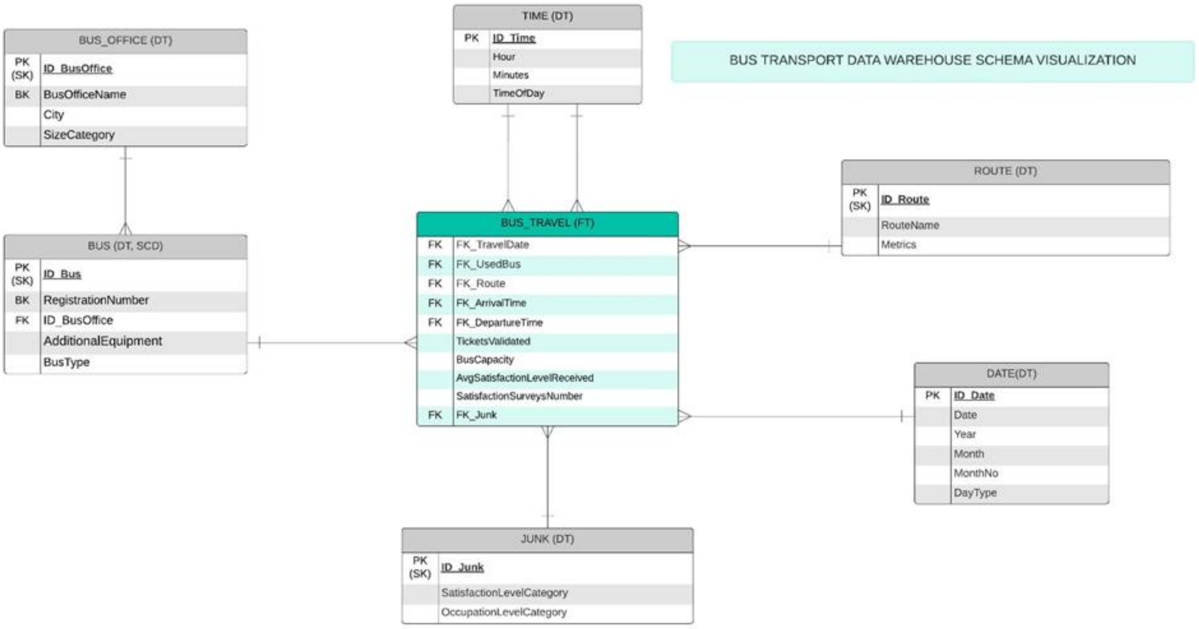
Bus transport Warehouse Design in Hive

Scenarios and explanation of the decisions made



Competency questions and scenarios

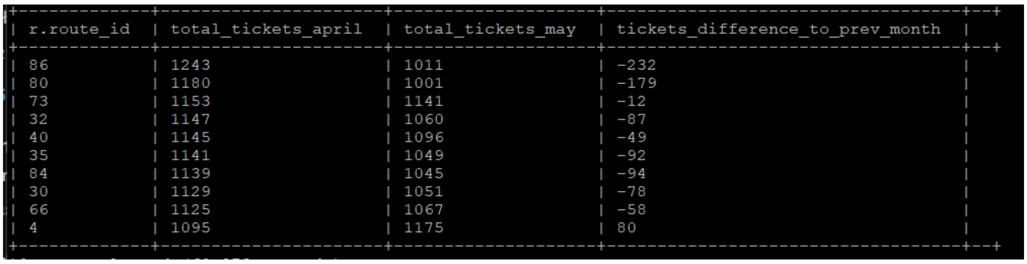
1. **What are the weekday routes with the highest number of tickets validated?(comparing two months to each other)**

   Scenario: A transport manager is analyzing weekday bus operations to identify the most profitable or popular routes, in order to allocate more buses for these routes.

   Partitioning by day_type (weekday) and month_no allows the query to efficiently filter for weekdays in given months.

   Bucketing on route_id helps speed up the join between travel and route tables.

```
SELECT r.route_id, SUM(CASE WHEN d.month_no = 4 THEN
t.tickets_validated ELSE 0 END) AS total_tickets_april,
SUM(CASE WHEN d.month_no = 5 THEN t.tickets_validated ELSE 0
END) AS total_tickets_may, (SUM(CASE WHEN d.month_no = 5 THEN
t.tickets_validated ELSE 0 END)-SUM(CASE WHEN d.month_no = 4
THEN t.tickets_validated ELSE 0 END)) AS
tickets_difference_to_prev_month
FROM travel t
JOIN route r ON t.route_id = r.route_id
JOIN date_dim d ON t.travel_date = d.date_id
WHERE d.month_no IN (4, 5) AND d.day_type= 'weekday'
GROUP BY r.route_id
ORDER BY total_tickets_april DESC
LIMIT 10;
```

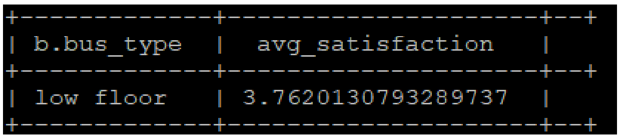


```
+--------------+------------------------+---------------------+-----------------------------------+--+
| r.route_id   | total_tickets_april    | total_tickets_may   | tickets_difference_to_prev_month  |
+--------------+------------------------+---------------------+-----------------------------------+--+
| 86           | 1243                   | 1011                | -232                              |
| 80           | 1180                   | 1001                | -179                              |
| 73           | 1153                   | 1141                | -12                               |
| 32           | 1147                   | 1060                | -87                               |
| 40           | 1145                   | 1096                | -49                               |
| 35           | 1141                   | 1049                | -92                               |
| 84           | 1139                   | 1045                | -94                               |
| 30           | 1129                   | 1051                | -78                               |
| 66           | 1125                   | 1067                | -58                               |
| 4            | 1095                   | 1175                | 80                                |
+--------------+------------------------+---------------------+-----------------------------------+--+
```

(case clause used in query: https://stackoverflow.com/questions/31489073/hive-else-in-sum)

2. **Which bus type has the highest average satisfaction level?**
   Scenario: The management team wants to understand which type of buses provide the best customer experience to decide on future bus purchases.
   Partitioning by bus_type in the bus table allows efficient retrieval of bus types (on bus_type we use static partitioning which ensures that each bus type has its own dedicated partition)
   Bucketing by bus_id helps speed up joins between the travel and bus tables.

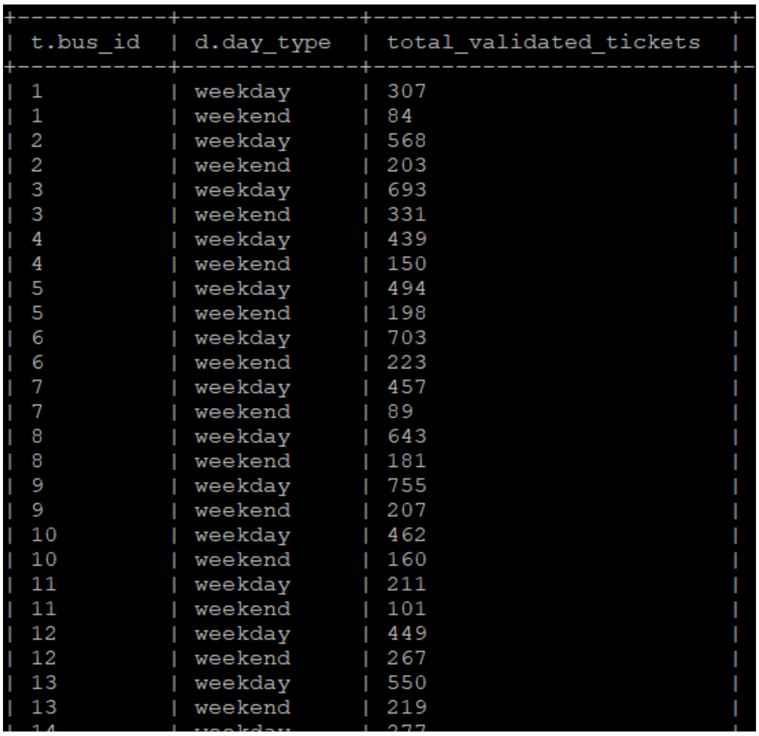
```
SELECT b.bus_type, AVG(t.avg_satisfaction_level_received) AS
avg_satisfaction
FROM travel t
JOIN bus b ON t.bus_id = b.bus_id
GROUP BY b.bus_type
ORDER BY avg_satisfaction DESC
LIMIT 1;
```



```
+-------------+---------------------+--+
| b.bus_type  |  avg_satisfaction   |
+-------------+---------------------+--+
| low floor   | 3.7620130793289737  |
+-------------+---------------------+--+
```

3. **What is the number of validated tickets for each bus comparing across weekdays and weekends in May?**
   Scenario: A data analyst is looking at bus usage in May to see which buses are used more on weekdays than at weekends to spot which buses have higher demand on weekdays versus weekends.
   Partitioning on by month_no and day_type for faster access only the relevant data for May and weekdays/weekends.
   Bucketing on bus_id in the travel table helps improve the efficiency of operations involving bus_id.

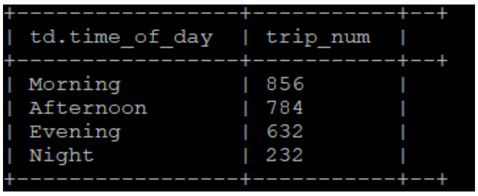
```
SELECT t.bus_id, d.day_type, SUM(t.tickets_validated) AS
total_validated_tickets
FROM travel t
JOIN date_dim d ON t.travel_date = d.date_id
WHERE d.month_no = 5 AND d.day_type IN ('weekday', 'weekend')
GROUP BY t.bus_id, d.day_type
ORDER BY t.bus_id, d.day_type;
```



```
+-----------+------------+--------------------------+-
| t.bus_id  | d.day_type | total_validated_tickets  |
+-----------+------------+--------------------------+-
| 1         | weekday    | 307                      |
| 1         | weekend    | 84                       |
| 2         | weekday    | 568                      |
| 2         | weekend    | 203                      |
| 3         | weekday    | 693                      |
| 3         | weekend    | 331                      |
| 4         | weekday    | 439                      |
| 4         | weekend    | 150                      |
| 5         | weekday    | 494                      |
| 5         | weekend    | 198                      |
| 6         | weekday    | 703                      |
| 6         | weekend    | 223                      |
| 7         | weekday    | 457                      |
| 7         | weekend    | 89                       |
| 8         | weekday    | 643                      |
| 8         | weekend    | 181                      |
| 9         | weekday    | 755                      |
| 9         | weekend    | 207                      |
| 10        | weekday    | 462                      |
| 10        | weekend    | 160                      |
| 11        | weekday    | 211                      |
| 11        | weekend    | 101                      |
| 12        | weekday    | 449                      |
| 12        | weekend    | 267                      |
| 13        | weekday    | 550                      |
| 13        | weekend    | 219                      |
| 14        | weekday    | 277                      |
```

4. **What was the busiest time of the day for bus routes on weekends in April?**
Scenario: The operations team is making changes to the weekend bus schedule in April to make better use of resources and reduce crowdes.
Partitioning by day_type (weekend) allows fast filtering for weekend records.
Partitioning by month_no helps quickly focus on April data.
Bucketing on route_id helps optimize joins between travel and route tables.

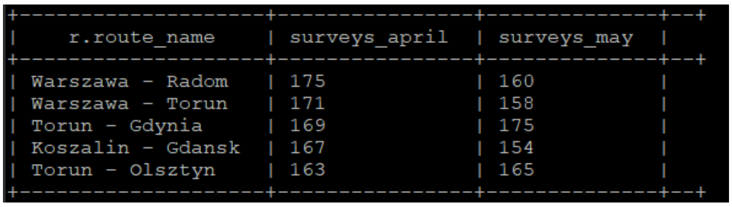
```
SELECT td.time_of_day, COUNT(*) AS trip_num
FROM travel t
JOIN date_dim d ON t.travel_date = d.date_id
JOIN time_dim td ON t.departure_time = td.time_id
WHERE month_no = 4 AND d.day_type = 'weekend'
GROUP BY td.time_of_day
ORDER BY trip_num DESC;
```



```
+-----------------+-----------+--+
| td.time_of_day  | trip_num  |
+-----------------+-----------+--+
| Morning         | 856       |
| Afternoon       | 784       |
| Evening         | 632       |
| Night           | 232       |
+-----------------+-----------+--+
```

5. **How many satisfaction surveys were received for a given route comparing two months (April and May)?**
   Scenario: The quality assurance team wants to check customer engagement by comparing the number of satisfaction surveys collected for a routes in April and May.
   Partitioning by month_no allows the query to quickly filter records for April and May.
   Bucketing on route_id helps optimize joins with the route table.

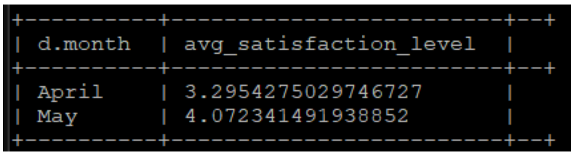
```
SELECT r.route_name, SUM(CASE WHEN d.month_no = 4 THEN
t.satisfaction_surveys_number ELSE 0 END) as surveys_april,
SUM(CASE WHEN d.month_no = 5 THEN
t.satisfaction_surveys_number ELSE 0 END) as surveys_may
FROM travel t
JOIN route r ON t.route_id = r.route_id
JOIN date_dim d ON t.travel_date = d.date_id
GROUP BY r.route_name
ORDER BY surveys_april DESC
LIMIT 5;
```



```
+--------------------+----------------+---------------+--+
|    r.route_name    | surveys_april  | surveys_may   |
+--------------------+----------------+---------------+--+
| Warszawa - Radom   | 175            | 160           |
| Warszawa - Torun   | 171            | 158           |
| Torun - Gdynia     | 169            | 175           |
| Koszalin - Gdansk  | 167            | 154           |
| Torun - Olsztyn    | 163            | 165           |
+--------------------+----------------+---------------+--+
```

6. **How satisfaction level of bus users changed over time?**
   Scenario: The quality assurance team wants to see how (for example) recent changes in service affect customer satisfaction.
   Partitioning by month_no helps quickly retrieve data for specific months.

```
SELECT d.month, AVG(t.avg_satisfaction_level_received) AS
avg_satisfaction_level
FROM travel t
JOIN date_dim d ON t.travel_date = d.date_id
GROUP BY d.month
ORDER BY d.month;
```



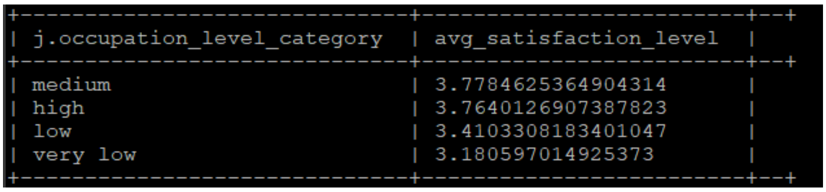
```
+----------+------------------------+--+
| d.month  | avg_satisfaction_level |
+----------+------------------------+--+
| April    | 3.2954275029746727     |
| May      | 4.072341491938852      |
+----------+------------------------+--+
```

7. **How does the satisfaction level vary based on occupation level categories?**

Scenario: A quality assurance team evaluates how passenger satisfaction levels change based on bus crowd levels to determine whether overcrowding impacts user satisfaction and identify areas for capacity improvement.

The junk table, which contains occupation categories, is small, so partitioning or bucketing isn't critical here.

```
SELECT j.occupation_level_category,
AVG(avg_satisfaction_level_received) AS avg_satisfaction_level
FROM travel t
JOIN junk j ON t.junk_id=j.junk_id
GROUP BY j.occupation_level_category
ORDER BY avg_satisfaction_level DESC;
```



```
+----------------------------+------------------------+--+
| j.occupation_level_category  | avg_satisfaction_level |
+----------------------------+------------------------+--+
| medium                     | 3.7784625364904314     |
| high                       | 3.764012690738782З     |
| low                        | 3.4103308183401047     |
| very low                   | 3.180597014925373      |
+----------------------------+------------------------+--+
```

8. **What is the breakdown of travel counts by bus type that departures in the morning?**

(how many buses departured in the morning until now and what bus type was it)

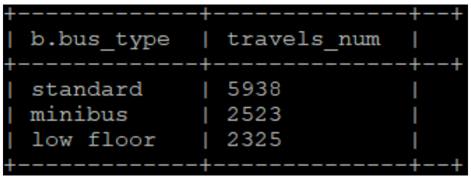
Scenario: The transport planner analyses the number of buses of each type that departures in the morning to identify patterns of use and to ensure a balanced use of the different bus types.

Partitioning by bus_type allows efficient filtering of travel records by bus type, partitioning by time_of_day (morning) filters morning travel data.

Bucketing by bus_id can help improve performance when joining the travel table with the bus table.

```
SELECT b.bus_type, COUNT(*) travels_num
FROM travel t
JOIN bus b ON t.bus_id=b.bus_id
JOIN time_dim tdim ON t.departure_time=tdim.time_id
WHERE tdim.time_of_day = 'Morning'
GROUP BY b.bus_type, tdim.time_of_day
```

```
ORDER BY travels_num DESC;
```



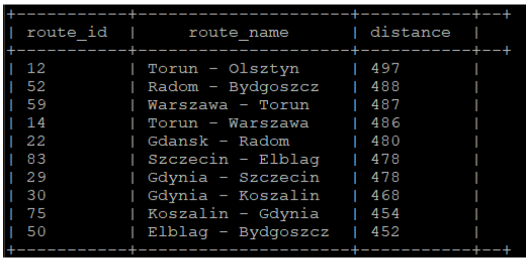
```
+-------------+-------------+--+
| b.bus_type  | travels_num  |
+-------------+-------------+--+
| standard    | 5938         |
| minibus     | 2523         |
| low floor   | 2325         |
+-------------+-------------+--+
```

9. **Which are the routes with the highest distance value?**

Scenario: Identifying long-distance routes for subsequent assessment of their viability, fuel consumption and resource requirements.

Mapping the distance_km field inside metrics['distance_km']. Because bucketing by route_id ensures that the data related to each route is grouped together, querying the metrics['distance_km'] map field becomes more efficient.

```
SELECT route_id, route_name, metrics['distance_km'] AS
distance
FROM route
ORDER BY distance DESC
LIMIT 10;
```



```
+-----------+--------------------+-----------+--+
| route_id  |     route_name     | distance  |
+-----------+--------------------+-----------+--+
| 12        | Torun - Olsztyn    | 497       |
| 52        | Radom - Bydgoszcz  | 488       |
| 59        | Warszawa - Torun   | 487       |
| 14        | Torun - Warszawa   | 486       |
| 22        | Gdansk - Radom     | 480       |
| 83        | Szczecin - Elblag  | 478       |
| 29        | Gdynia - Szczecin  | 478       |
| 30        | Gdynia - Koszalin  | 468       |
| 75        | Koszalin - Gdynia  | 454       |
| 50        | Elblag - Bydgoszcz | 452       |
+-----------+--------------------+-----------+--+
```

10. **What is the satisfaction level on weekend routes that have the highest avg duration? (in the selected month - April)**

Scenario: A quality control team looks at how customers feel about long weekend trips.

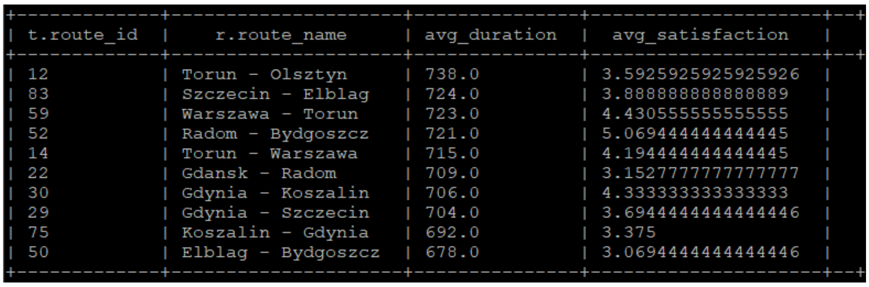
Partitioning by day_type (weekend) and month_no (April) allows for efficient filtering of month and day type.

Bucketing on route_id speeds up the join with the route table.

Mapping the duration_min in metrics for each route, and bucketing by route_id helps optimize the retrieval of metrics['duration_min']

```
SELECT t.route_id, r.route_name,
AVG(r.metrics['duration_min']) as avg_duration,
AVG(t.avg_satisfaction_level_received) AS avg_satisfaction
```

```
FROM travel t

JOIN route r ON t.route_id = r.route_id

JOIN date_dim d ON t.travel_date = d.date_id

WHERE d.month_no = 4 AND d.day_type = 'weekend'

GROUP BY t.route_id, r.route_name

ORDER BY avg_duration DESC

LIMIT 10;
```



```
+-------------+----------------------+----------------+----------------------+---+
| t.route_id  |     r.route_name     | avg_duration   |   avg_satisfaction   |
+-------------+----------------------+----------------+----------------------+---+
| 12          | Torun - Olsztyn      | 738.0          | 3.5925925925925926   |
| 83          | Szczecin - Elblag    | 724.0          | 3.888888888888889    |
| 59          | Warszawa - Torun     | 723.0          | 4.430555555555555    |
| 52          | Radom - Bydgoszcz    | 721.0          | 5.069444444444445    |
| 14          | Torun - Warszawa     | 715.0          | 4.194444444444445    |
| 22          | Gdansk - Radom       | 709.0          | 3.1527777777777777   |
| 30          | Gdynia - Koszalin    | 706.0          | 4.333333333333333    |
| 29          | Gdynia - Szczecin    | 704.0          | 3.6944444444444446   |
| 75          | Koszalin - Gdynia    | 692.0          | 3.375                |
| 50          | Elblag - Bydgoszcz   | 678.0          | 3.0694444444444446   |
+-------------+----------------------+----------------+----------------------+---+
```

1. Partitioning:

Used for tables where queries often filter by specific columns like month_no, day_type, time_of_day, travel_date, bus_type. It limits the number of nodes that needs to be scanned when filtering some columns.

Partitions in tables are comparatively equal size (travel_date in travel table as the schedule of buses does not vary significantly on different days) or the values possible for each field is limited (like month_no, day_type, time_of_day and bus_type). Static partitioning is used for bus_type where we manually decide on number of partitions and to which partition data will be loaded, we need to be sure that we put data into right partition e.g. loading buses of only one type at once! For dynamic paritioning (month_no, day_type, time_of_day, travel_date) - partitions are created automatically depending on data we are inserting into, what for travel_date field adding records while manually giving the date may be more complex process, probably this table will grow large in time.

2. Bucketing:

Bucket by route_id, bus_is travel table to improve performance of aggregations and joins. Partitioning here might lead to some larger and some smaller partitions as we assume that some routes have more travels – we use buckets to decompose dataset based on value from hash function. As the number of travels is around 20k (data of travels is from April) we can also imply that the number of records in the datawarehouse will arise at most to 240k – when storing data for one year, and we have 90 unique routes we decided on splitting data into 50 buckets.

Bucketing also on route_id in routes table improves joins and searching on this column, we have 90 routes so splittinig into 10 buckets is reasonable (on average 9 routes per bucket). In queries the route_id field (what is unique attribute) will be used often in queries (e.g. searching for specific routes usually combined with travel table) so map-side joins will work faster on bucketing tables.

Bucketing on bus_id into 10 buckets and date_id into 30 buckets – for joins with travel table.

No partitioning/bucketing on junk table as it has only 12 records.

3. Data Format:

ORC – used for routes and buses as it support complex datatypes (in routes table map, in bus array) as well as queries that are using aggregations (travel is join-heavy with other tables).

PARQUET – used for date, time, junk tables, frequently queried for filtering/grouping (day_type, time_of_day). Those tables are realatively small tables. Parquet avoids reading unnecessary columns during query execution (columnar storage).

TEXTFILE – used for bus_office table; as it may be easier to use for external systems. (reading data in Excel, Python or R) No partitioning here due to relatively small table.