026	1060	SEV24	1	S-Bahn Hamburg
1	71025	1060	SEV10	1	S-Bahn Hamburg
2	70978	1060	SEV21	2	S-Bahn Hamburg

In [193]:
```python
# step 2b: join the two relations using the Route Id
r3 = pd.merge(r1, r2, how='left', left_on=['RouteId'], right_on=['route_id'])
r3.head(3)

# finally, it is possible to...
# 1. determine the delay according to different agencies
# 2. determine the delay according to means of transportation
```

Out[193]:

	TripId	StopId	StopSequence	ArrivalDelay	ArrivalTime	DepartureDelay	DepartureTime	Sch
0	254163638	9057862	NaN	NaN	NaN	0.0	NaN	
1	282351984	9013478	NaN	NaN	NaN	0.0	NaN	
2	282385362	9058008	5.0	-60.0	NaN	0.0	NaN	

In [194]:
```python
# step 3: join TARGET trips and stop_times
r4 = pd.merge(target_data['stop_times'], target_data['trips'], how='left', left_on=['tr
r4.head(3)
```

Out[194]:

	trip_id	arrival_time	departure_time	stop_id	stop_sequence	pickup_type	drop_off_type	rout
0	282300445	23:20:00	23:20:00	8002557	0	0	0	71
1	282300445	23:25:00	23:25:00	2804001	1	0	0	71
2	282300445	23:30:00	23:30:00	2047308	2	0	0	71

Step 4

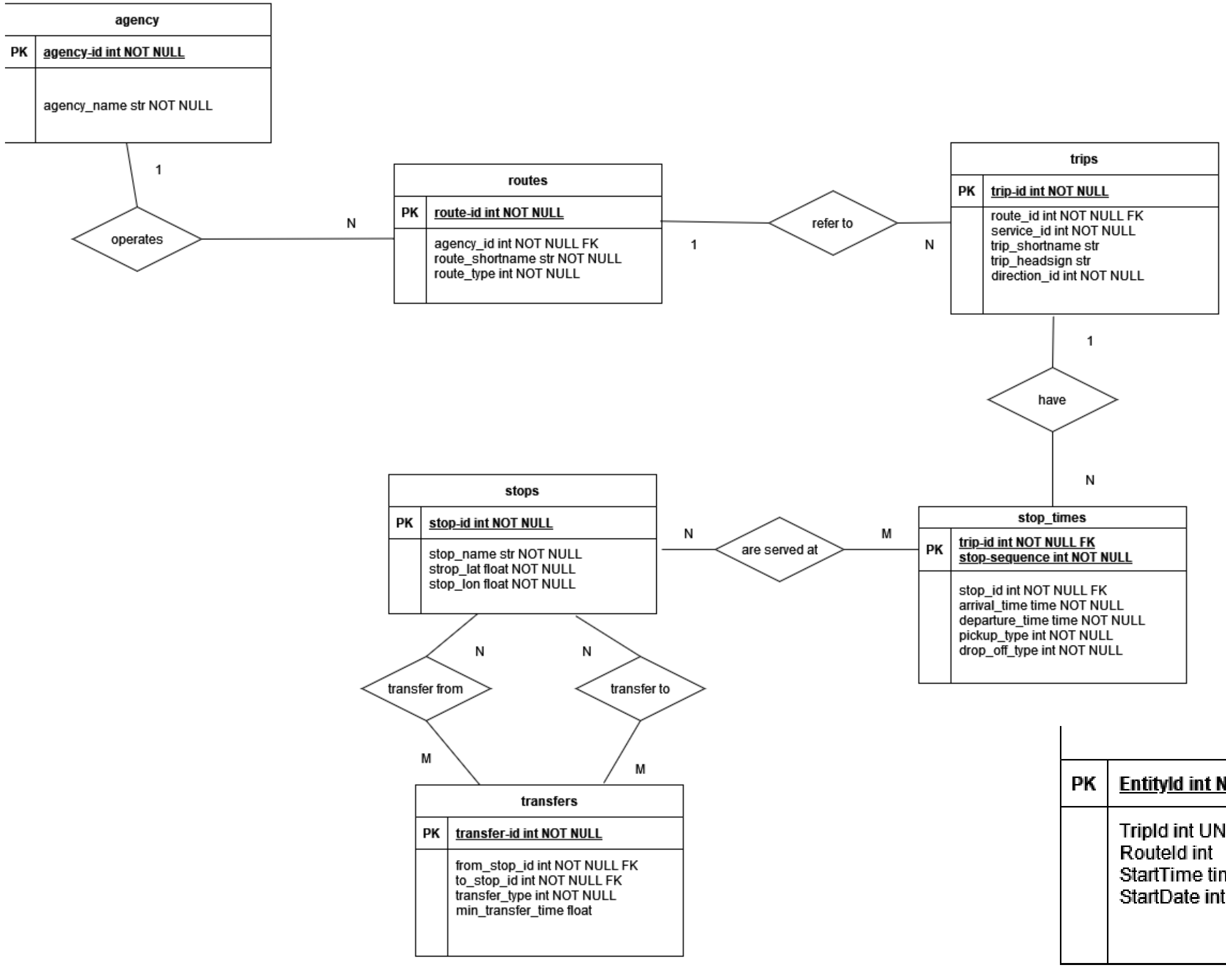
In [195]:
```python
# step 4: join r1 and r4 on Trip Id and Stop Id
# their schemata match for the most part
# however, using a left join enables to get all delays even if there is no join match
# (possible because there is no inclusion dependency)
r5 = pd.merge(r1, r4, how='inner', left_on=['TripId', 'StopId'], right_on=['trip_id', '
r5
```

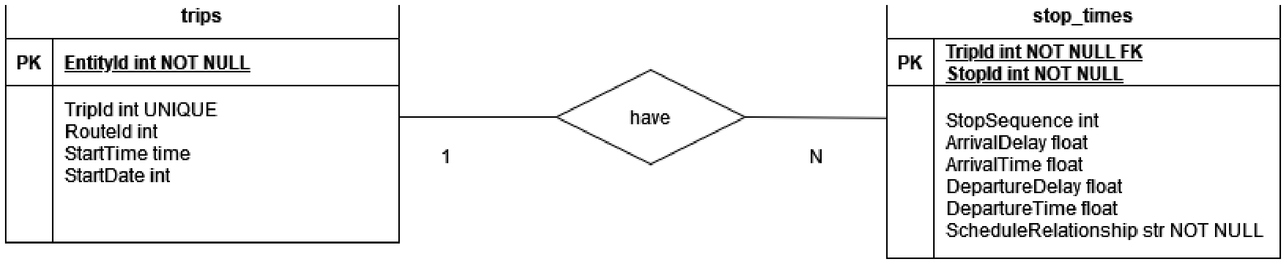
Out[195]:

	TripId	StopId	StopSequence	ArrivalDelay	ArrivalTime	DepartureDelay	DepartureTime
0	264505093	9050640	11.0	30.0	NaN	30.0	NaN
1	282384857	9049320	NaN	NaN	NaN	0.0	NaN
2	282188083	9090291	NaN	NaN	NaN	0.0	NaN
3	282351771	9023280	7.0	60.0	NaN	0.0	NaN
4	282184222	9049079	NaN	NaN	NaN	0.0	NaN
...	...	...	...	...	...	...	...
1722	282347652	9066582	NaN	NaN	NaN	0.0	NaN
1723	279028155	9014285	NaN	NaN	NaN	0.0	NaN
1724	279027722	9014184	NaN	NaN	NaN	0.0	NaN
1725	282384687	9044660	NaN	NaN	NaN	0.0	NaN
1726	282351593	8000049	NaN	NaN	NaN	0.0	NaN

1727 rows × 24 columns

Current status on ER model





Data profiling – functional dependencies

Data profiling – functional dependencies

```python
In [ ]: # Sample data

dir_target = 'target_data_vbn'
# Sample data
file_trips = f'{dir_target}\\trips.txt'
file_stops = f'{dir_target}\\stops.txt'
file_times = f'{dir_target}\\stop_times.txt'
file_agency = f'{dir_target}\\agency.txt'
file_routes = f'{dir_target}\\routes.txt'
file_transfers = f'{dir_target}\\transfers.txt'

data = [
    [file_trips],
    [file_stops],
    [file_times],
    [file_agency],
     [file_routes],
    [file_transfers]
]

# Discover functional dependencies
functional_dependencies = []

# Get attribute names
attributes = data[0]

# Iterate over each attribute
for i, attr in enumerate(attributes):
    attr_values = [row[i] for row in data[1:]]

    # Iterate over each attribute combination
    for j, comb in enumerate(attributes):
        if j != i:
            comb_values = [row[j] for row in data[1:]]

            # Check if the combination is a functional dependency
            if all(val1 == val2 for val1, val2 in zip(attr_values, comb_values)):
                functional_dependencies.append((attr, comb))

# Print functional dependencies
for fd in functional_dependencies:
    print(f"{fd[0]} -> {fd[1]}")
```

Data profiling – Benfords Law Analysis

```python
]

# Load data
#data_file = 'target_data_vbn_modified'
# Load data
#data = pd.read_csv('your_data_file.csv')  # Replace 'your_data_file.csv' with your actual data file

# Extract the first digit from the data
first_digit = []
with open(data, 'r') as file:
    reader = csv.reader(file)
    for row in reader:
        if len(row) > 0:
            first_digit.append(int(str(row[0]).strip()[0]))

# Calculate the frequency of leading digits
benford_freq = np.log10(1 + 1 / np.arange(1, 10))

# Calculate the observed frequency of leading digits
observed_freq = np.histogram(first_digit, bins=np.arange(1, 11), density=True)[0]

# Plot the results
plt.figure(figsize=(10, 6))
plt.bar(range(1, 10), observed_freq, label='Observed')
plt.plot(range(1, 10), benford_freq, 'r-', label='Benford')
plt.xlabel('Leading Digit')
plt.ylabel('Frequency')
plt.title('Benford\'s Law Analysis')
plt.legend()
plt.show()
```

Next up:

Find further algorithms to get the solution for the showcase

Integration into showcase

Gathering further datasets