

# **Exploring Vehicle Trajectory Data using MobilityDB**



**By**

**Raees Ahmed**

**(2020-NUST-MS-GIS-330057)**


**A thesis submitted in partial fulfillment of the requirements for the  
degree of Master of Sciences in Remote Sensing and GIS**

**Institute of Geographical Information Systems  
School of Civil and Environmental Engineering  
National University of Sciences & Technology  
Islamabad, Pakistan**

**October 2022**

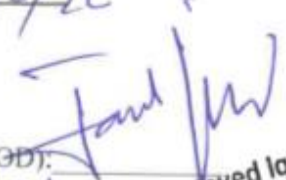
### THESIS ACCEPTANCE CERTIFICATE

Certified that final copy of MS/MPhil thesis written by **Raees Ahmed (Registration No. MSRSGIS 00000330057), of Session 2020 (Institute of Geographical Information System)** has been vetted by undersigned, found complete in all respects as per NUST Statutes/Regulation, is free of plagiarism, errors, and mistakes and is accepted as partial fulfillment for award of MS/MPhil degree. It is further certified that necessary amendments as pointed out by GEC members of the scholar have also been incorporated in the said thesis.

Signature: 

Name of Supervisor: Dr Ali Tahir

Date: 21/10/22

Signature (HOD): 

Date: 21/10/22 Dr. Javed Iqbal  
Professor & HOD IGIS, SCEE (NUST)  
H-12, Islamabad


Signature (Principal): 

Date: 03 AUG 2023

PROF DR MUHAMMAD IRFAN  
Principal & Dean  
SCEE, NUST

**National University of Sciences & Technology****MASTER THESIS WORK**

We hereby recommend that the dissertation prepared under our supervision by: Mr Raees Ahmed (Reg # 00000330057) Titled: **“Exploring Vehicle Trajectory Data Using MobilityDB”** be accepted in partial fulfillment of the requirements for the award of MS degree with (B+) grade.


**Examination Committee Members**1. Name: Dr. Salman AtifSignature: 2. Name: Dr. Muhammad Tariq SaeedSignature: Supervisor's Name: Dr. Ali TahirSignature: Date: 6 Sep 2022

  
**Dr. Javed Iqbal**  
 Head of Department & HOD IGIS, SCEE (NUST)  
 Islamabad

Head of Department

6/9/2022  
 Date

**COUNTERSIGNED**Date: 03 AUG 2023

  
 Principal & Dean SCEE  
**PROF DR MUHAMMAD IRFAN**  
 Principal & Dean  
 SCEE, NUST

## ACADEMIC THESIS: DECLARATION OF AUTHORSHIP

I, **Raees Ahmed**, declare that this thesis and the work presented in it are my own and have been generated by me as the result of my own original research.

### **Exploring Vehicle Trajectory Data using MobilityDB**

I confirm that:

1. This thesis is composed of my original work, and contains no material previously published or written by another person except where due reference has been made in the text;
2. Wherever any part of this thesis has previously been submitted for a degree or any other qualification at this or any other institution, it has been clearly stated;
3. I have acknowledged all main sources of help;
4. Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself;
5. None of this work has been published before submission.
6. This work is not plagiarized under the HEC plagiarism policy.

Signed: .....

Date: .....

## **DEDICATION**

This dissemination is dedicated to my Late Dearest Mother. By the virtue of whose prays, I have been able to reach at this high position. My Late Father whose support and believe in me made me stand in front of every obstacle in my life. My wife and daughters whose support and courage after my parents made it possible to successfully continue this sacred path of knowledge. To my brothers and sisters for their unconditional support. May Allah bless them all with best reward.

## ACKNOWLEDGEMENTS

All praise to Allah Almighty, the most merciful, most forgiving and loves to forgive. To whom all the knowledge belongs for known and unknown. Who always blessed me with best of the best to achieve this knowledgeable landmark. This achievement was not possible without guidance under light of Prophet Mohammad (SAWA) saying “Seek knowledge from the cradle to the grave”.

Many thanks to my family who always believed in me and prayed for my success, especially my father who supported me throughout his life. When I was sad due to broke down of my old laptop during first semester, He gifted me new laptop to continue my MS degree on last evening of his life. Enormous thanks to my late mother. Without her prayers, I was never able to complete this journey. A prayer follows every moment your thought crosses my heart. I wish you both to be granted highest place in Jannah.

I would like to express my heartfelt appreciation to my supervisor Dr. Ali Tahir for patient guidance, advice and encouragement throughout this wonderful task. He always remained available to answer my queries promptly. Fortunately, I would also like to acknowledge Dr. Salman Atif and Dr. M Tariq Saeed for their kind support and guidance which gave me new ideas throughout the course of the study. I would especially thank Dr. Ejaz Ahmed for continuously motivating me to work harder and complete this research in an easy and optimized way. Getting mobility desired datasets from TPL Pvt Ltd was not an easy task without support from my supervisor Dr. Ali Tahir and my senior Mr. M Daud Kamal.

I am very thankful and pay my gratitude to all IGIS staff who always worked behind and arrange administrative support in time. I would like to acknowledge my fellow students Mr. Faqir Hussain, Mr. Asif and Mr. Alauddin for creating healthy discussions on the subject.

***Raees Ahmed***

# TABLE OF CONTENTS

CERTIFICATE .....	i
ACADEMIC THESIS: DECLARATION OF AUTHORSHIP .....	ii
DEDICATION .....	iii
ACKNOWLEDGEMENTS .....	iv
TABLE OF CONTENTS .....	v
LIST OF FIGURES .....	viii
LIST OF TABLES .....	x
ABSTRACT .....	xi
1 INTRODUCTION .....	1
1.1 Background Information .....	2
1.2 Rationale .....	3
1.3 Objectives .....	4
1.4 Scope of study .....	4
1.5 MobilityDB at Present .....	5
1.6 Mobility Data Properties (MDP) .....	6
1.7 Mobility Data Space (MDS) .....	8
1.8 Advantages of the MDS .....	9
1.8.1 Transparency and data sovereignty .....	9
1.8.2 Wide range of data from a single source .....	9
1.8.3 Easy to use .....	10

1.8.4	Data protection and security .....	10
1.9	Areas of application in Pakistan.....	11
1.10	MobilityDB Visualization.....	15
1.11	Challenges faced by MobilityDB.....	16
2	MATERIALS AND METHODS.....	19
2.1	Study Area.....	19
2.2	Methodology .....	19
2.3	Environment Building.....	20
2.4	Linux Installation .....	21
2.5	Windows Installation.....	24
2.6	Comparison of different MobilityDB versions .....	28
2.7	Data Loading to MobilityDB .....	29
2.8	Preprocessing .....	37
2.9	Generation of Trip ID.....	40
2.10	Database Schemas preparation.....	41
2.11	Creation of Point and Temporal Point.....	41
2.12	Data Cleaning.....	43
2.13	Creation of Trajectory .....	44
3	RESULTS AND DISCUSSIONS.....	49
3.1	Exploratory Data Analysis (EDA) .....	49
3.2	Histogram Analysis.....	54



3.3	Spatiotemporal Analyses.....	57
3.4	Applicability scenarios.....	57
4	CONCLUSIONS AND RECOMMENDATIONS .....	75
4.1	Conclusions .....	75
4.2	Recommendations for further research .....	77
5	REFERENCES .....	78
6	Appendix-1. Python Based Tool to Generate Trip ID. ....	80
7	Appendix-2. SQL curser to generate trip ID.....	81
8	Appendix-3. SQL schema to update columns.....	82
9	Algorithm 4: SQL schema to monthly hours driven.....	83

## LIST OF FIGURES

Figure 1.1. MobilityDB ecosystem.....	18
Figure 2.1. Methodology flow chart. ....	27
Figure 2.2. Study area map. ....	31
Figure 2.3. Month wise GPS points distribution.....	34
Figure 2.4. PostgreSQL ODBC driver configuration. ....	34
Figure 2.5. ODBC connection saved. ....	35
Figure 2.6. Import / Export wizard - data source selection.....	35
Figure 2.7. Data destination - DSN name specification.....	36
Figure 2.8. Table selection for migration to MobilityDB.....	36
Figure 2.9. Data migration from SQL to PostgreSQL.....	39
Figure 3.1. Record reduction with Trip generation.....	50
Figure 3.2. SQL to MobilityDB data compression ratio.....	51
Figure 3.3. Comparison between hours driven and actual trips.....	51
Figure 3.4. Three micro-mobility descriptors. ....	52
Figure 3.5. Comparison of total distance travelled with total trips performed.....	56
Figure 3.6. All active trips at specified time.....	60
Figure 3.7. Temporal range - QGIS plugin Move. ....	60
Figure 3.8. QGIS display of results Scenario 2. ....	62
Figure 3.9. Visual display of Scenario 2 result - Petrol filling .....	65
Figure 3.10. Trips crossing Burhan Interchange M1.....	66

Figure 3.11. Spatiotemporal locations of active vehicles. ....	67
Figure 3.12. Stepwise varying non-linear speed interval of trajectory. ....	69
Figure 3.13. Stepwise speed interval visualization. ....	69
Figure 3.14. Temporally active trips.....	71
Figure 3.15. Spatiotemporal K-NN display. ....	73
Figure 3.16. Visual display of scenario 08. ....	74

## LIST OF TABLES

Table 1.1. Key parameters of MoD. ....	7
Table 2.1. MobilityDB instance functional comparison. ....	30
Table 2.2. Meta Data. ....	32
Table 2.3. Monthly record details.....	32
Table 2.4. Data elements received.....	38
Table 2.5. Updated data elements.....	42
Table 2.6. Duplication removal query and utility algorithm. ....	45
Table 2.7. Outlier records.....	48
Table 3.1. Dataset description after data cleaning. ....	50
Table 3.2. Month wise histogram analysis.....	84
Table 3.3. Temporal range query output. ....	59
Table 3.4. Spatial range query results - Scenario 02. ....	62
Table 3.5. Trip crossing nearby given location ....	65
Table 3.6. Temporal location identification - Scenario 04. ....	67
Table 3.7. Result of spatiotemporal K-NN query ....	71
Table 3.8. Result set - Scenario 08 ....	73
Table 4.1. Application areas in Pakistan. ....	76

## ABSTRACT

MobilityDB, an extension to PostgreSQL and PostGIS, is used for the management of spatiotemporal mobility data. With the continuous growth in mobility datasets, utilization of MobilityDB is need of time for storage management and extraction of useful information from uncertain moving data. Due to complex architecture and configuration of MobilityDB on Debian operating systems and non-availability on Windows, the study focuses on three areas related to MobilityDB in Pakistan. (a) Exploration of MobilityDB using Abstract Data Types (ADT) and identification of various use cases in Pakistan. (b) Data compression by constructing trajectories from raw GPS logs (c) Spatiotemporal Trajectory analysis to produce decision support information followed by animated visualization with respect to Cluster and Frequent patterns. Big mobility datasets of TPL received in SQL format were migrated to MobilityDB followed by environment building. Preprocessing of mobility data was performed with respect to trips generation and outlier elimination. Generation of trips is performed with the help of vehicle statuses using custom build SQL cursor utility. Finally spatiotemporal trajectory analysis performed using MobilityDB and visualization of extracted information carried out using an open-source plugin named 'Move' in QGIS. To make this study self-contained and easy for readers, all areas were documented with configurational commands, source codes, spatiotemporal range and aggregate queries along with brief description. The results were concluded with the help of spatiotemporal queries followed by animated visualization of cluster and frequent mobility patterns. Results revealed that MobilityDB successfully implemented on big datasets reducing 99% data size thus increasing the database performance and optimization. The study also identified various applicability areas in Pakistan considering real-life use cases based upon TPL's mobility data.

## **1 INTRODUCTION**

Easy, flexible management and visualization of temporal geographic always remained need of time. Since 1980s many efforts have been contributed to manage and visualize temporal geographic data in computer systems. Based upon three components space, time and attributes, GIS modelers proposed several spatiotemporal models. The Spatiotemporal Object and Space Time Models can also represent state changes at space. The Temporal Map Sets (TMS) model, an extensions of snapshot model, proposed to manage geographic data in a space and time (Beller, 1991) environment. GIS and computer scientists had presented many theories and techniques to utilize temporal data along with relational datasets. MobilityDB is an advanced platform to manage, analyze and visualize temporal datasets. MobilityDB an extension of PostGIS and PostgreSQL, for handling spatiotemporal data, platform developed by Computer & Decision Engineering Department of the Université Libre de Bruxelles (ULB) with the monetary support by European Commission, Fonds de la Recherche Scientifique (FNRS), Belgium and Innoviris, Belgium. Project was developed by team of developers supervised by Project Steering Committee (PSC) members. MobilityDB works with PostgreSQL (Zimányi, 2020) and PostGIS (an add on to the database) along with QGIS (An open-source Geographic Information System) for mobility data visualization. Move is also an open-source plugin to QGIS for temporal visualization of mobility datasets (Schoemans, 2022). This study aims on highlighting issues faced by MobilityDB users while installation, utilization and upgradation followed by test scenarios with their proposed resolution guidelines.

In this regard a case study pertaining to mobility datasets of TPL insurance company of Pakistan will also be analyzed to help understand the power and utilization of MobilityDB in Pakistan. In this study we will present all phases of MobilityDB starting from installation / configuration, data loading, preprocessing and explorations onto MobilityDB and spatiotemporal analysis followed by trajectory construction.

## **1.1 Background Information**

Unfortunately, thousands of research work have been carried out on moving object databases in past decade but few of them remained focal point due to their prototype and very less such prototypes were used at commercial level (Güting et al., 2010). MobilityDB an extension to PostgreSQL and PostGIS is the main prototype which works on abstract data type. MobilityDB builds indexes and aggregation thus reducing the storage capacity with powerful data extraction capability (Zimányi et al., 2020). Demand of moving object data and its utilization in spatiotemporal database management is increasing day by day. Mobility objects generate huge amount of data in Realtime which require an efficient management system. Applications configured on PostgreSQL and PostGIS working in simple architecture face performance issues. To resolve such issues a scalable architecture of PostgreSQL is required which is achieved by Citus (an extension of Postgres). Citus enables PostgreSQL a distributed query processing allowing horizontal scalability (Bakli et al., 2019). Citus based Postgres architecture generally have one Coordinator node for managing metadata along with multiple worker nodes for storing and managing actual data. Both the Coordinator and worker nodes are of same application configuration except their roles. Client when execute some query is received at coordinator which is then

transferred to respective worker node for efficient processing (Cubukcu et al., 2021). Visualization / temporal animation of moving object data is the area still require lot of work. Move (Moving Objects Visual Exploration) is only an open source QGIS plugin used to query and visualize the temporal data from MobilityDB (Schoemans et al., 2022).

## **1.2 Rationale**

Large amount of multi-dimensional data is generated on daily basis by GPS equipped devices such as mobile phones, vehicle trackers and other electronic equipment. Many studies have already been done to use these data for commercial and security purposes by assessing user's movement pattern and their social behaviors. These big data required big data platform to analyze the data quickly and efficiently. Traditional database systems like My SQL, Oracle, MongoDB etc can easily accommodate large spatial datasets, but such database management system is still lacking with respect to temporal notion. MobilityDB is the only database system at present which not only provide efficient and optimized data storage but its sophisticated and continuously growing temporal mobility functions and views help manage large moving datasets. MobilityDB is an emerging platform also provide support to PostgreSQL platform along with maximum database adapter support.



### **1.3 Objectives**

Following are the objectives of the study:

- (a). To explore MobilityDB using Abstract Data Types (ADT) and identification of various use cases in Pakistan.
- (b). Performing data compression by constructing trajectories from raw GPS logs
- (c). To Accomplish spatiotemporal trajectory analysis to produce decision support information based upon the following:
  - Cluster Patterns
  - Frequent Patterns
  - Outlier Identification

### **1.4 Scope of study**

The work displayed in this research shows a proof of idea where GPS information examination can yield results that assistance numerous areas of decision making. It shows the utilization of GPS data produced by two thousand two hundred and sixty-one users. This information gathered can be considered to classify user behavior and describe interests of users on different levels. This knowledge can be used to personalize road safety, anticipation and security related issues to improve the user experience. It elaborates procedures adopted for processing the raw GPS data from collection till analysis. Some records can be created to generalize the process for use with any kind of mobility data. The GPS mobility data collected is constrained free and movement data is based on the behaviors of the user.

## **1.5 MobilityDB at Present**

MobilityDB aims in overcoming the limitations of SECONDO and HERMES while providing real world solutions to moving objects (Zimányi, 2019). MobilityDB is currently utilized for ground and sea level transport management, however, aerial utilization is still in its initial phases. Both data variants of General Transit Feed Specification (GTFS) have been analyzed using Mobility for Buenos Aires public transportation system (Godfrid, 2022). Many public and private offices in Pakistan still working on static object data structure like PostgreSQL or MongoDB to manage their spatial data. In contrast, MobilityDB presents four temporal data types (tbool, tint, tfloat and ttext) originated from discrete data types to manage temporally dynamic behavior of moving objects and two spatiotemporal types (tgeompoint and tgeogpoint) for managing geometric and geographic object properties respectively. These data types further refer parameters which contribute to successful implementation of moving object databases.

## **1.6 Mobility Data Properties (MDP)**

Data quality and data standards collectively play a very important role in defining how good a GIS application / utility can be for users. Data quality is related to “fitness for use” which varies with intended use, scale, and method of collection. Metadata with appropriate / acceptable data standards and training levels in data production are of prime value. Like all other databases, MobilityDB also relies on the quality of mobility data both on dataset and parameters level. Metadata for Mobility data must include accuracy, reliability, relevance, timeliness, completeness, conciseness and time dimension in space in line with Federal Geographic Data Committee (FGDC) and United States National Map Accuracy Standard (NMAAS). In addition to aforesaid data quality standards, projection, scale, cartographic quality, classification scheme, periodic instances along with transfer format are of paramount importance. MDP must include timestamps, universally unique identifier (UUID), Vehicle type / state, costs, currencies, Propulsion types and temporal trip information. A trip is the combination of all GPS points collected since start till end of a journey inclusive of time instances. MDP data provisioning must be based upon reliable application program interfaces (APIs) rather than conventional electronic data interchange. MDP provider APIs must be flexible enough to support pagination, municipality boundaries and event timings. Table 1.1 shows data segments as key parameters with respect to moving objects trips:

Table 1.1. Key parameters of MoD.

<b>Field</b>	<b>Type</b>	<b>Req / Opt</b>	<b>Remarks</b>
Provider ID	UUID	Required	A unique UUID must be according to provider list
Provider Name	String	Required	Public name to each MDP
Device ID	UUID	Required	GPS Device broadcasting UUID compatible
Object ID	String	Required	Unique License number allotted to each moving Object
Object Type	Enum	Required	Type of Moving Object
Propulsion Type	Enum []	Required	Array of Propulsion types for multiple values
Trip ID	Alpha-Numeric	Required	Unique values assigned to each Trip
Trip Duration	Period	Required	Trip start and end time
Trip Distance	Int	Optional	Distance travelled by moving object (can be calculated with GIS function based upon source and destination)
Route	Tgeompoint	Required	Continuous mapped temporal geometric point locations recorded during the trip
Accuracy	Int	Required	Approximation accuracy level in distance units (meters) of points during route
Start time	Timestamp	Required	Self-explanatory
End time	Timestamp	Required	
Publication time	Timestamp	Required	Data and Time of trip availability
Trip Cost	Int	Optional	Standard trip charges
Currency	String	Optional	Alphabetic standard currency code

With the application of MDP, mobility data exchange always remains the key factor for implementation of successful mobilityDB in any country. Unfortunately, no sophisticated Mobility data exchange portals exists in Pakistan on national level. However, national spatial data infrastructure (NSDI) is under study since last decade under the umbrella of Survey of Pakistan (SoP) as a joint venture with Space & Upper Atmosphere Research Commission (SUPARCO) Pakistan, Pakistan agricultural research council (PARC) and Pakistan bureau of statistics (PBS). The trajectory of NSDI in Pakistan started since 2009 with efforts put in by SoP and SUPARCO, however, a company named GIS plus recently assigned the viability study of practical implementation of NSDI in Pakistan. Issues related to the implementation of NSDI remains same from 2009 till date. These issues related to data, policies at institutions, technical limitations, and skilled human resource (Ali, 2020).

### **1.7 Mobility Data Space (MDS)**

The Mobility Data Space is a data marketplace where partners in the mobility sector can share data based on equal rights and data sovereignty to facilitate and advance innovative, environmentally sustainable, and user-friendly mobility concepts. It sets itself apart from existing data marketplaces through its focus on transparency, decentralized architecture, high data protection standards and data sovereignty. It also offers users unique opportunities to benefit from the added-value potential of their data. Establishing the Mobility Data Space in the market aims to contribute to the German and European economic and digital sovereignty.

Its technical basis is the result of close collaboration of European and national initiatives to ensure compatibility with GAIA-X domain projects and

other European data spaces. The German government has provided start-up financing through the Federal Ministry of Transport and Digital Infrastructure. The Mobility Data Space is open to all companies, organizations and legal entities working on mobility solutions. From major vehicle manufacturers to rideshare services, from public transport operators to navigation software companies as well as from research institutes to bikeshare providers.

## **1.8 Advantages of the MDS**

### **1.8.1 Transparency and data sovereignty**

Users have access to the Mobility Data Space under the same terms and conditions. Data exchange between buyer and seller is free, however both can make transaction on mutual agreement. Data is transmitted directly between the contractual parties (peer-to-peer). There is no central data storage, and the data provider always knows who is using their data. At present, there is no comparable mobility platform available to organizations who do not want to go through an intermediary when exchanging data. Today, they must enter time-consuming and cost-intensive individual contracts. Data providers can decide who can use their data. They are also free to deny certain entities the usage of their data. Pricing is completely self-determined by the providers.

### **1.8.2 Wide range of data from a single source**

The Mobility Data Space covers all kinds of mobility-related data, including i.e., maps, weather, and infrastructure data. This sets it apart from other data spaces in the mobility sector which often only cover certain segments. The data offered in the National Access Point for mobility data, operated as a mobility data marketplace by the German Federal Highway Research Institute on behalf of

the German Federal Ministry of Transport and Digital Infrastructure, will also be made available in the Mobility Data Space. This includes data that public companies and authorities must share under the European guidelines on smart transport systems and open data, such as information on hazards, roadwork and available truck parking space along motorways and highways. Interoperability with other data spaces, such as other European countries and GAIA-X domain data spaces, will give users cross-sector opportunities to access data and thus offering a high degree of scalability.

### **1.8.3 Easy to use**

The data space offers its users easy access and follows transparent rules. For the most basic types of data sharing, a prefabricated and easy-to-configure connector is available. Training and tutorials are available to assist users in completing various tasks within the data space.

### **1.8.4 Data protection and security**

The Mobility Data Space complies with antitrust legislation and the European General Data Protection Regulation (GDPR) as well as the European directives and conceptual thinking on the Data Governance Act (DGA), Digital Service Act (DSA) and Digital Markets Act (DMA). To ensure complete compliance with data protection requirements, the German Federal Commissioner for Data Protection and Freedom of Information (GFDI) has been consulted. In accordance with the principle of “Security by Design”, the Mobility Data Space has been designed in a decentralized manner, and data is held by its users. Only the data catalogue itself is stored centrally.

## **1.9 Areas of application in Pakistan**

Pakistan is far behind in utilization of Mobility data in the field of GIS.

Analysis of speed and transport infrastructure can not only help generate useful velocity maps but also make administration upgrade existing road network in Pakistan. Based upon route and trajectory analysis, travel time can be accurately calculated. Trip time forecasting can contribute to the development of more innovative software applications which can help plan / book before time. Most of big cities like Karachi, Lahore, Faisalabad, Rawalpindi, and Multan etc. have already grown tremendously in terms of population, however, their road infrastructures are not upgraded with the passage of time. MobilityDB can provide current and upcoming road congestions warning. Utilization of datasets including vehicle speed, school and offices timings, roads conjoining, traffic lights, weather condition, road infrastructure condition and number of lanes along with Mobility data can help resolve traffic congestion issues currently faced by major cities in Pakistan (Syed, 2014). Traffic measurements can be done based upon different time intervals like hourly, daily, or weekly etc. MobilityDB can easily extract traffic volume / count information on a certain time and space using tgeogpoint (inst) function. Trips data can help extract multi-dimensional information including mobility optimization, intermodal, and commuting scenarios (Rovinelli, 2021).

MobilityDB can help understand pedestrian mobility patterns for population / density evaluation. MobilityDB can help overcome the crash ratio with the help of heatmaps based upon vehicle, bike and pedestrians crash incidents in densely populated cities like Karachi, Lahore, Faisalabad, Peshawar and Multan. This would not only save human life but also reduce burden on



emergency departments like Rescue 1122 and Hospitals. MobilityDB can predict crash risks on major roads. Pakistan is the 4<sup>th</sup> largest polluted country as per annual assessment by World Health Organization (WHO). MobilityDB has the capability to track temporal emissions by transport and help overcome pollution related hazards. MobilityDB can help record sea level rise and its impact on road infrastructure.

MobilityDB can forecast equality of resources based on population, demographic, and socioeconomic parameters. Equity evaluation can also be made with the help of velocity maps produced with mobilityDB. MobilityDB Can analyze and classify different roadways based upon their mobility loads, facilities, and services. Public mobility services in Pakistan including Careem, Uber, Bykea etc. can further be identified for their different socio-economic impacts on society. Currently there is no sophisticated mechanism in Pakistan for upgradation of road infrastructure. Mobility data can be utilized for identification places with poor infrastructure for further restoration. Moreover, point of interest and placed like school, hospitals and groceries can well be managed with the availability of mobility Data. Mobility pattern is very fundamental identifier in crime prediction. Socio temporal Trajectory Analysis in MobilityDB can be applied in law & order management offices in Pakistan.

Moving objects databases generally has large sizes due to continuous interval-based recording of geometric points during the journey of objects. One of the benefits of using MobilityDB is the storage of large temporal dataset onto a smaller storage by making temporal geometric point sequence (Zimányi, 2020). MobilityDB offers a construct for representing the evolution of a value during a sequence of time instants where value between successive time instant is interpolated by a linear function (Rovinelli, 2021). Moving objects data if recorded in a temporal way produces many records These records are uniquely identified on the bases of timestamp and geometrical locations. It is therefore a computational resource hungry and complex task to store and manage mobility datasets in traditional database management systems like SQL or MongoDB etc. MobilityDB helps us to resolve these issues of data scaling and management by creating an aggregate geospatial trajectory record thus reducing overall storage space. This ability of data compactness by detaching redundance provide an extra edge to MobilityDB over PostGIS. Analysis performed on Moscow public transportation showed extraordinary storage reduction for 10 billion rows (500 MB / day) to 15 thousand (05 MB / day) rows (Godfrid, 2022).

MobilityDB provide many functions (timespan, direction, startTimestamp, endTimestamp and nearestApproachDistance etc) views (geography\_columns, geometry\_columns, raster\_columns and custom materialized views etc) and abstract data types (tgeogpoint for temporal geography point, tgeompoint for temporal geometry point, Time Types and Period Types etc.) as mentioned in Table 1.1. These functions provide a rich analytical capability to MobilityDB by using temporal data types and help extract precious information. MobilityDB allows us to visualize semantic trajectories

modeled upon temporal patterns. This means that we can store spatiotemporal points along with additional information in a single record / table (Rovinelli, 2021). Computing travel time, distance among different locations, road / path on which certain object is travelling, pattern and semantic analysis are some of MobilityDB analytical functions.

Mobility datasets are generally large in scale and require an efficient distributed database management technique. MobilityDB and Citus (Postgres extension for distributed data management) together make it possible to manage gigantic trajectory datasets. Tsismelis, Dimitrios deployed big data on BerlinMOD and Scalar datasets using a MobilityDB cluster and Citus on MS Azure (Tsismelis, 2021). During the project, several other factors regarding performance were also identified with the help of an open-source software named 'autoscaler' that used to scale and automatically monitor performance matrices and help take performance related decisions. Installation and configuration of Citus on top of MobilityDB can be completed by using kubectl command from your host machine or simply pulling docker image 'mobilitydb-aws:latest' configured with Citus extension, however, step by step instructions are available on GitHub portal. SQL (Structured Query Language) is an ANSI (American National Standards Institute) standard query language used to manage relational databases while PostgreSQL is an open-source and world's most advance and quickly evolving relational DBMS (Relational Database Management System) developed by PostgreSQL Global Development Group (PGDG) in compliance with SQL. Large industry has been developed around these two words that is PostgreSQL and ecosystem (Open-source applications, certified application vendors). Postgres ecosystems include all those tools and

applications which are interoperable with PostgreSQL (Figure 1.1). MobilityDB utilizes full scale SQL interface and support for majority of PostgreSQL ecosystems while performing query-based operations. A database adapter is the implementation mechanism of database connector. In case of MobilityDB, python version of MobilityDB connector is available which support both the psycopg2 (most powerful python adapter for python language) and asyncpg (3x efficient than psycopg2 and clean database client library for custom data types) adapters for PostgreSQL developed by MobilityDB team. Similarly, MobilityDB-SQLAlchemy is also an extension package for SQLAlchemy (famous object relational mapper for mapping classes as SQL objects like tables or views etc giving databases and SQL a new orientation).

### **1.10 MobilityDB Visualization**

Visual illustration of temporal datasets always remained a cumbersome task for researched. Generally relational databases are not equipped with tools or functions to display time series datasets. DB researchers are working since last decade on the development of interactive tools to display MOD. Tools which are capable of handling sophisticated and complex temporal queries and similarly display the temporal results. MobilityDB team has provided an open-source tool named 'MOVE' as QGIS plugin with simple interface to perform query execution, data retrieval and visualization of dynamic attributed of trajectories. MOVE tool creates separate layers for each PostGIS geometry type (MultiPoint, MultiLineString and MultiPolygon) and MobilityDB temporal points (tgeogpoint, tgeompoint). Each newly created layer with MobilityDB temporal type is marked as temporal layer and can be explored or visualized with the temporal controller of QGIS. It is pertinent to mention that an optimized

visualization depends upon the number of records returned, step interval and frame rate settings. Any query which results millions of temporal records can make QGIS unstable if executed in MOVE plugin configured on low specification computer. In such scenarios, limiting the SQL query to small number of rows might be helpful. Another solution to the visualization of temporal data is provided by Python based open-source libraries named ‘MovingPandas’ (Graser, 2020) and ‘moveVis’.

### **1.11 Challenges faced by MobilityDB**

Two levels of issues currently faced by MobilityDB. The first issue is related to configuration of working platform which is the source of motivation behind this study. The second level issues are related to the implementation of MobilityDB due to missing of trajectory segmentation and ML based APIs (Zimányi, 2021). Moving object datasets (MoDs) mostly require data warehouse (DW) to support data analysis and online analytical processing (OLAP) queries (Vaisman, 2019). On data processing side, MobilityDB still facing issues while performing spatio-temporal join queries. Very limited reading / help material is available which is the main motivation behind this study. Similarly, visualization of mobility data is done in QGIS and python libraries ‘moveVis’ and ‘MovingPandas’. Visualization of Mobility data in QGIS is a bit simple and works fine on select statements. However, for complex queries having joins with one or more tables, Move plugin produces errors. On other hand, visualization of mobility data using python package ‘moveVIS’ is very powerful and works well on all type of queries but requires python development skills. With the application of MDP, mobility data exchange always remains key factor for implementation of successful mobilityDB in any country.

Mobility Data is generated in many public and private sectors which include Pakistan Railway, Metro Bus Service, Rescue 1122, Civil Aviation, Uber, Careem, Bykea, TPL Insurance and many more. Unlike many public and private organizations in European Union who regularly release their data in publish form for research purposes, no such trend exists in Pakistan. Although many private data exchange portals exists in Pakistan but most of the portals are managed by private organizations with limited geospatial data availability. Open Data Pakistan (ODP) is collaborative effort by National Center for Big Data and Cloud Computing (NCBC), Lahore University of Management Sciences (LUMS) and Higher Education Commission (HEC) with lot of non-geospatial datasets. Most of above-mentioned offices were approached through verbal and non-verbal means of communications for sharing their old mobility data for this study but none of them cooperated in this regard.

TPL is the leading insurance company in major cities including Karachi, Islamabad, Lahore, Multan and Faisalabad in Pakistan which provides insurance services in the field of transport, property, tourism, mobile, health, marine, engineering and miscellaneous. However, we are using mobility dataset of vehicles insured with TPL. Keeping in view, the enormous size of data, TPL provided dataset in the form of SQL backup file of size 2.92 GB.

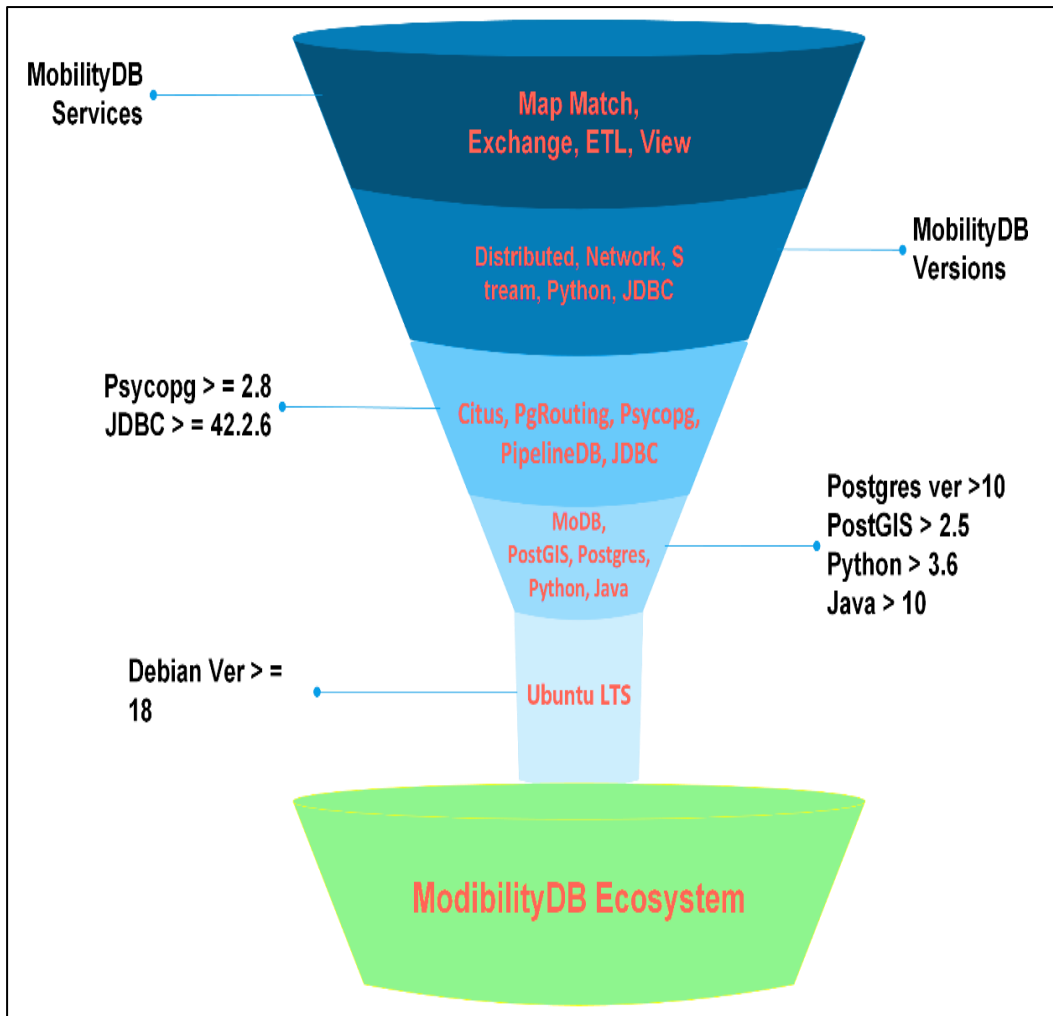


Figure 1.1. MobilityDB ecosystem.

## **2 MATERIALS AND METHODS**

### **2.1 Study Area**

Pakistan's urban transport is still far behind in the field of automated online facilities achieved in developed countries. Several factors including good governance, capacity building and urban planning for efficient, cost-effective public transportation have already been highlighted (Imran, 2009). In addition to the improvement of transport infrastructure, modernization of urban transport is the need of time. Many areas including real time traffic monitoring, security, optimized spatial Out-of-home (OOH) advertising, historical trends analysis and moving objects database management are some of many areas which require radical research to be done. This study focuses on the management of large datasets of TPL with MobilityDB and analyses different applications of MobilityDB. TPL's GPS equipped vehicles mostly work in major cities of Pakistan, however, their trips may extend to sub urban areas of Pakistan as shown in Figure 2.2. However, this study will focus on Islamabad city due to its organized road infrastructure and organized traffic conditions.

### **2.2 Methodology**

This study has been segregated into two major phases. First phase includes comparison of different versions of MobilityDB, installation and configuration of the most suitable version to be used in this study. Second phase includes loading of TPL's dataset into MobilityDB, its upgradation, socio-temporal analysis and extraction of useful information with the help of MobilityDB queries which makes it possible in an optimized and simpler way. Configuration of MobilityDB is a cumbersome task due to limited helping material viz a viz non availability on



windows platform. In this regard, Docker (Linux environment simulation application for windows) application had been used to configure MobilityDB over windows and dedicated Ubuntu operating system, along with MobilityDB instance, is to be installed in parallel to further execute complex queries.

Data loading into MobilityDB, upgradation and evaluation is the initiating and complex steps in the study followed by data acquisition. Data received in SQL server backup file which was further migrated to PostgreSQL server with the help of SSIS (SQL Server Integration Services) model, followed by restoration on Microsoft SQL Server 2019.

Upgradation of data with respect to MobilityDB i.e. generation of points, temporal points and trajectories prior to visualization of temporal trajectories in QGIS 3.22.4 with the help of open-source plugin named 'Move'. Generation of spatiotemporal queries for extraction of analytical information like max concurrent trips, total travelled distance, average duration / speed of trips, longest trip, histogram of trip length, minimum distance between pair of vehicles, activation of certain vehicles at certain point in time etc. Figure 2.1 shows the proposed methodology flowchart to be followed in this study.

### **2.3 Environment Building**

Currently, MobilityDB is available on Linux / Debian operating systems only, however, MobilityDB team is dedicatedly working for implementing MobilityDB extension on PostGIS in windows. Two main branches of MobilityDB i.e Master and Development are available for end users to use in their research or production environment. Difference between the two is only the number of functions each database instance carries. Master copy is the latest release with every function thoroughly test by MobilityDB team before publishing on GIT or Docker hubs

(Zimányi, 2020). However, development branch is the upcoming version still under validation by MobilityDB development team with more function than master branch. To successfully configure MobilityDB extension to Postgres installed on Linux following requirements are mandatory: -

- PostgreSQL Version Higher than 10 --{Checked by simple SQL command # SELECT version();}--
- CMake Version higher than 3.6 --{Checked by batch command # cmake --version}--
- PostGIS higher then 2.4 10 --{Checked by simple SQL command # SELECT PostGIS\_Version();}--
- JSON-C to implement RCO (Reference Counting Object) model for easy construction of JSON objects in C and their easy casting from C to JSON representation of objects.
- GNU Scientific Library (GSL) is free numerical library of C & C++ for mathematical routines.
- Development files for PostgreSQL, PostGIS / liblwgeom, JSON-C, PROJ

## 2.4 Linux Installation

To install MobilityDB in Linux following batch commands are required to be executed in chronological order: -

(a) Following command would incorporate all deficiencies requirements for Linux / Debian systems.

- *sudo apt install build-essential cmake postgresql-server-dev-13 libproj-dev libjson-c-dev*

(b) Run gist (commands used to share code or snippet with others) in chronological order: -

- *git clone https://github.com/MobilityDB/MobilityDB*
- *mkdir MobilityDB/build*
- *cd MobilityDB/build*
- *sudo apt-get update*
- *sudo apt install cmake # version 3.16.3-1ubuntu1*

Note: if the local machine is already running any application (PgAdmin, Postgres or Docker Container on port: 5432 (default Postgres port), You need to stop all such services / applications before executing following command)

(c) To install PostGIS following apt repository command would be required: -

- *sudo apt update*
- *sudo apt -y upgrade*
- *sudo reboot*
- *sudo apt -y install gnupg2*
- *wget--quiet-O-*

*https://www.postgresql.org/media/keys/ACCC4CF8.asc | sudo apt-key add -*

- *echo "deb http://apt.postgresql.org/pub/repos/apt/`lsb\_release -cs`-pgdg main" |sudo tee /etc/apt/sources.list.d/pgdg.list*

(d) To install PostGIS version 13 or later, run following commands: -

- *sudo apt update*
- *sudo apt install PostGIS postgresql-13-PostGIS-3*

(e) Enable newly installed PostGIS : -

- *sudo -i -u postgres*
- (f) Create user & database: -
- *createuser PostGIS\_user*
  - *createdb mobilitydb -O PostGIS\_user*
- (g) Postgres default user 'postgres' comes with default password 'postgres' which may be also used as well. To change default user password, following command would be required: -
- *sudo su - postgres*
  - *psql -c "alter user postgres with password 'some\_password'"*
- (h) Connect to the database:
- *psql -d mobilitydb*
- (i) Verify that newly installed PostGIS is working properly, check version installed: -
- *select PostGIS\_version();*
- (j) With all above mentioned configuration, we need to update 'postgresql.conf' file with the compatible version of PostGIS (In following command, we will use PostGIS version 3.2.1). It is pertinent to mention that 'postgresql.conf' file may be located on Linux / Debian based operating system by two methods (Generic Search and Path Based). In Generic search, user can use to locate the configuration file by using Linux open search on installation folder while path-based search requires complete path as mentioned below: -
- *\$ which postgres /usr/local/pgsql/bin/postgres*
  - *\$ ls /usr/local/pgsql/data/postgresql.conf*

'postgresql.conf' is generally used by Linux operating system, which cannot be modified with following changes due to limited update privileges by an ordinary login. The same can be modified by Linux 'nano' utility with administrative right or by using sudo commands: -

- *shared\_preload\_libraries='PostGIS-3'*
- *max\_locks\_per\_transaction = 128*

(k) If everything configured smoothly, now we need to enable the PostGIS and Mobilitydb extensions on the database in the same order as mentioned below: -

- If MobilityDB is connected to PgAdmin4, execute following commands in SQL query pane:
  - *create extension postgis;*
  - *create extension mobilitydb cascade;*
- By using Linux command terminal with psql commands: -
  - *psql mobility -c "create extension postgis"*
  - *psql mobility -c "create extension mobilitydb"*

## **2.5 Windows Installation**

Installation of MobilityDB on Linux from scratch is a very complex process to configure each required component with similar or compatible specifications. To ease out this cumbersome process, pre-compiled images have been made available on Docker Hub by MobilityDB team. Benefit of using these images include an easy installation and upgradation of MobilityDB on Linux / Debian and Windows based operating systems. Docker is an independent and open platform for developing, sharing, and running applications. While Docker containers help execute multiple application simultaneously in a loosely coupled

environment. It is portable artifact to package applications with all necessary deficiencies and configurations. Docker containers improve the development process in a way that generally when a team of developers working on some application is required to install most of the services on operating system directly on each development machine locally. This type of development architecture is usually complex and involve many errors related to configurational and versioning of services required with respect to host operating system. This approach of setting up a new environment can be a tedious job depending upon application complexity. In our case, setting up MobilityDB on a remote client from scratch. Each remote client requires all the requirement of MobilityDB mentioned earlier. This kind of cumbersome installation of MobilityDB is replaced by MobilityDB containers where no requirement is meant to be installed on local operating system because the container has its own separate OS layer mounted by Linux based image. Here we have everything packaged in one isolated environment i.e., every requirement of MobilityDB with specific version packaged with a configuration in the start script inside of one container. There are multiple Docker images of MobilityDB available on Docker hub (a public repository for Docker accessible without login requirement). Every image corresponds to different release and branch of MobilityDB with different PostGIS extensions / functions. Most stable with maximum extensions / functions with all dependencies is by codewit repository used in this study as well, however, for large and scalable datasets MobilityDB version named ‘MobilityDB-aws-Latest’ is preferred as it is equipped with Citus extension (specifically designed for distributed query management). Docker images are pulled and executed in Docker engine (a docker runtime) which can be installed on Mac, Linux, and

Windows. With the implementation of MobilityDB using docker give opportunity to execute multiple branch containers of MobilityDB simultaneously on different host port.

Following is the installation process of MobilityDB from Codewit repository of Docker hub: -

(a) Download docker requires only one command to fetch the container regardless of which operating system on host machine.

- *docker pull codewit/mobilitydb*

(b) Although MobilityDB container can work independently without any Docker-volume but to keep MobilityDB files outside of docker container, following command is used to create Docker Volume named 'mobility\_data'

- *sudo docker volume create mobility\_data*

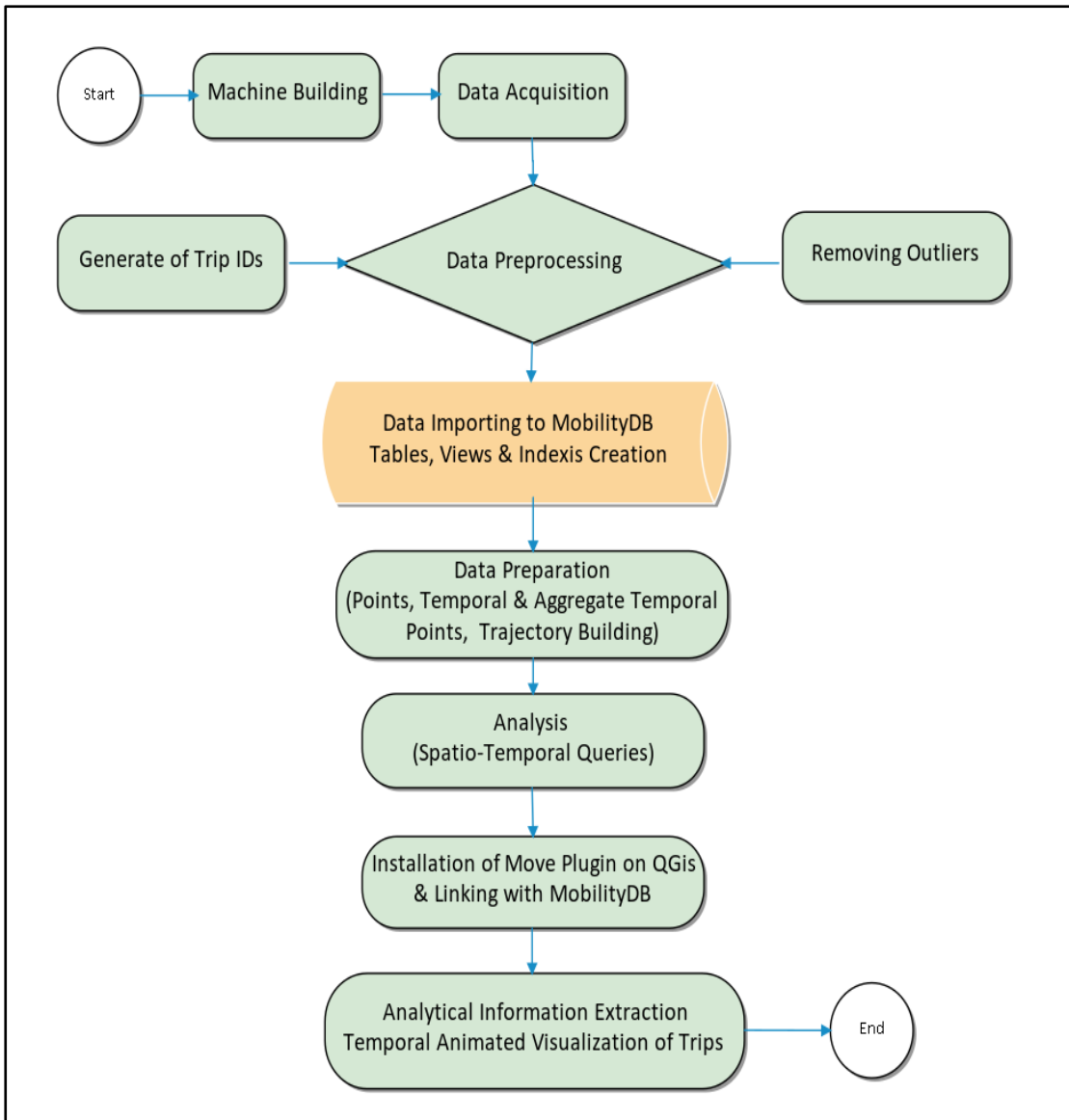


Figure 2.1. Methodology flow chart.



(c) All pulled images are required to be executed to make them operational. To make any conflict with any Postgres instance already running on local machine, we have mapped the postgres default port 5432 with the host port 25432 in the following command: -

- `docker run --name "mobilitydb_codewit" -d -p 25432:5432 -v mobilitydb_data:/var/lib/postgresql codewit/mobilitydb`

Upon successful execution of above three commands, MobilityDB docker image is now up and running and is ready to accept connections. We can connect newly configured MobilityDB container by using any version of PgAdmin4 Client (a web-based GUI tool used to communicate with Postgres database both locally and remotely) with following parameters: -

- **Name:** mobilitydb\_codewit or any name of your choice
- **Host:** localhost or IP address of MobilityDB container
- **Maintenance Database:** postgres
- **Username & Password:** docker (default ID & Pwd for all Docker images of MobilityDB, can be changed as well)

## 2.6 Comparison of different MobilityDB versions

MobilityDB is a new database management platform for moving object geospatial trajectories. There are various versions of MobilityDB available online mainly two branches. Having fully functional instance of PostgreSQL loaded with all desired extensions including PostGIS and MobilityDB is the key to smooth and successful development of applications based upon temporally moving

objects. To validate multiple docker images with all key parameters, following comparisons have been drawn: -

Comparison presented in Table 2.1 illustrates that different docker images are suitable for different scenarios. MobilityDB codewit-latest image is suitable for initial study or small datasets, however, for a scalable large dataset, MobilityDB-AWS-latest image equipped with Citus is recommended.

## **2.7 Data Loading to MobilityDB**

Applications running on or capturing mobility data always work on big and growing datasets. Such datasets require a sophisticated mechanism for ETL (extract, transform and load) like SSIS (SQL Server Integration Services) or Python based customized ETL processes which text databases cannot have. Traditional text databases like notepad, excel or access are not well suited to manage large and real-time growing datasets due to large data count and low performance limitations. Table 2.2 summarizes the overall metadata and Table 2.3 elaborate about the composition of TPL dataset in the form of SQL tables. Each table is of same structure listing month wise GPS points collected during trips except RegCom which lists the work done on each vehicle dataset details: - Monthly records and temporal points are illustrated in Figure 2.3.

Table 2.1. MobilityDB instance functional comparison.

S No	Image Name	Tag	Size-MB	Extensions	Functions /Views	Casts	Citus
1.	mobilitydb	Latest	641	06 (hstore, mobilitydb,	3021 / 04	84	No
2.		Master	674	pg_cron, plpgsql, postgis & postgis_topology)	3038 / 04		
3.	mobilitydb-aws	Latest	672	04 (citus, mobilitydb, plpgsql & postgis)	3044 / 05	82	Yes
4.	mobilitydb	Latest	440	03 (mobilitydb	2874 / 02	97	No
5.	13-2.5-develop	Develop	647	, plpgsql & postgis)	3031 / 04	84	No
6.	13-2.5-master	Master	617	Container does not have MobilityDB extension			

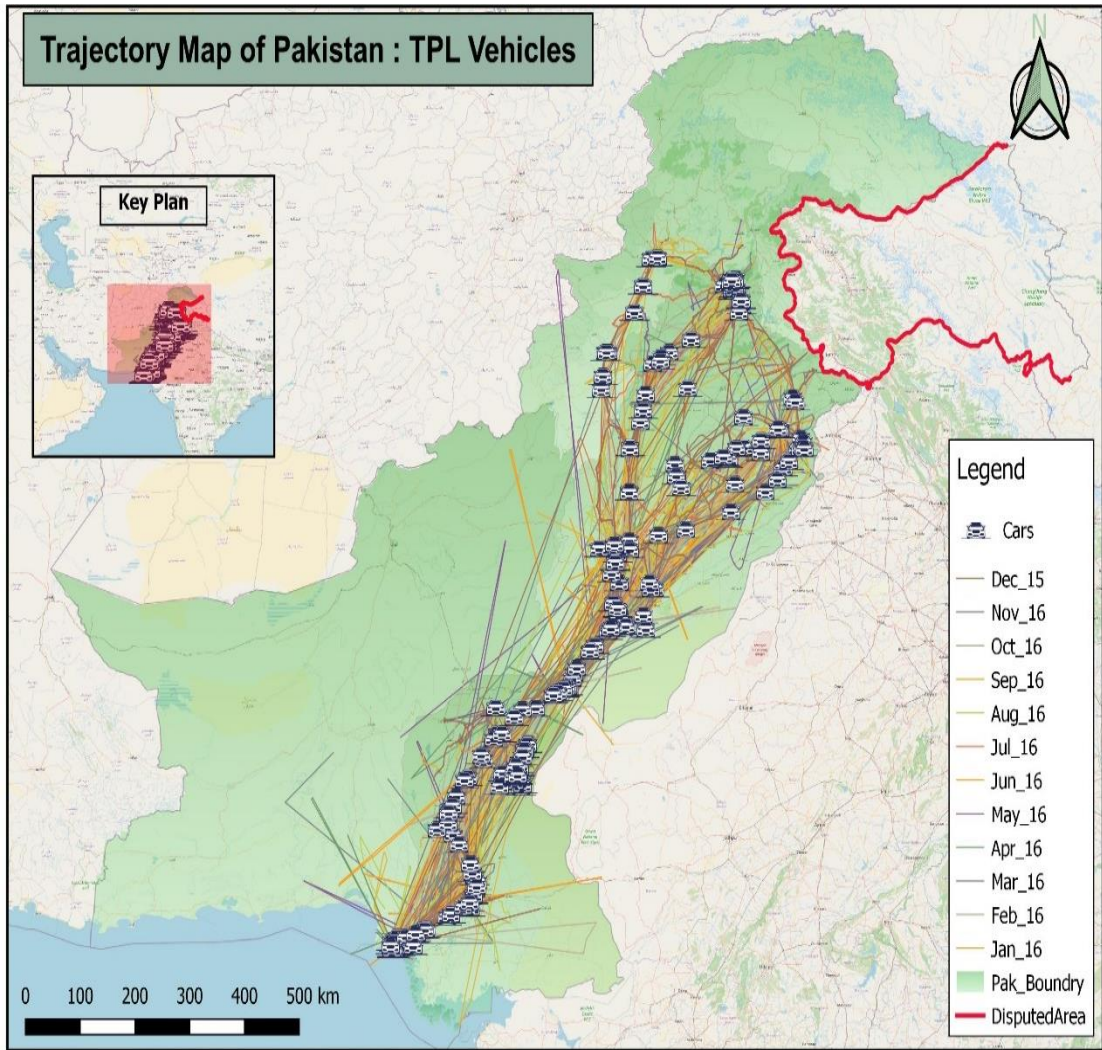


Figure 2.2. Study area map.

Table 2.2. Meta Data.

Sr No	Data	Remarks
(a)	Duration	01 October 2015 to 30 September 2016 (01 Year)
(b)	Vehicles Count	107
(c)	Record Count	105,301378 (105 million plus)
(d)	Size	DB Backup (2.92) and restored size is 32 GB

Table 2.3. Monthly record details.

Sr No	Name	Type	Records	Size	Remarks - Data Time
(a)	Analytics_Set_1	SQL Tables	7070801	2.4GB	Jan,2016
(b)	Analytics_Set_2		8256378	2.5GB	Feb,2016
(c)	Analytics_Set_3		9917801	2.9GB	Mar, 2016
(d)	Analytics_Set_4		10413247	3.1GB	Apr, 2016
(e)	Analytics_Set_5		11382101	3.5GB	May,2016
(f)	Analytics_Set_6		10345889	3.2GB	Jun,2016
(g)	Analytics_Set_7		10178527	3.1GB	Jul,2016
(h)	Analytics_Set_8		11147331	3.4GB	Aug,2016
(j)	Analytics_Set_9		9890153	3.0GB	Sep,2016
(k)	Analytics_Set_10		4916962	1.7GB	Oct,2016
(m)	Analytics_Set_11		5185764	1.8GB	Nov,2015
(n)	Analytics_Set_12		6594163	2.2GB	Dec,2015
(p)	RegCom		2261	0.1GB	Oct,2015 to Sep, 2016

Keeping in view the scope of dataset, SQL server's import / export tool was used to import to mobilityDB which uses SSIS data flow task at backend. Connectivity of MS SQL server with PostgreSQL require PostgreSQL ODBC driver (psqlODBC) to be installed. Once the ODBC driver is installed, it required connection configurations with MS SQL server (data source, database name, server name / IP address, username, password, and port number). Figure 2.4 shows the connection configurations of PostgreSQL server. Upon provisioning of required configurational details, a button named 'Test' is to be used to validate the connection strength. Once the connection is test as successful, the configurations are saved with any alias, 'pgadmin13' as shown in Figure 2.5.

After making successful PostgreSQL connection instance, data migration stage comes. MS SQL server import / export wizard is used which require data source information (MS SQL server in our case as mentioned in Figure 2.6). Data destination is selected with the help of data source name (DSN) as mentioned in Figure 2.7. Same DSN name is to be entered in the field named 'DSN' under connection string in Figure 2.7. Figure 2.8 shows the selection of tables as per Table 2.3 on next step.

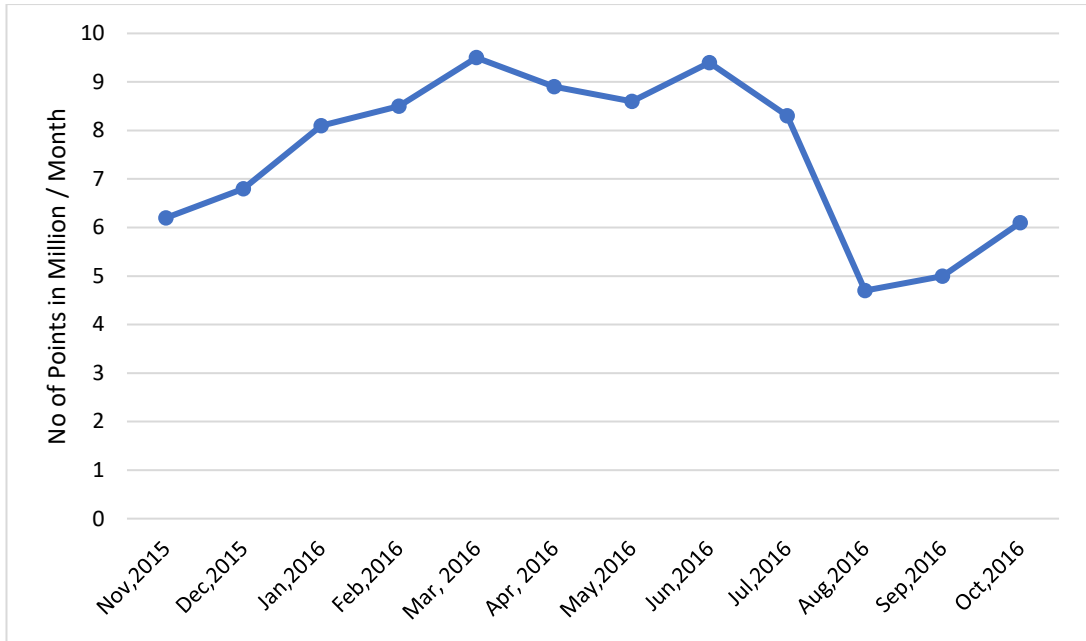


Figure 2.3. Month wise GPS points distribution.

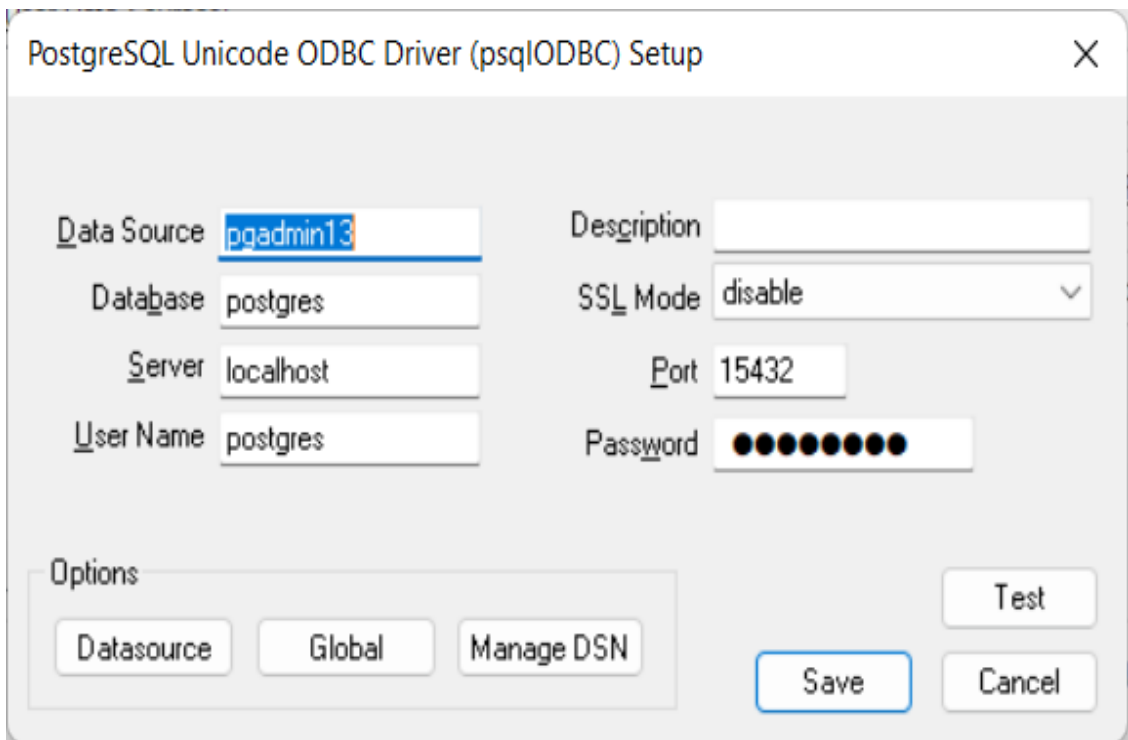


Figure 2.4. PostgreSQL ODBC driver configuration.

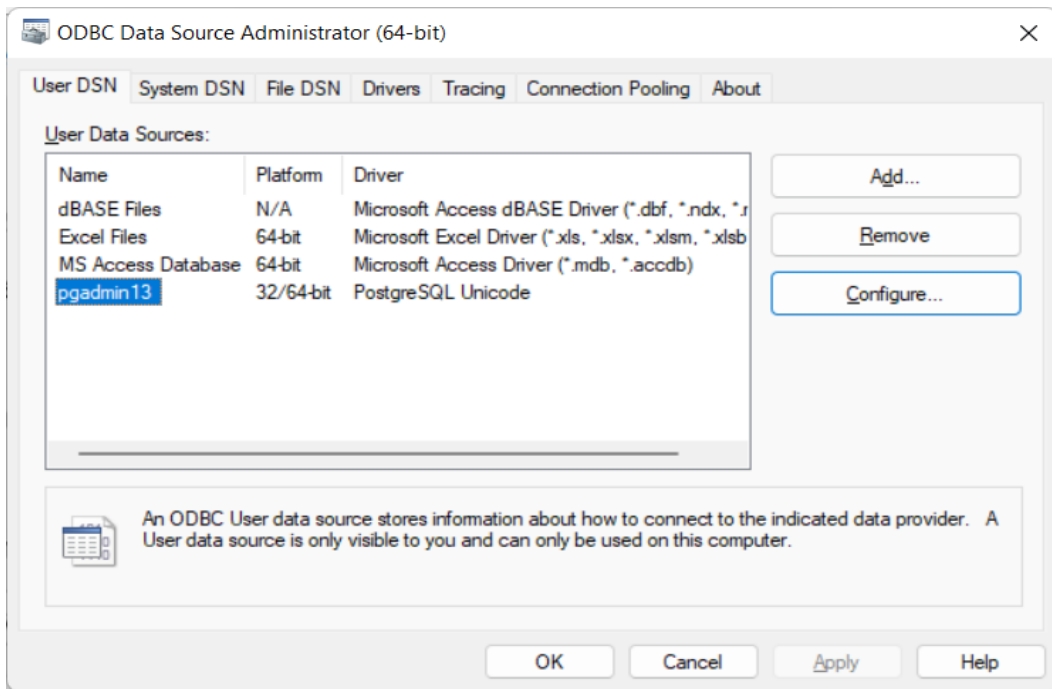


Figure 2.5. ODBC connection saved.

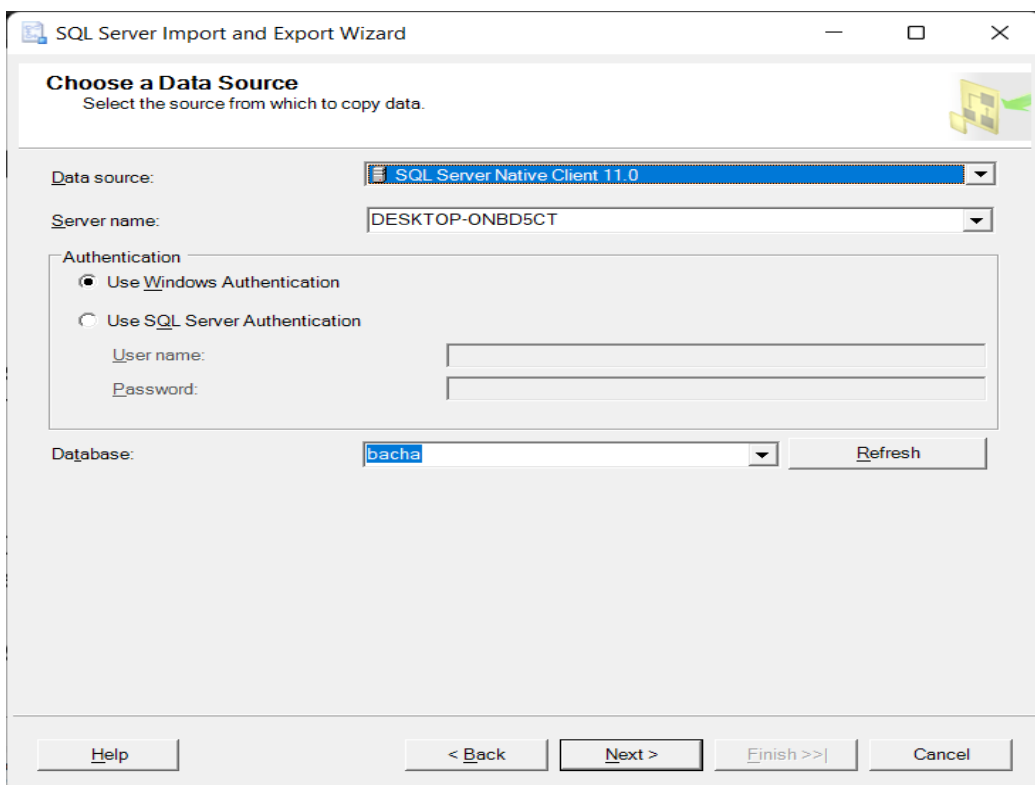


Figure 2.6. Import / Export wizard - data source selection.



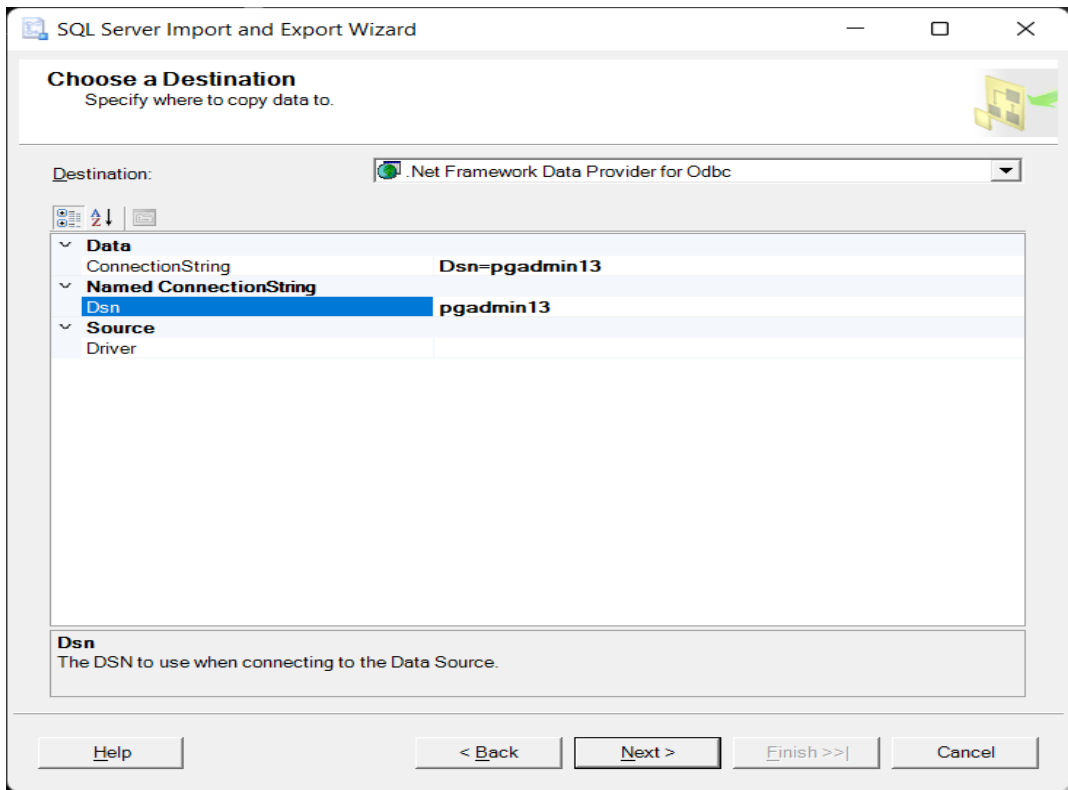


Figure 2.7. Data destination - DSN name specification.

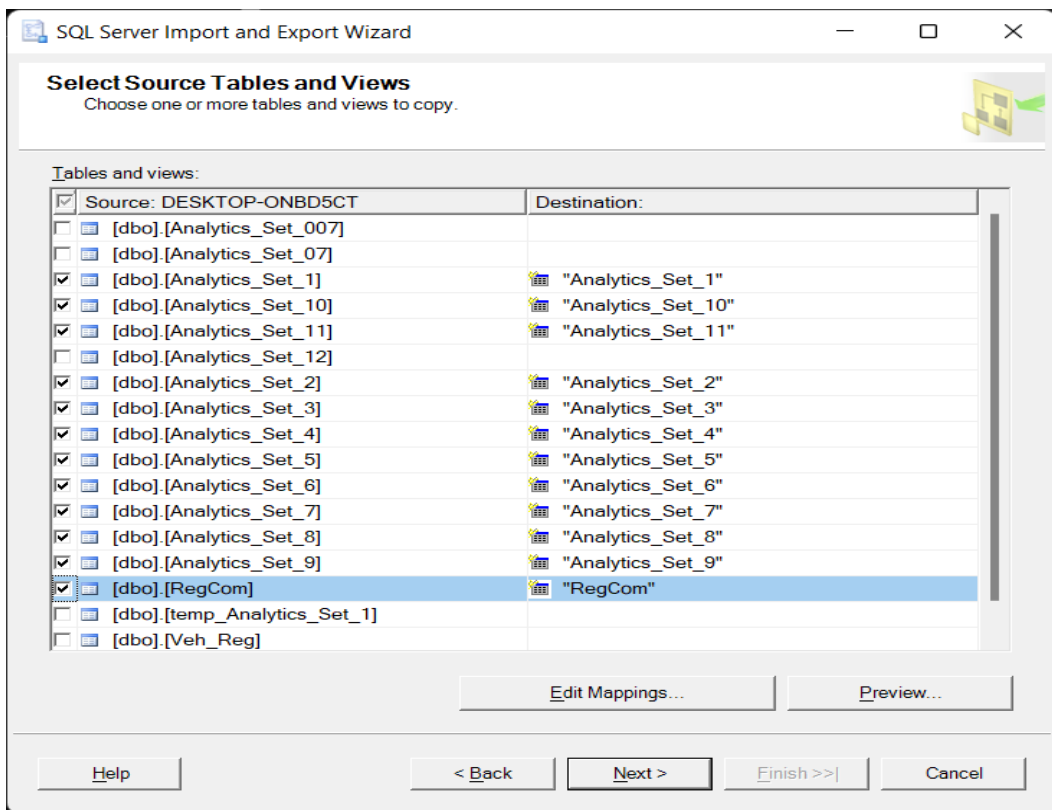


Figure 2.8. Table selection for migration to MobilityDB.

Data loading from SQL server to Postgres was accomplished in phased manner where table named “Analytics\_Set\_1” was imported first and “Analytics\_Set\_12” at last in chronological order. Data migration took several days due to large dataset and migration from local machine SQL server to MobilityDB hosted over Docker platform where each table took on average 12 hours maximum time as depicted in Figure 2.9.

## **2.8 Preprocessing**

After successful data migration to Postgres, data structure evaluation carried out. Table 2.4 clearly shows some data elements received from TPL office, was not required in this study. The same has been summarized in the remark’s column of Table 2.4 where some elements were marked as not required due non availability of information.

Table 2.4. Data elements received.

S No	Column Name	Type	Remarks
1	Id	int	Identity / unique column – Required
2	ReportGroupDate	datetime	Record Date and Time – Both columns are same, only one can be kept for future use
3	Ordered	datetime	
4	assembletime	Datetime	
5	Cellnumber	varchar(100)	TPL specific unique ID for each vehicle - Required
6	vehicleReg	varchar(100)	Vehicle registration number - Required
7	DriverId	varchar(500)	Driver names - Required
8	Vehiclestatus	varchar(500)	Engine ON / OFF – Required
9	MobileSpeed	int	Speed of Vehicle – Required
10	Mobileodo	int	ODO meter speed – Required
11	Location	varchar(100)	Blank Column Not – Required
12	Skillset	varchar(500)	
13	c2	int	
14	C3	int	
15	C4	int	
16	C5	int	
17	C6	int	
18	C7	int	
19	C8	int	
20	C9	int	
21	distance	int	
22	direction	varchar(100)	
23	gpstime	Datetime	GPS time – Required
24	locationName	varchar(5000) )	Customized location names by company– Required

<b>continued</b>			
<b>S No</b>	<b>Column Name</b>	<b>Type</b>	<b>Remarks</b>
25	locationdistance	Int	Location distance – Required
26	locationdirection	varchar(20)	Direction to / from location – Required
27	latitude	float	Latitude value – Required
28	longitude	float	Longitude Value – Required
29	zipcode	varchar(100)	Blank – will be updated by Reverse GeoCoding
30	country	int	Blank Column Not – Required
31	areagroup	int	Area group – Required
32	locationtolerance	int	Location tolerance – Required
33	province	int	Blank – will be updated by Reverse GeoCoding

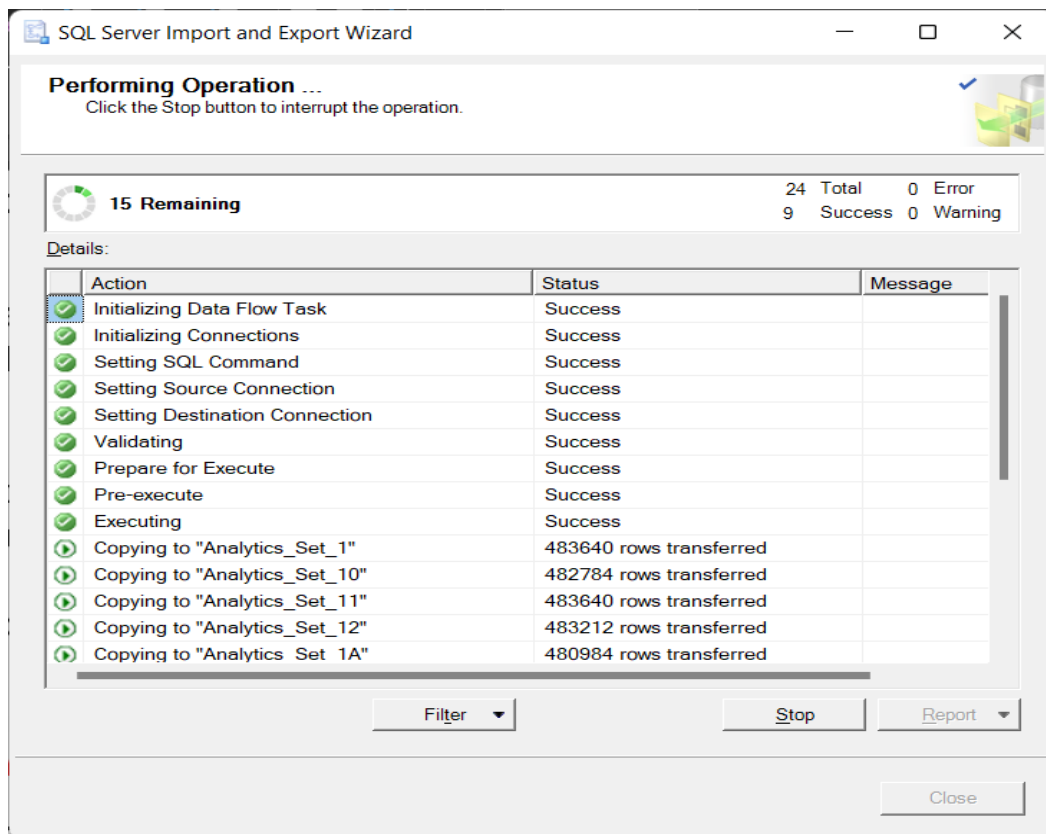


Figure 2.9. Data migration from SQL to PostgreSQL.

All such columns which were marked as not required were deleted and columns named city, state (province), country, zip code, complete address, trip, trip\_id and trajectory were added for further analysis. Table 2.4 depicts the final list of columns to be used in this study.

## **2.9 Generation of Trip ID**

It is pertinent to mention that trajectory analysis is incomplete and complex without any identification of start and stop of every trip. Trip ID field plays pivotal role in making individual vehicle's trajectory analysis. However, trip start and stop information was not available in provided dataset as mentioned in Table 2.4. Keeping in view the large dataset, it was necessary to generate accurate trip ID with available data columns. Two indigenously developed utilities were tested on subset of dataset.

Algorithm 1 shows a tool that was developed in python to read each vehicle's GPS record and assigns Trip ID to every group of records which appear in chronological order as per GPS time stamp.

Algorithm 2 shows the SQL snippet for cursor to automatically generate and assign a unique trip ID to every bunch of records which start with vehicle status ID as 'Start Up' and closes with vehicle status ID as 'Ignition off'. The cursor script is required to be executed on every table i.e Analytics\_Set\_1, Analytics\_Set\_2, Analytics\_Set\_3, Analytics\_Set\_4, Analytics\_Set\_5, Analytics\_Set\_6, Analytics\_Set\_7, Analytics\_Set\_8, Analytics\_Set\_9, Analytics\_Set\_10, Analytics\_Set\_11, Analytics\_Set\_12 respectively by replacing only table name.

Comparison of both utilities for assignment of trip ID proved the SQL cursor method with respect to performance and accuracy. On the other hand, Python

based tool gave 90% accuracy and required lot of CPU and database resources. This tool not only took lot of time due to multiple database hits but also missed trip ID on some legitimate records. Finally, SQL based cursor was applied on whole dataset due to better accuracy and optimization to generate trip IDs. After generation of Trip IDs, data columns were finalized (Table 2.6).

## **2.10 Database Schemas preparation**

Algorithm 3 shows PostgreSQL schema was used for alteration of each SQL data table as mentioned in Table 2.6.

## **2.11 Creation of Point and Temporal Point**

A Point is a zero-dimensional spatial representation of discrete information while temporal points are time associated points. To create a point using latitude and longitudes, PostgreSQL function named ST\_MakePoint is used followed by assigning spatial reference system ID 4326 (a common ID used to relocate objects bearing latitude and longitude on earth surface). Following query is used on all tables mentioned in Table 2.3 respectively.

```
Update public."Analytics_Set_1" Set point =  
ST_SetSRID(ST_MakePoint(longitude ,latitude), 4326), tpoint =  
tgeompointinst(ST_SetSRID(ST_MakePoint(longitude ,latitude), 4326),  
gpstime);
```

Table 2.5. Updated data elements.

S No	Column Name	Type	Remarks
1	Id	int	Identity / unique column – Required
2	ReportGroupDate	datetime	Record Date and Time – Both columns are same, only one can be kept for future use
3	Ordered	datetime	
4	assembletime	Datetime	
5	Cellnumber	varchar(100)	TPL specific unique ID for each vehicle - Required
6	DriverId	varchar(500)	Driver names - Required
7	vehicleReg	varchar(100)	Vehicle registration number - Required
8	Vehiclestatus	varchar(500)	Engine ON / OFF – Required
9	MobileSpeed	int	Speed of Vehicle – Required
10	Mobileodo	int	ODO meter speed – Required
11	gpstime	Datetime	GPS time – Required
12	locationName	varchar(5000)	Customized location names by company– Required
13	locationdistance	Int	Location distance – Required
14	locationdirection	varchar(20)	Direction to / from location – Required
15	latitude	float	Latitude value – Required
16	longitude	float	Longitude Value – Required
17	areagroup	int	Area group – Required
18	locationtolerance	int	Location tolerance – Required
19	province	int	Blank Column Not – Required
20	City	varchar(100)	New Columns – generated by Reverse Geo Coding
21	State	varchar(500)	
22	Country	varchar(20)	
23	Complete_address	varchar(5000)	
24	Trip	tgeogpoint	Columns created in newly created table ‘Analytics_trips’
25	Trajectory	geometry	
26	Point	geometry	Newly created column Geographic Point
27	Tpoint	Tgeompointinst	Newly created column Temporal Geographic Point
28	Trip_id	int	Newly created column Unique Trip ID

## 2.12 Data Cleaning

While generating trajectories, number of garbage / outlier records were observed. Such records violate the functional dependencies of MobilityDB. For instance, most of the records were duplicate due repeating time stamps. In other words, these records were generated duplicate due to malfunction of GPS instrument installed on moving object. MobilityDB's temporal sequence function 'tgeompointseq' expects every point with unique and incremented time stamp. In case of duplicate records with same time stamps, this function return error message 'Timestamps for temporal values must be increasing + duplicate time stamp'. Similarly, every data segment also bears some outliers where latitude was -80 and longitude was zero (outside the boundaries of Pakistan) due to malfunction of installed GPS transponder. GPS time recorded during the trip was used to get trip start and stop time. These timestamps were then converted into MobilityDB period data type for further calculation of total journey time. Calculation revealed that some trips were longer than 48 hrs. Which upon investigation further revealed the merging of more than one trip was due to missing trip start status. Table 2.8 illustrate number of duplicated records, outliers or trips with abnormal lengths or Lat/Lng which were eliminated before generating trajectories.

On this stage, all duplicate records were identified and only one unique record was kept for further processing and trajectory generation. Outliers were simply deleted against mentioned latitude / longitude. Elimination of such duplicate time stamp record was very tedious process which is completed in two steps. First step was to copy all duplicate records in a separate table against each table mentioned in Table 2.3 with alias as '\_Dup' at the end. In second step, a



python based small utility was developed which reads every record from newly created duplicate records table and perform removal of respective duplicate record bearing max ID from source table. Table 2.7 shows the query and python-based duplication removal utility code which were used in data cleaning.

### **2.13 Creation of Trajectory**

Trajectory is the path followed by an object with the passage of time. Trajectory creation is the core step performed in the study which requires temporal points to be arranged in the form of array. A PostgreSQL function named 'array\_agg' is used to arrange temporal points on the elements of array returned. These array elements are sorted on ascending order of GPS time and passed to a temporal sequence function of MobilityDB named 'tgeompointseq' which transforms it to temporal geometric point. These temporally and geometrically arranged set of points are now passed to another MobilityDB function named 'trajectory' which finally creates trajectory of latitude and longitudes recorded during movement of TPL vehicles. The instance query creating trajectory of sample trip ID '12379' from table named 'Analytics\_Set\_7' is mentioned below:

```
select "vehicleReg", trip_id, trajectory(tgeompointseq(array_agg(tpoint order by gpstime))) as Traj FROM public."Analytics_Set_7" where trip_id=12379 group by "vehicleReg", trip_id;
```

Table 2.6. Duplication removal query and utility algorithm.

Step	PostgreSQL Query	Remarks
1.	<pre>Select "vehicleReg",gpstime, count(*) Duplication_Count into public."Analytics_Set_1_Dup" from public."Analytics_Set_1" group by "vehicleReg", gpstime having count(*) &gt; 1</pre>	<p>Copy all duplicate records into a temporary table 'Analytics_Set_1_Dup'.</p>
2.	<pre># import module import psycopg2 conn = psycopg2.connect(     database="postgres", user='postgres',     password='postgres', host='127.0.0.1', port='15432') cur = conn.cursor() query = """select * from public."Analytics_Set_1_dup";""" cur.execute(query) table_data = cur.fetchall() # iterate the list of tuple rows for num, row in enumerate(table_data):     sql_update_query = "delete from public."""Analytics_Set_1"" where id = (select max(id) from public."""Analytics_Set_1"" where ""vehicleReg"" = '%s' and ""gpstime"" = '%s')" % (     row[0],row[1])     cur.execute(sql_update_query)     conn.commit() # close cursor objects to avoid memory leaks cur.close() # close the connection object to avoid memory leaks conn.close() # This is OK print('Job Done.....')</pre>	<p>Source code of data cleaning utility.</p> <p>Note: PostgreSQL query mentioned in step 1 and this tool were executed against all data tables sequentially by updating table name.</p>

To store generated trajectories along with their associated information from each of 12 data segments named Analytics\_Set\_1, Analytics\_Set\_2, Analytics\_Set\_3, Analytics\_Set\_4, Analytics\_Set\_5, Analytics\_Set\_6, Analytics\_Set\_7, Analytics\_Set\_8, Analytics\_Set\_9, Analytics\_Set\_10, Analytics\_Set\_11, Analytics\_Set\_12 following 12 additional data segments named Analytics\_Trips\_1, Analytics\_Trips\_2, Analytics\_Trips\_3, Analytics\_Trips\_4, Analytics\_Trips\_5, Analytics\_Trips\_6, Analytics\_Trips\_7, Analytics\_Trips\_8, Analytics\_Trips\_9, Analytics\_Trips\_10, Analytics\_Trips\_11, Analytics\_Trips\_12 are created are created with the help of following PostgreSQL schema only by changing table names mentioned above each time: -

```

-- Table: public.Analytics_Trips_1
DROP TABLE IF EXISTS public."Analytics_Trips_1";
CREATE TABLE IF NOT EXISTS public."Analytics_Trips_1"
(
    trip_id integer,
    "vehicleReg" character varying(100) COLLATE pg_catalog."default",
    trip tgeompoint,
    traj geometry,
    trip_start date,
    trip_stop date,
    trip_span period
)

TABLESPACE pg_default;
ALTER TABLE IF EXISTS public."Analytics_Trips_1"
    OWNER to postgres;
GRANT ALL ON TABLE public."Analytics_Trips_1" TO postgres;
GRANT ALL ON TABLE public."Analytics_Trips_1" TO PUBLIC;

```

Trajectories on complete dataset are generated and stored into newly created tables with the help of insert / select PostgreSQL query for 'Analytics\_Trips\_1' table given below: -

```
insert into public."Analytics_Trips_1"  
select trip_id, "vehicleReg", tgeompointseq(array_agg(tpoint order by  
gpstime))  
,trajectory(tgeompointseq(array_agg(tpoint order by gpstime))) Traj  
,min(gpstime),max(gpstime), period(min(gpstime), max(gpstime))  
FROM public."Analytics_Set_1" group by "vehicleReg", trip_id
```

Table 2.7. Outlier records.

<b>Sr No</b>	<b>Month</b>	<b>Duplicate</b>	<b>Outliers</b>	<b>Len = 0 Hrs</b>	<b>Len &gt; 48 Hrs</b>	<b>Invalid Lat / Long</b>
(a)	Jan,2016	0	500	10	62	22
(b)	Feb,2016	50	338	04	71	17
(c)	Mar, 2016	54	145	31	224	05
(d)	Apr, 2016	59	30	12	80	24
(e)	May,2016	01	44	06	82	08
(f)	Jun,2016	70	714	09	90	13
(g)	Jul,2016	47	101	06	87	06
(h)	Aug,2016	86	692	15	89	10
(j)	Sep,2016	65	124	09	71	09
(k)	Oct,2016	50,386	563	02	28	04
(m)	Nov,2015	0	966	04	27	21
(n)	Dec,2015	0	596	23	52	09

### **3 RESULTS AND DISCUSSIONS**

#### **3.1 Exploratory Data Analysis (EDA)**

Insurance industry is still small in Pakistan as compared to its competitors in the region. Many other companies including State Life Corporation, Jubilee Life, EFU Life are working in insurance industry in Pakistan, however, TPL Insurance is the leading insurance company which is providing its services in various fields including car insurance since 2005. Dataset used bears the movement of TPL insured vehicles within the road network of Pakistan. It includes trips performed round the clock and seven days a week by 107 vehicles. Generally, there is no fix rules for data explanation and analysis in data sciences, however, EDA mainly depends upon data dynamics. Table 3.1 shows trips performed from Nov, 2015 till Oct, 2016 in each monthly data segment with total number of GPS points recorded during movement. Figure 3.1 indicate the data reduction while trips generation using GPS points. Figure 3.2 shows one of the promising characteristics of MobilityDB which is data size reduction. MOD generally bears large data set due to storage of dedicated record in database against every GPS point recorded. Moreover, statistical distribution analysis in Figure 3.2 indicates three micro mobility descriptors namely duration of trips performed, total distance travelled and average speed while performing trips in each month. We further compare the actual trips performed and number of hours consumed in performing those trips, as shown in Figure 3.3.

Table 3.1. Dataset description after data cleaning.

Sr No	Data Segment	Duration	Trips	Points (Mil)	Distance KM	Hrs driven	Size (MB)
(a)	Analytics_Trips_1	Jan,2016	19291	6.2	235264	14766	21
(b)	Analytics_Trips_2	Feb,2016	21741	6.8	297657	17862	23
(c)	Analytics_Trips_3	Mar, 2016	24290	8.1	611115	26051	27
(d)	Analytics_Trips_4	Apr, 2016	23935	8.5	396840	21127	27
(e)	Analytics_Trips_5	May,2016	25187	9.5	409696	21861	30
(f)	Analytics_Trips_6	Jun,2016	22510	8.9	375197	20306	27
(g)	Analytics_Trips_7	Jul,2016	22714	8.6	360607	19407	27
(h)	Analytics_Trips_8	Aug,2016	24490	9.4	380693	20328	29
(j)	Analytics_Trips_9	Sep,2016	22824	8.3	326675	16632	27
(k)	Analytics_Trips_10	Oct,2016	16411	4.7	198210	10317	17
(m)	Analytics_Trips_11	Nov,2015	19243	5.0	201033	11306	20
(n)	Analytics_Trips_12	Dec,2015	18417	6.1	218095	13241	20

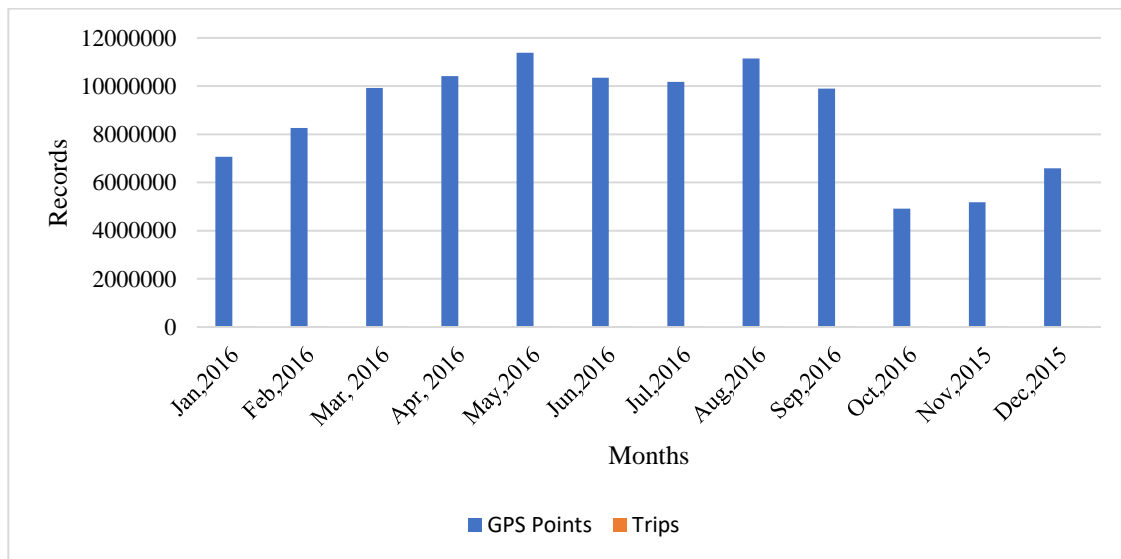


Figure 3.1. Record reduction with Trip generation

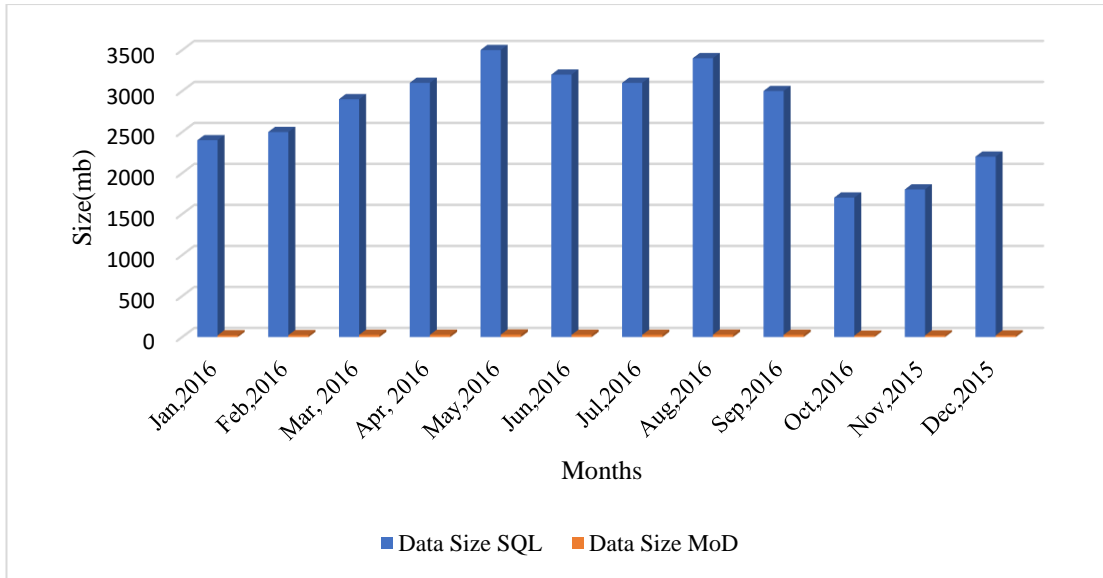


Figure 3.2. SQL to MobilityDB data compression ratio.

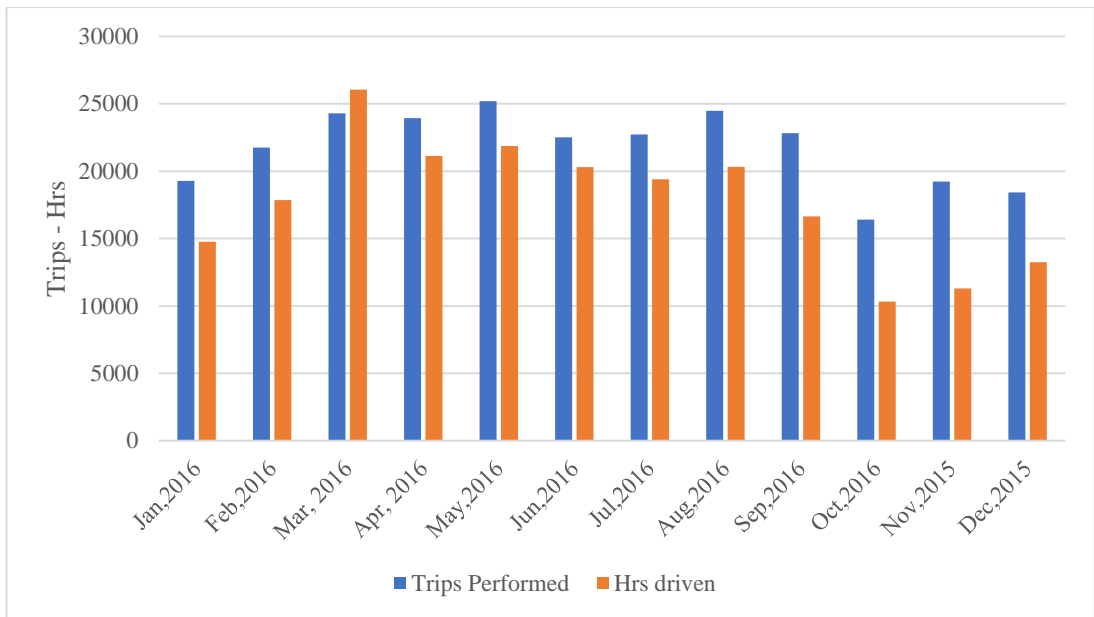


Figure 3.3. Comparison between hours driven and actual trips.



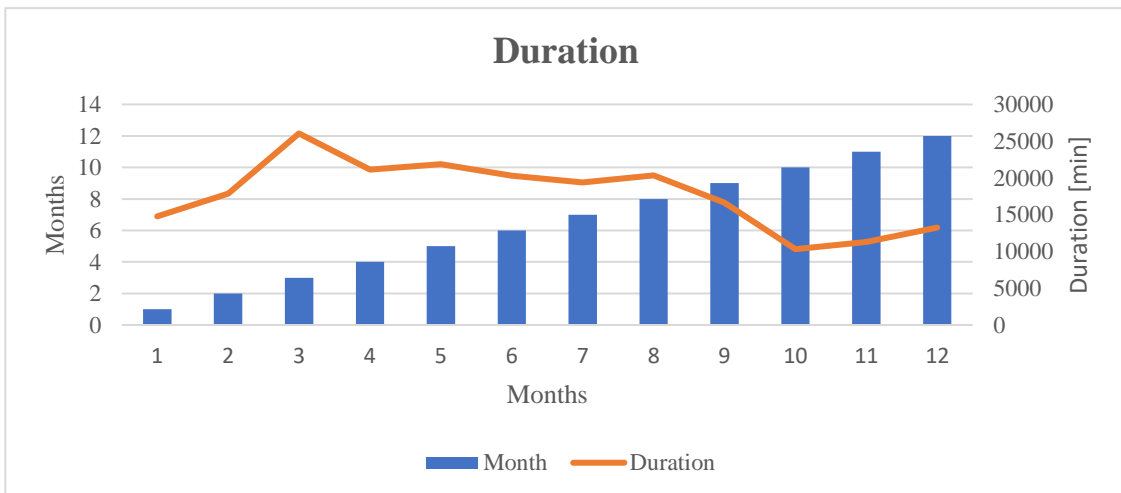
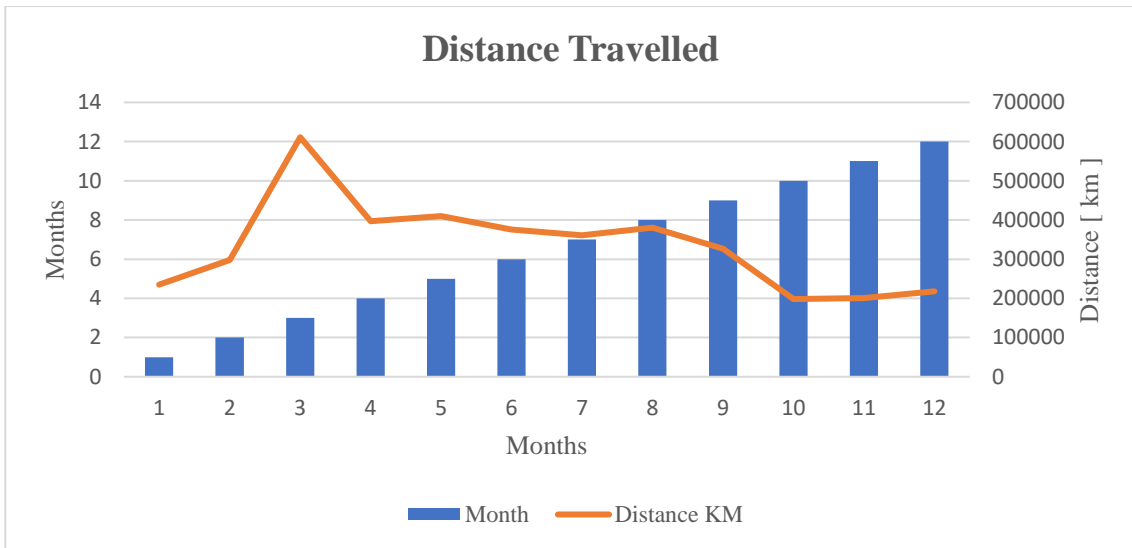
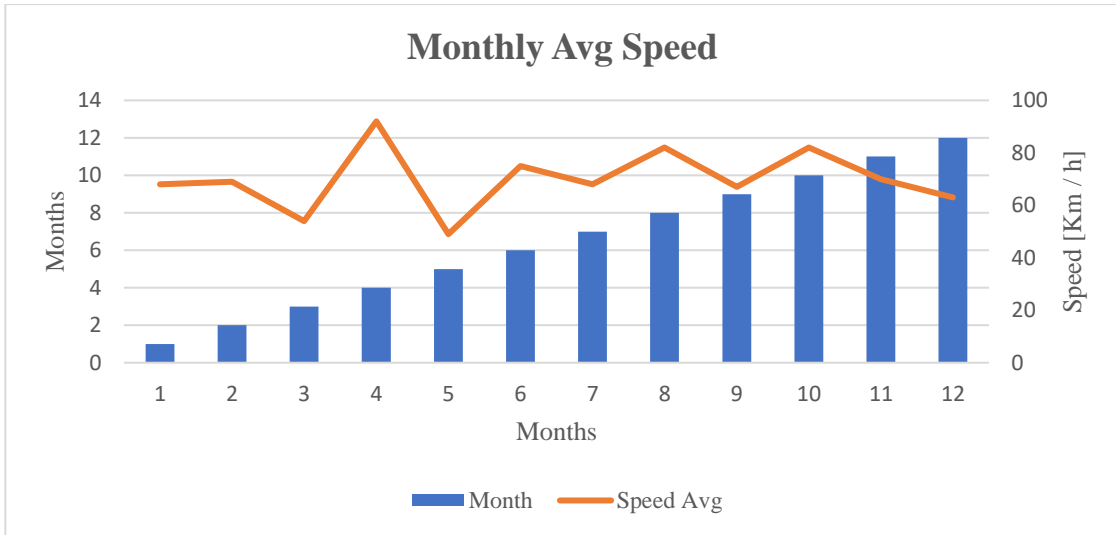


Figure 3.4. Three micro-mobility descriptors.

Query used to calculate number of trips and total number of points recorded in each trip is as follows:

```
SELECT COUNT(*) as NumTrips, sum(numinstants(trip)) as numPoints from public."Analytics_Trips_x"
```

Calculation of total hours consumed in each month to perform trips is carried out in two steps as mentioned in Algorithm 4.

Geographic coordinates are not like coordinates in Mercator, State plane or UTM and they do not give linear distance between two points. These spherical coordinates represent angular coordinates on the globe. The distance calculated in Geographic coordinate system considers curvature of earth. Contrary to Geographic, Geometric reference system is based on plane 2D surface. To get more accurate distance travelled in each month, geometric trajectories are converted into geographic reference system in Table 3.1 which give results in meters. PostgreSQL and MobilityDB queries used to calculate travelled distance for each data table are given below: -

```
Select st_length(traj::geography)::int / 1000 from public."Analytics_Trips_x";
```

Same results may be obtained by using MobilityDB function ‘length’ as mentioned in following query: -

```
Select round(Sum(length(trip)*100) :: int, 0) from public."Analytics_Trips_x";
```

There may be slight difference in the calculation due to different mathematical algorithms used by Geometric and Geographic Coordinate system. Moreover, table size was calculated with the help of information schema using PostgreSQL function named ‘pg\_relation\_size’. This function returns values in Bytes which further converted to Mega Bytes (MBs) for easy understanding: -

```
select table_name, (pg_relation_size(quote_ident(table_name)) / 1024) /  
1024 as Size_MB from information_schema.tables where table_schema  
= 'public' order by table_name
```

### **3.2 Histogram Analysis**

This section would elaborate data distribution with the help of Histogram.

It would be completed in 02 steps given below: -

#### **Step 01: Data Consolidation:**

Each month's data segments is summarized with respect to trips performed and copied into table named 'analytics\_trips\_all' with the help of SQL schema 01.

#### **Step 02: Histogram Generation:**

This section summarizes the distribution of dataset into 07 groups and assigns each group the occurrence frequency with respect to the trips performed and number of GPS points collected in each trip. It would help understand the length of trips and their frequency in whole dataset by making histogram for each monthly segment with the help of schema 02. Output as Histogram is mentioned in Appendix 05.

**Schema 01.**

```
SELECT x.*
INTO analytics_trips_all
FROM (Select 1 as month_id,trip_id, COUNT(*) from
public."Analytics_Set_1" group by trip_id union
Select 2 as month_id,trip_id, COUNT(*) from public."Analytics_Set_2" group
by trip_id union
Select 3 as month_id,trip_id, COUNT(*) from public."Analytics_Set_3" group
by trip_id union
Select 4 as month_id,trip_id, COUNT(*) from public."Analytics_Set_4" group
by trip_id union
Select 5 as month_id,trip_id, COUNT(*) from public."Analytics_Set_5" group
by trip_id union
Select 6 as month_id,trip_id, COUNT(*) from public."Analytics_Set_6" group
by trip_id union
Select 7 as month_id,trip_id, COUNT(*) from public."Analytics_Set_7" group
by trip_id union
Select 8 as month_id,trip_id, COUNT(*) from public."Analytics_Set_8" group
by trip_id union
Select 9 as month_id,trip_id, COUNT(*) from public."Analytics_Set_9" group
by trip_id union
Select 10 as month_id,trip_id, COUNT(*) from public."Analytics_Set_10"
group by trip_id union
Select 11 as month_id,trip_id, COUNT(*) from public."Analytics_Set_11"
group by trip_id union
Select 12 as month_id,trip_id, COUNT(*) from public."Analytics_Set_12"
group by trip_id ) x
```

**Schema 02.**

*With Groups (GroupNo, GroupRange) AS (*

*Select 1, intrange (0, 2) Union*

*Select 2, intrange (2, 10) Union*

*Select 3, intrange (10, 50) Union*

*Select 4, intrange (50, 200) Union*

*Select 5, intrange (200, 500) Union*

*Select 6, intrange (500, 1000) Union*

*Select 7, intrange (1000, 1000000)),*

*TripCount (month\_id,trip\_id, no\_observ) AS (*

*Select month\_id,trip\_id, sum(count) from analytics\_trips\_all group by*

*month\_id,trip\_id), GroupTrip (month\_id,GroupNo, GroupRange, trip\_id) AS (*

*Select month\_id,GroupNo, GroupRange, trip\_id from Groups left outer join*

*TripCount ON no\_observ::int <@ GroupRange), histogram*

*(month\_id,GroupNo, GroupRange, freq) AS (Select month\_id,GroupNo,*

*GroupRange, Count(\*) from GroupTrip group by month\_id,GroupNo,*

*GroupRange order by month\_id,GroupNo, GroupRange)*

*select month\_id,GroupNo, GroupRange,freq, repeat ('█', (freq::float / max(freq)*

*OVER() \* 30)::int) AS bar from histogram*

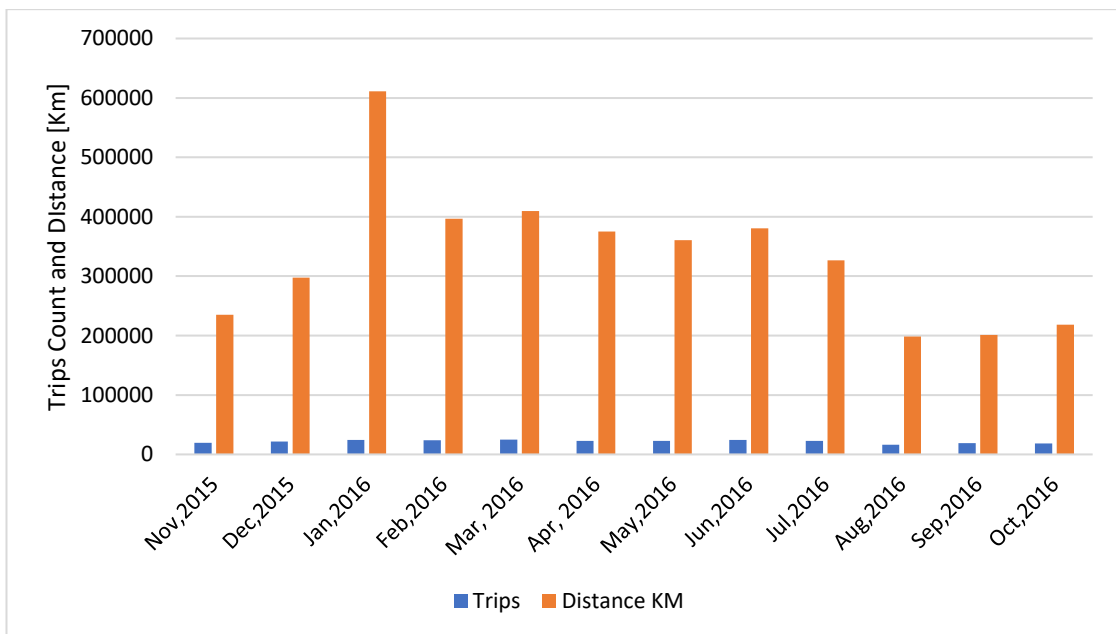


Figure 3.5. Comparison of total distance travelled with total trips performed.

### 3.3 Spatiotemporal Analyses

Temporal data is always large in volume due to continuous recording of GPS points during movement of objects. In addition to the management of spatiotemporal trajectory data, MobilityDB (Syed, 2014) reduces the size of data by generating sequential temporal trips and trajectories and improved the query processing time. Same can be concluded upon comparing results from Table 3.1 and Table 2.8. Keeping in view the volume and complexity of mobility data, further analysis will be performed on data segment named ‘Analytics\_Trips\_1’ bearing trips data performed during January 2016. It is pertinent to mention that data segments from ‘Analytics\_Trips\_1’ to ‘Analytics\_Trips\_12’ have same schema and all queries related to ‘Analytics\_Trips\_1’ would also be applicable to other data tables. Following PostgreSQL queries using MobilityDB functions will further emphasize the strength and importance of MobilityDB. These queries will try to extract useful information in a precise and simple way for making this study self-contained.

### 3.4 Applicability scenarios

**Scenario 01:** Which vehicles performed trips during specific time frame?

**Explanation:** These are temporal range queries where all those trips will be extracted which overlap with given time frame. In the following query, period ('2016-01-01 20:00', '2016-01-01 20:10') was applied on temporal geometric point array column named ‘trip’ with type ‘tgeompoint’.

*Select distinct "vehicleReg", trip\_id, trip\_span from public."Analytics\_Trips\_1"  
where trip && period ('2016-01-01 20:00', '2016-01-01 20:10')*

Table 3.3 shows results with active trips during the specified time frame. We can measure the speed of moving objects by using speed function applied over

trajectory column in the same query. However, interpretation of temporal float results would be little tedious due to stepwise array of points along with changing speed over time. Figure 3.6 shows the visualization of temporal results from Scenario 01 on QGIS. It is pertinent to mention that QGIS plugin 'Move' is capable of interpreting only simple queries and produces erroneous errors on complex MobilityDB queries. This problem can be solved by copying results while executing complex queries into a separate table in MobilityDB for further accessibility of this data in QGIS by simple select statement. Figure 3.7 shows the temporal range in QGIS temporal controller for visual movement of results obtained in result of query pertaining to Scenario 01.

Table 3.2. Temporal range query output.

<b>S No</b>	<b>Vehicle ID</b>	<b>Trip ID</b>	<b>Trip Life / Span</b>
1	TLY 863	13960	[2016-01-01 07:33:08+00, 2016-01-02 12:57:34+00]
2	TLZ 792	18205	[2016-01-01 10:54:09+00, 2016-01-01 21:21:36+00]
3	TLZ 692	16805	[2016-01-01 10:54:52+00, 2016-01-02 11:09:26+00]
4	TLV 882	11912	[2016-01-01 13:33:56+00, 2016-01-01 20:03:49+00]
5	TLZ 361	15681	[2016-01-01 16:15:42+00, 2016-01-01 23:48:17+00]
6	TLV 872	10705	[2016-01-01 17:33:08+00, 2016-01-01 20:01:03+00]
7	TLY 903	14006	[2016-01-01 17:46:26+00, 2016-01-01 21:05:22+00]
8	TLY 361	13680	[2016-01-01 18:05:14+00, 2016-01-01 20:28:07+00]
9	TLT 123	5322	[2016-01-01 18:34:59+00, 2016-01-01 20:32:23+00]
10	TLZ 561	16453	[2016-01-01 19:03:02+00, 2016-01-01 20:16:56+00]
11	TLU 363	6897	[2016-01-01 19:03:10+00, 2016-01-01 20:56:35+00]
12	TLV 361	9789	[2016-01-01 19:06:20+00, 2016-01-01 20:13:38+00]
13	TLV 250	9586	[2016-01-01 19:10:24+00, 2016-01-01 21:12:53+00]
14	TLW 292	12753	[2016-01-01 19:25:35+00, 2016-01-01 22:40:38+00]
15	TLV 870	10473	[2016-01-01 19:29:33+00, 2016-01-01 21:27:16+00]
16	FK 274	2053	[2016-01-01 19:31:52+00, 2016-01-01 21:13:07+00]
17	TLU 663	8178	[2016-01-01 19:39:21+00, 2016-01-01 20:45:16+00]
18	TLV 878	10995	[2016-01-01 19:40:33+00, 2016-01-01 21:07:19+00]
19	TLX 061	13051	[2016-01-01 19:40:43+00, 2016-01-01 20:41:24+00]
20	TLU 417	7283	[2016-01-01 19:56:42+00, 2016-01-01 20:22:56+00]
21	TLV 450	10072	[2016-01-01 19:59:10+00, 2016-01-02 11:57:31+00]
22	TLU 792	8791	[2016-01-01 20:03:50+00, 2016-01-01 20:04:37+00]
23	TLV 992	12393	[2016-01-01 20:03:52+00, 2016-01-01 20:09:59+00]
24	TLV 872	10706	[2016-01-01 20:04:00+00, 2016-01-01 21:46:21+00]
25	TLV 882	11913	[2016-01-01 20:05:55+00, 2016-01-01 20:17:30+00]



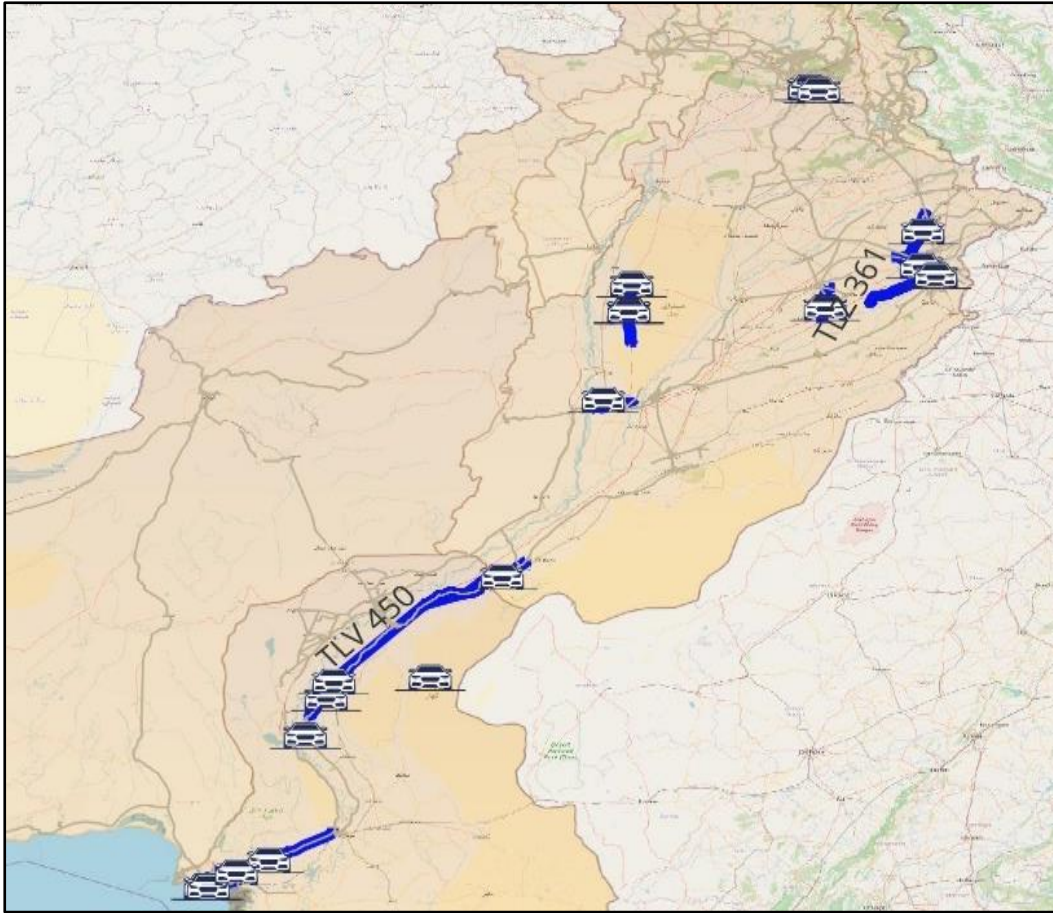


Figure 3.6. All active trips at specified time.

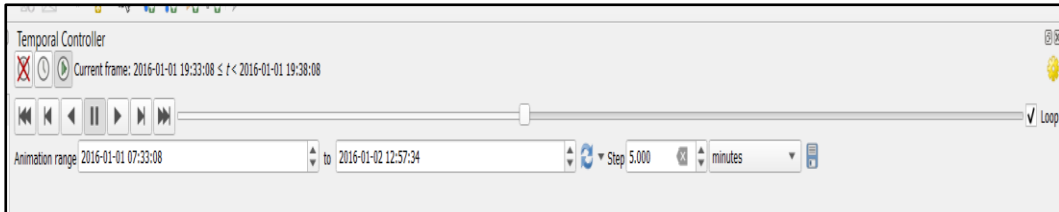


Figure 3.7. Temporal range - QGIS plugin Move.

**Scenario 02:** Which vehicles passed from specified region or area?

**Explanation:** For this query, OSM data was used for Islamabad region in the table named 'ibd\_landuse'. This is an example spatial range query which uses MobilityDB's function named 'intersects' which filters geographical intersection of trips along with geometry of provided region.

```
Select distinct t.vehicleReg,trip_id, trip_span, trip, traj into tbl_q2 from Analytics_Trips_1 t, ibd_landuse m where m.name like '%F-10 Markaz%' AND intersects(t.trip, m.geom)
```

Results shows total 52 trips performed by same vehicle registration number "TLU 663" between 01-01-2016 till 30-01-2016 (Table 3.4) to almost same location. This also means that these trips belong to some specific purpose in nearby location. Following query will be used in MOVE plugin to check temporal visualization given in Figure 3.8.

```
Select vehicleReg,trip_ID, trip,traj from tbl_q2
```

Table 3.4 shows results trips passed from specified location.

Table 3.3. Spatial range query results - scenario 02.

Sr No	Vehicle ID	Trip ID	Trip duration / Span
1	TLU 663	8174	[2016-01-01 05:43:40+00, 2016-01-01 06:20:12+00]
2	TLU 663	8175	[2016-01-01 06:20:45+00, 2016-01-01 06:22:10+00]
3	TLU 663	8176	[2016-01-01 06:45:37+00, 2016-01-01 07:24:07+00]
4	TLU 663	8187	[2016-01-02 00:47:44+00, 2016-01-02 01:33:06+00]
5	TLU 663	8188	[2016-01-02 01:51:41+00, 2016-01-02 02:03:10+00]
6	TLU 663	8174	[2016-01-01 05:43:40+00, 2016-01-01 06:20:12+00]
-	-	-	-
-	-	-	-
51	TLU 663	8771	[2016-01-30 00:22:12+00, 2016-01-30 00:22:35+00]
52	TLU 663	8772	[2016-01-30 00:28:35+00, 2016-01-30 01:07:51+00]



Figure 3.8. QGIS display of results Scenario 2.

Upon further analysis to identify the purpose of same vehicle visiting same area revealed that vehicle visited Shell Filling station location in F-10 sector Islamabad (Figure 3.9).

Scenario 02 can also be answered using K-Nearest Neighbor (KNN) technique which is the basic machine learning algorithm based upon supervised learning mechanism. MobilityDB uses distance operator ‘|=|’ to identify Euclidean distance between two geometries. On the other hand, PostGIS uses ‘<->’ as distance operator which give 2D distance between two-point geometries only. Following MobilityDB query return same 52 results showing all those trips which cross the specified location: -

```
Select distinct trip_id,vehicleReg,(t.trip |=| m.geom) e_distance,traj from Analytics_Trips_1 t , ibd_landuse m where m.name like '%F-10 Markaz%' and (t.trip |=| m.geom) =0;
```

**Scenario 03:** Which vehicles passed nearby given location?

**Explanation:** This is another spatial range example where PostGIS function named ‘st\_dwithin’ will be used to find which trip passes from within specified distance of given geometry. It is pertinent to mention that both the geometries must be in the same Coordinate Reference System (CRS) i.e same SRID which is 4326 in this case. Currently the data is in decimal degrees and 01 degree = 111km. We used here 0.001 meters (0.11 Km) to draw intersecting area. Following query is used to extract the desired information: -

```
SELECT distinct a.vehicleReg,a.trip_ID, a.trip_span, a.trip,a.traj into tbl_q3 FROM analytics_trips_1 a, ibd_points b WHERE st_dwithin(a.traj, b.geom, 0.001) and b.name like '%Burhan Interchange%'
```

Following query will be used in MOVE plugin to check temporal visualization given in Figure 3.10.

```
SELECT vehicleReg,trip_ID, trip,traj from tbl_q3
```

Table 3.5 shows that total **03 vehicles** crossed Burhan Interchange during **09 trips** performed on different time periods.

Figure 3.10 shows the geometrical display of vehicle crossing Burhan Interchange M1 using plugin Move in QGIS.

**Scenario 04 :** Find the location of vehicle at certain time stamp?

**Explanation:** This is temporal identification of location scenario where MobilityDB function ‘valueAtTimestamp’ can easily extract point location from the trip. This function has two parameters. One is the trip from which the point is to extracted and other parameters is the specified time stamp. For instance, following query will extract locations of vehicles on time stamp ‘2016-01-01 20:00:00’: -

```
select vehicleReg, ST_AsText(valueAtTimestamp(trip, timestamptz '2016-01-01 20:00:00')), valueAtTimestamp(trip, timestamptz '2016-01-01 20:00:00') FROM analytics_trips_1 where ST_AsText(valueAtTimestamp(trip, timestamptz '2016-01-01 20:00:00')) is not null
```

Table 3.6 shows that total 21 vehicle’s location at given time stamp on same time stamp. Same is depicted in Figure 3.11.

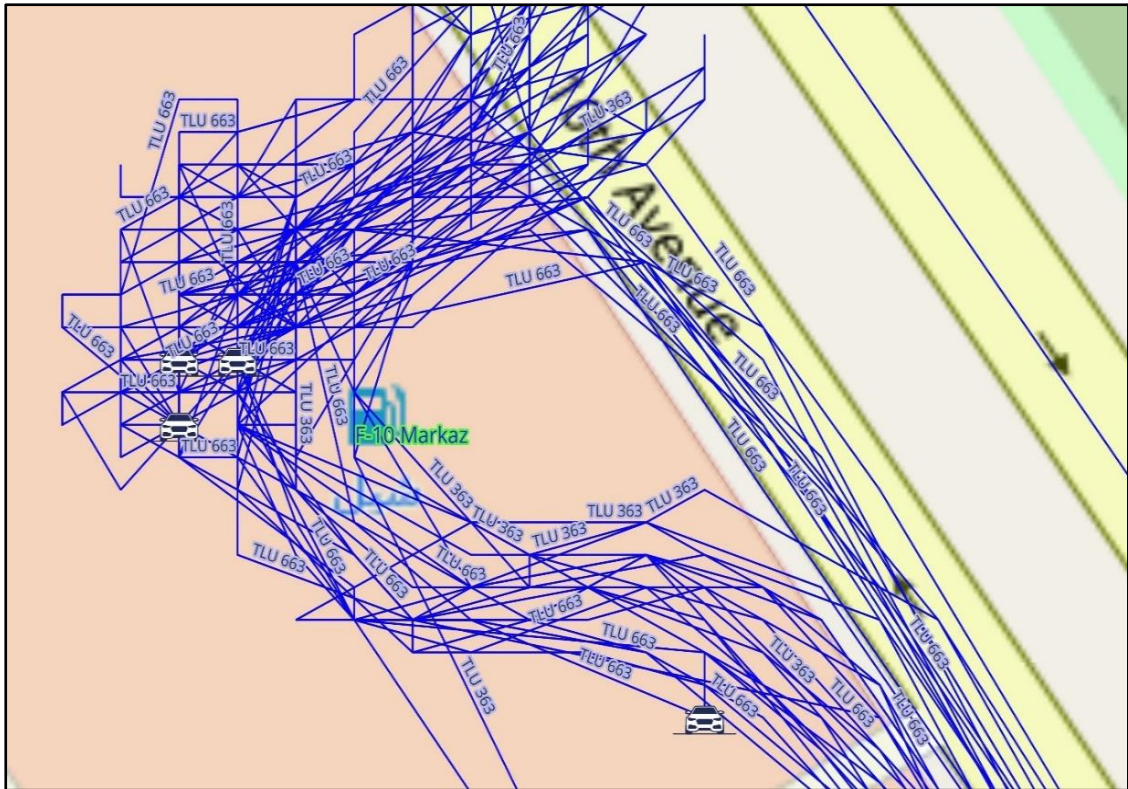


Figure 3.9. Visual display of Scenario 2 result - Petrol filling

Table 3.4. Trip crossing nearby given location

S No	Vehicle ID	Trip ID	Trip duration / Span
1	TLU 363	6907	[2016-01-02 05:11:10+00, 2016-01-02 07:23:26+00]
2		6909	[2016-01-02 08:16:52+00, 2016-01-02 08:58:23+00]
3		7040	[2016-01-13 01:57:40+00, 2016-01-13 02:37:32+00]
4		7041	[2016-01-13 03:29:15+00, 2016-01-13 04:57:08+00]
5		7111	[2016-01-18 07:06:52+00, 2016-01-18 07:36:58+00]
6		7112	[2016-01-18 08:41:55+00, 2016-01-18 10:38:07+00]
7	TLV 879	11488	[2016-01-13 06:45:22+00, 2016-01-13 10:00:26+00]
8		11494	[2016-01-14 10:41:00+00, 2016-01-14 13:59:22+00]
9	TLV 992	12518	[2016-01-12 07:01:40+00, 2016-01-12 09:36:23+00]



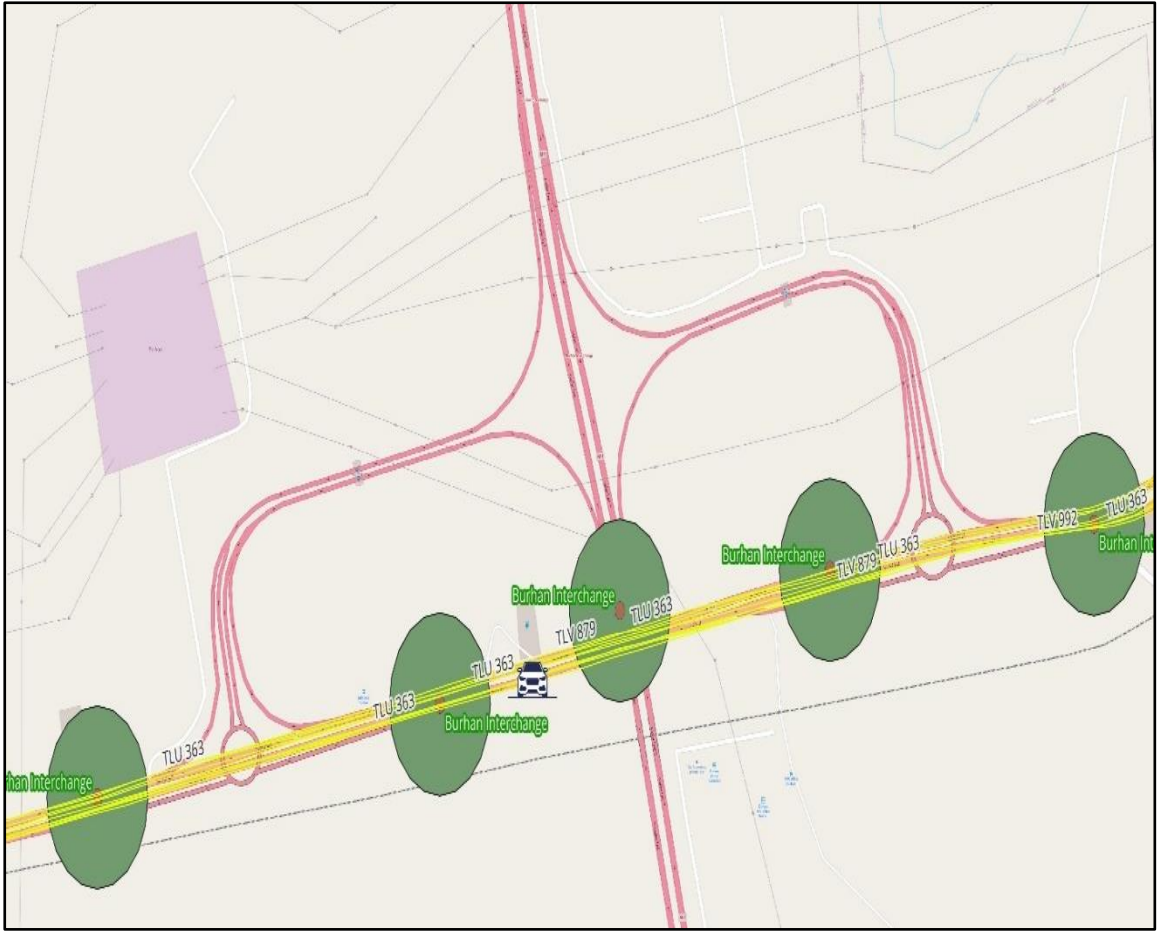


Figure 3.10. Trips crossing Burhan Interchange M1.

Table 3.5. Temporal location identification - scenario 04.

Sr No	Vehicle ID	Trip ID	Type
1	FK 274	2053	POINT(69.2265238776655 27.129102483165)
2	TLU 363	6897	POINT(73.00763333333333 33.62243333333333)
3	TLT 123	5322	POINT(67.0452 24.82856666666667)
4	TLU 663	8178	POINT(73.1032213761822 33.5996701071222)
5	TLU 417	7283	POINT(66.9821 24.81771666666667)
6	TLV 882	11912	POINT(67.98106666666667 26.50836666666667)
-	-	-	-
-	-	-	-
19	TLZ 792	18205	POINT(66.9826086513589 24.8302188295064)
20	TLZ 561	16453	POINT(71.20886666666667 31.1199875)
21	TLZ 692	16805	POINT(66.98256666666667 24.83023333333333)

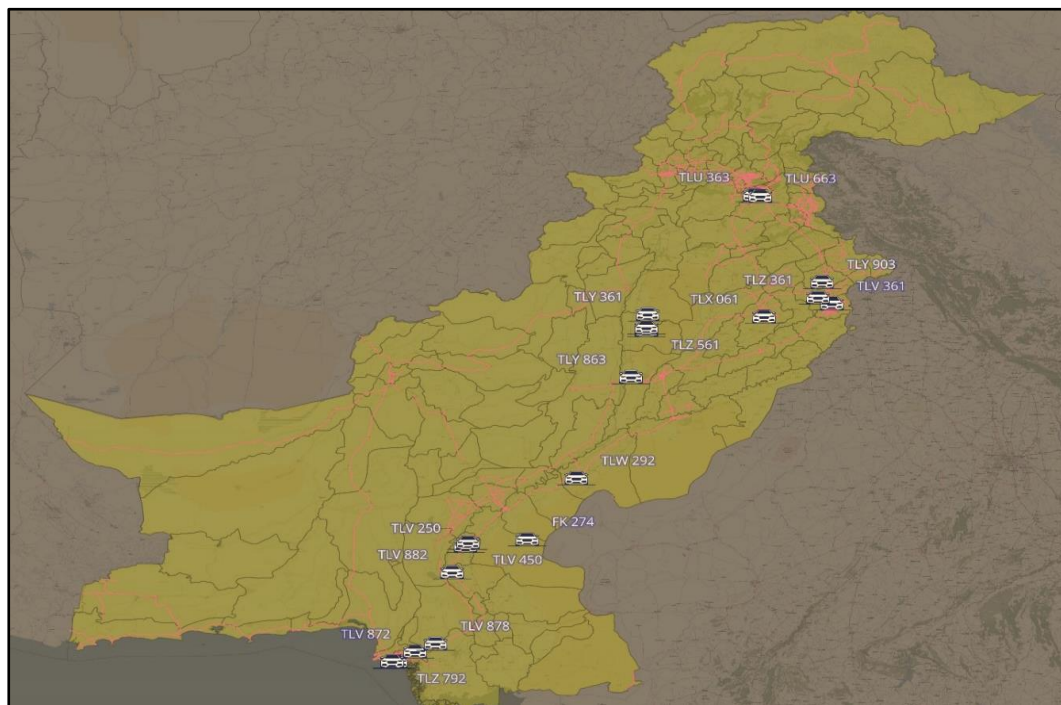


Figure 3.11. Spatiotemporal locations of active vehicles.



**Scenario 05:** Identify the speed pattern of any trip?

**Explanation:** Speed of moving objects generally changes with the passage of time. To identify step wise non-linear speed pattern, MobilityDB's function named speed is used. This function takes trajectory as input and splits the speed intervals in stepwise manner. For instance, following query will extract the desired output: -

*Select speed(trip) from Analytics\_Trips\_1 where trip\_id=7040*

Figure 3.12 shows the speed interval of a trip (trip id 7040) where after start of the trip, vehicle achieves speed interval of 8.53 km /hr. from 01:57:40 till 01:57:45. Then speed drop and reaches to zero km / hr. This again changes on 02:00:50 at the rate of 2.98 km / hr. Similarly, speed interval of whole trip can be accessed with the help of function named 'speed'.

Figure 3.13 shows the visual display of trip id 7040, where upon visual analysis, it is revealed that the vehicle after ignition just want to enter on a two-lane highway (Express Way – Islamabad). Driver upon reaching the verge of lane entrance, waited to take a suitable pause in coming traffic. Upon getting the desired traffic pause, entered on highway and continues his journey.

```

Interp=Stepwise;[0.00016996731712032954@2016-01-13 01:57:40+00,
8.537498983416246e-05@2016-01-13 01:57:45+00, 0@2016-01-13 01:57:50+00,
0.001782710171495535@2016-01-13 01:58:45+00,
0.00014051492605512557@2016-01-13 01:58:50+00,
0.00014126413400297673@2016-01-13 01:58:55+00,
0.0001467045405643012@2016-01-13 01:59:10+00,
0.0001546321800641485@2016-01-13 01:59:20+00,
0.00015333333333273914@2016-01-13 01:59:30+00,
0.00015365907428776675@2016-01-13 01:59:40+00,
0.00015095171331590314@2016-01-13 01:59:50+00,
0.00015202339001309259@2016-01-13 02:00:10+00,
0.00018439088914581333@2016-01-13 02:00:20+00,
2.9814239699715866e-05@2016-01-13 02:00:30+00,
5.537498983416246e-05@2016-01-13 02:00:40+00,
6.266268265058564e-05@2016-01-13 02:09:11+00,
9.614803401241173e-05@2016-01-13 02:09:55+00,
11.0001639783183490728@2016-01-13 02:10:00+00,
11.6.0001401982723002658@2016-01-13 02:10:05+00,
20.0001556349003986163@2016-01-13 02:10:10+00,
22.00015377653339332454@2016-01-13 02:10:15+00,

```

Figure 3.12. Stepwise varying non-linear speed interval of trajectory.



Figure 3.13. Stepwise speed interval visualization.

**Scenario 06:** Find count of vehicles at every instant between 20:00 and 20:10?

**Explanation:** This is a temporal aggregation pattern query, where we want to identify all active trips between given time stamp i.e '2016-01-01 20:00' and '2016-01-01 20:10'. Following query return the varying number of periods with active trips (Figure 3.14): -

```
Select tcount(atperiod(trip, period('2016-01-01 20:00','2016-01-01 20:10')))
Num_Trip from Analytics_Trips_1 where trip && period('2016-01-01
20:00','2016-01-01 20:10')
```

**Scenario 07:** What was the closest distance between any vehicle and given location (Blue Area) on a given Date?

**Explanation:** This is a spatiotemporal K-NN query, where we want to identify the closest location or direction between any vehicle and given location. For instance, 'Blue Area' location was selected in the given query and MobilityDB's distance function named '|=' is used to calculate the minimum distance between selected trajectory and given location: -

```
Select MIN(trip |=/ m.geom) as distancee,t.vehicleReg, t.traj, t.trip_span from
Analytics_Trips_1 t , ibd_landuse m where m.name like '%Blue Area%' AND
intersects(t.trip, m.geom) and t.trip && period ('2016-01-06', '2016-01-07')
group by t.vehicleReg, t.traj, t.trip_span
```

Table 3.7 illustrate a vehicle nearest location during 04 trips with minimum distance during the specified time. Same is depicted in Figure 3.15, where a vehicle with registration number 'AH 391' visited 'Blue Area' location same day about 04 times.

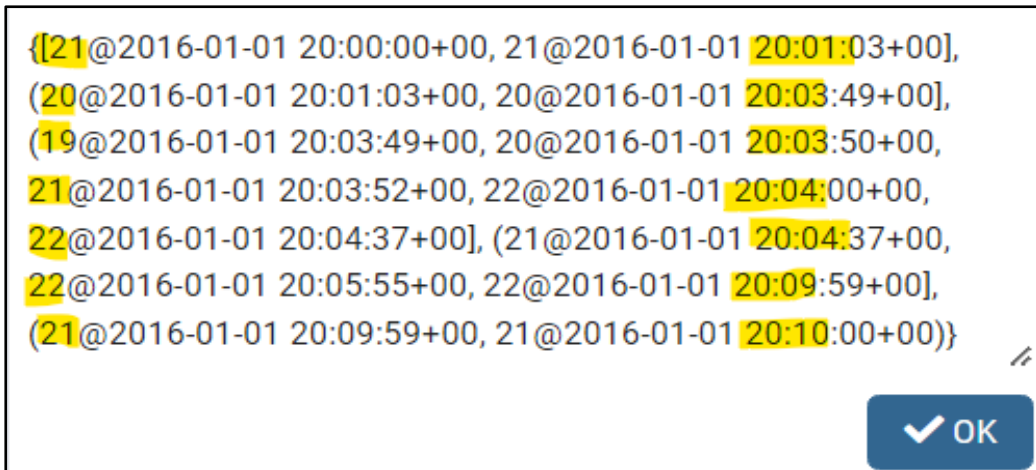


Figure 3.14. Temporally active trips

Table 3.6. Result of spatiotemporal K-NN query

Reg No	Trip ID	Distance	Duration
AH 391	5	0	[2016-01-06 02:00:49+00, 2016-01-06 02:31:28+00]
	9	0	[2016-01-06 15:20:16+00, 2016-01-06 15:30:14+00]
	10	0	[2016-01-06 15:47:45+00, 2016-01-06 15:54:50+00]
	11	0	[2016-01-06 15:59:21+00, 2016-01-06 16:03:22+00]

**Scenario 08:** How many trips start and finish in different areas in the same location (Islamabad)?

**Explanation:** This is another spatiotemporal K-NN intersection query, where task is to identify all those trips which started and finished in the given location i.e Islamabad. Following query used PostGIS function named ‘*ST\_Intersects*’ which further used two geometries. MobilityDB functions ‘*startValue*’ and ‘*endValue*’ returned the start and end geometries of trajectories which further intersected with the given geometry (Islamabad).

```
Select t.vehicleReg,t.trip_id,t.traj,t.trip_span from Analytics_Trips_1 t ,
ibd_landuse m where ST_Intersects(startValue(t.trip),m.geom) AND
ST_Intersects(endValue(t.trip),m.geom)
```

Table 3.8 shows total 104 trips which were started and finished in Islamabad.

Same is displayed in Figure 3.16.

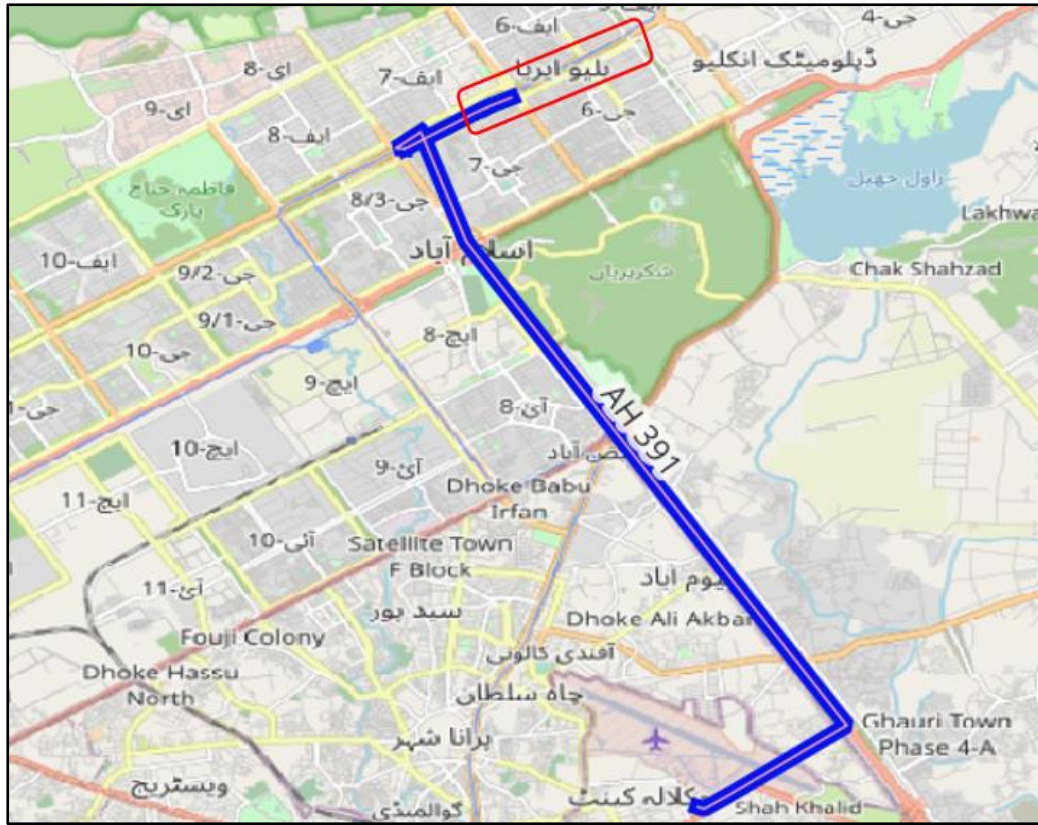


Figure 3.15. Spatiotemporal K-NN display.

Table 3.7. Result set - Scenario 08

Sr No	Trip ID	Distance	Duration
1	AH 391	16	[2016-01-07 07:36:52+00, 2016-01-07 07:36:58+00]
2	DV 748	2024	[2016-01-30 12:29:08+00, 2016-01-30 12:30:30+00]
3	TLU 363	6976	[2016-01-08 02:51:25+00, 2016-01-08 02:54:15+00]
4	TLW 292	12837	[2016-01-13 06:13:17+00, 2016-01-13 06:14:56+00]
5	TLY 361	13739	[2016-01-09 00:58:19+00, 2016-01-09 00:58:45+00]
-	-	-	-
-	-	-	-
-	-	-	-
103	VA 489	19078	[2016-01-27 14:40:43+00, 2016-01-27 14:43:17+00]
104	ZF 531	19210	[2016-01-19 15:23:29+00, 2016-01-19 15:26:11+00]



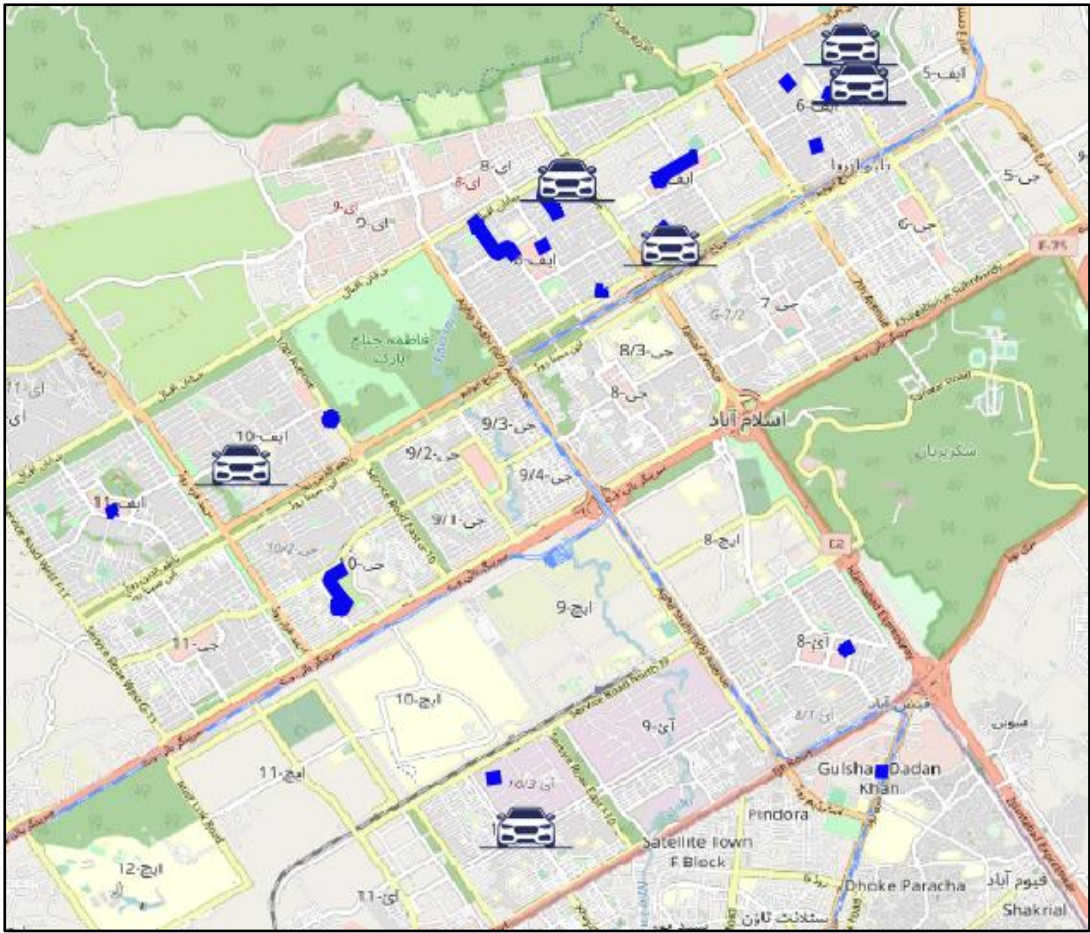


Figure 3.16. Visual display of scenario 08.

## **4 CONCLUSIONS AND RECOMMENDATIONS**

### **4.1 Conclusions**

In this study, we have created a deep understanding of MobilityDB which is mainly based upon its configurations till implementation using moving vehicles data set of an insurance company named TPL Pakistan Pvt Ltd in a phased manner. Complete insightful steps were also made part of the study starting from data loading, pre-processing and analysis in order to make the document self-contained for readers. We comprehensively analyzed the implementation scenarios of MobilityDB and identified some major potential areas in Pakistan which need to be automated with MobilityDB. Multiple use cases were analyzed for moving and static vehicles to extract decision support information. Based upon these practical scenarios, it is evident that MobilityDB is the backbone and only moving object database system in mobility automation. GIS based visualization created in an open-source plugin named 'Move' in QGIS 3.22 precisely exhibit the utilization of MobilityDB for Mobility data. The results achieved with the help of real-life scenarios can be vital for multiple public and private sectors. Table 4.1 shows some of the major MobilityDB implementation sectors in Pakistan which can benefit with MobilityDB. To achieve true benefits of MobilityDB, more research and elaborated documentation with implementation scenarios need to be carried out in addition to the availability of mobility datasets.



Table 4.1. Application areas in Pakistan.

<b>S No</b>	<b>Sector</b>	<b>Department</b>
1	Transport Infrastructure	National Highway Authority
2	Innovative Automation using Mobility Data	Public & Govt Offices
3	Congestion Index (CI) Forecasting and Mitigation	Planning & Development
4	Equity (Population, demographic and socio-economic)	
5	Spatiotemporal Traffic Analysis	Traffic Police
6	Mobility Patterns and Commute analysis	Research & Development
7	Crash Ratio and Risk identification	Rescue & NDMA
8	Carbon Emission Contribution (CEC)	Environmental Protection Department
9	Transit Categorization	Provincial Mass Transit Authority
10	Emergency / Crime Management	Law Enforcement Agencies

## **4.2 Recommendations for further research**

This study mainly focuses on analysis of 2D datasets using mobilityDB, however, there is still research gap in application of mobilityDB using 3D datasets which involve altitude variable. Future trajectory prediction for both long term and short-term mobility agents is another potential area to be explored by the research community. Currently future trajectory is predicted by feeding spatiotemporal datasets to machine learning algorithms, however, MobilityDB application in this regard can be further explored. Mobility data is generated by GPS equipped moving objects in real time resulting large number of records on servers which need to be processed in an efficient and optimized way. Citus, an extension to MobilityDB, is another area of research to process large spatiotemporal datasets in distributed environment.

## 5 REFERENCES

1. Ali, A., & Imran, M. (2020). The Evolution of National Spatial Data Infrastructure in Pakistan-Implementation Challenges and the Way Forward. *International Journal of Spatial Data Infrastructures Research*, 15, 110-142.
2. Beller, A. (1991). A temporal GIS prototype for global change research. In *Proceedings of GIS/LIS*.
3. Godfrid, J., Radnic, P., Vaisman, A., & Zimányi, E. (2022). Analyzing public transport in the city of Buenos Aires with MobilityDB. *Public Transport*, 1-35.
4. Graser, A., & Dragaschnig, M. (2020). Exploring movement data in notebook environments. In *IEEE VIS 2020 Workshop on Information Visualization of Geospatial Networks, Flows and Movement (MoVis)*.
5. Imran, M. (2009). Public transport in Pakistan: a critical overview. *Journal of Public Transportation*, 12(2), 4.
6. Rovinelli, G., Matwin, S., Pranovi, F., Russo, E., Silvestri, C., Simeoni, M., & Raffaetà, A. (2021). *Multiple aspect trajectories: a case study on fishing vessels in the Northern Adriatic sea*. Paper presented at the EDBT/ICDT Workshops.
7. Georgiou, H., Karagiorgou, S., Kontoulis, Y., Pelekis, N., Petrou, P., Scarlatti, D., & Theodoridis, Y. (2018). Moving objects analytics: Survey on future location & trajectory prediction methods. arXiv preprint arXiv:1807.04639.
8. Güting, R. H., Behr, T., & Xu, J. (2010). Efficient k-nearest neighbor search on moving object trajectories. *The VLDB Journal*, 19(5), 687-714.
9. Schoemans, M., Sakr, M. A., & Zimányi, E. (2022). MOVE: Interactive Visual Exploration of Moving Objects. In *EDBT/ICDT Workshops*.
10. Syed, W. H., Yasar, A., Janssens, D., & Wets, G. (2014). Analyzing the real time factors: which causing the traffic congestions and proposing the solution for Pakistani City. *Procedia Computer Science*, 32, 413-420.
11. Tsesmelis, D. (2021). Big Data Management and Analytics Master Thesis from Université Libre de Bruxelles.

12. Vaisman, A., & Zimányi, E. (2019). Mobility data warehouses. *ISPRS International Journal of Geo-Information*, 8(4), 170.
13. Helland-Hansen, W., & Hampson, G. J. (2009). Trajectory analysis: concepts and applications. *Basin Research*, 21(5), 454-483.
14. Zimányi, E., Sakr, M., & Lesuisse, A. J. A. T. o. D. S. (2020). MobilityDB: A mobility database based on PostgreSQL and PostGIS. 45(4), 1-42.
15. Zimányi, E., Sakr, M., & Lesuisse, A. (2020). MobilityDB: A mobility database based on PostgreSQL and PostGIS. *ACM Transactions on Database Systems (TODS)*, 45(4), 1-42.
16. Zimányi, E., Sakr, M., Lesuisse, A., & Bakli, M. (2019, August). Mobilitydb: A mainstream moving object database system. In *Proceedings of the 16th International Symposium on Spatial and Temporal Databases* (pp. 206-209).
17. Zimányi, E., Sakr, M., Bakli, M., Schomans, M., Tsesmelis, D., & Choquet, R. (2021, November). MobilityDB: hands on tutorial on managing and visualizing geospatial trajectories in SQL. In *Proceedings of the 3rd ACM SIGSPATIAL International Workshop on APIs and Libraries for Geospatial Data Science* (pp. 1-2).

## 6 Appendix-1. Python Based Tool to Generate Trip ID.

```
# import module
import psycopg2
conn = psycopg2.connect(
    database="postgres", user='postgres', password='postgres',
    host='127.0.0.1', port='15432')
cur = conn.cursor()
cur2 = conn.cursor()
query2 = """Select distinct "vehicleReg" from Analytics_cars;"""
cur2.execute(query2)
table_data2 = cur2.fetchall()
tripid = 1
# iterate the list of Cars
for num, row in enumerate(table_data2):
    print(tripid)
    query = "SELECT id,vehiclestatus FROM public.\""Analytics_Set_1\""
    where "\""vehicleReg\"" = '%s' order by gpstime" % (row[0])
    cur.execute(query)
    table_data = cur.fetchall()
    if len(table_data) > 0:
        # iterate the list of tuple rows
        for num, row in enumerate(table_data):
            Status = 'off'
            if (row[1] == 'Start up') or (row[1] == 'Driving') or (row[1] ==
'Start up;(No Driver Id)
            or (row[1] == 'Excess Idle') or (row[1] == 'Speed violation') or
(row[1] == 'GPS unlocked')
            or (row[1] == 'Harsh Braking') or (row[1] == 'Nogo Area') or (row[1]
== 'Speed violation'):
                Status = 'Start up'
                sql_update_query = "Update public.\""Analytics_Set_1\"" set
                "\""tripid\"" = '%s' where id = %s" % (
tripid, row[0])
            else:
                sql_update_query = "Update public.\""Analytics_Set_1\"" set
                "\""tripid\"" = '%s' where id = %s" % (tripid, row[0])
                Status = 'off'
                tripid += 1
                cur.execute(sql_update_query)
                conn.commit()
# close cursor objects to avoid memory leaks
cur.close()
cur2.close()
# close the connection object to avoid memory leaks
conn.close()
print('Job Complete.....')
```

## 7 Appendix-2. SQL curser to generate trip ID.

```
Declare @ID varchar(30);
Declare @CarID varchar(30);
Declare @CarStatus varchar(30);
Declare @PrevStatus varchar(30) = "";
Declare @TripID int = 1;
Declare Cur1 CURSOR FOR
Select distinct "vehicleReg" from Veh_Reg
OPEN Cur1
FETCH NEXT FROM Cur1 INTO @CarID;
WHILE @@FETCH_STATUS = 0
BEGIN
    DECLARE Cur2 CURSOR FOR
        Select id,vehiclestatus from Analytics_Set_1 where trip_id = 0 and
vehicleReg = " + @CarID + " order by gpstime
    OPEN Cur2;
    FETCH NEXT FROM Cur2 INTO @ID,@CarStatus;
    WHILE @@FETCH_STATUS = 0
    BEGIN
        if @CarStatus in ('Start up','Driving','Start up;(No Driver
Id)','Excess Idle','Speed violation','GPS unlocked','Driving;(Unauthorised
Driver)','Harsh Braking','Nogo Area','Nogo Area;(Unauthorised
Driver)','Speed violation','Speed violation;(Unauthorised Driver)','Trailer
Door','Trailer Door;(Unauthorised Driver)')
            begin
                -- Store Prev Status to double check trip closing
                set @PrevStatus = @CarStatus
                update Analytics_Set_1 set trip_id = @TripID where
id = @ID
            end
        FETCH NEXT FROM Cur2 INTO @ID,@CarStatus;
        if @CarStatus in ('Ignition off') and @PrevStatus in ('Start up','Driving','Start
up;(No Driver Id)','Excess Idle','Speed violation','GPS
unlocked','Driving;(Unauthorised Driver)','Harsh Braking','Nogo Area','Nogo
Area;(Unauthorised Driver)','Speed violation','Speed violation;(Unauthorised
Driver)','Trailer Door','Trailer Door;(Unauthorised Driver)')
            begin
                update Analytics_Set_1 set trip_id = @TripID where id = @ID
                set @TripID = @TripID + 1;
                set @PrevStatus = ""
            end
    END;
    CLOSE Cur2;
    DEALLOCATE Cur2;
    FETCH NEXT FROM Cur1 INTO @CarID;
END;
PRINT 'DONE';
CLOSE Cur1;
DEALLOCATE Cur1;
```

## 8 Appendix-3. SQL schema to update columns

```
ALTER TABLE public."Table_Name"  
DROP COLUMN "DriverId",  
DROP COLUMN Location,  
DROP COLUMN Skillset,  
DROP COLUMN C2,  
DROP COLUMN C3,  
DROP COLUMN C4,  
DROP COLUMN C5,  
DROP COLUMN C6,  
DROP COLUMN C7,  
DROP COLUMN C8,  
DROP COLUMN C9,  
DROP COLUMN "distance",  
DROP COLUMN "province",  
DROP COLUMN "country",  
ADD COLUMN "City" character varying COLLATE pg_catalog."default",  
ADD COLUMN "State" character varying COLLATE pg_catalog."default",  
ADD COLUMN "Country" character varying COLLATE  
pg_catalog."default",  
ADD COLUMN "Complete_address" character varying COLLATE  
pg_catalog."default",  
ADD COLUMN point geometry,  
ADD COLUMN tpoint tgeompoint;
```

## 9 Algorithm 4: SQL schema to calculate monthly hours driven.

### Step 01: Consolidate Monthly Data

```
SELECT x.* INTO analytics_trips_time FROM (  
  Select 1 as month_id,vehicleReg, Sum(EXTRACT(epoch FROM  
  duration(trip_span))/3600) Hr from Analytics_trips_1 group by vehicleReg  
  union  
  Select 2 as month_id,vehicleReg, Sum(EXTRACT(epoch FROM  
  duration(trip_span))/3600) Hr from Analytics_trips_2 group by vehicleReg  
  union  
  Select 3 as month_id,vehicleReg, Sum(EXTRACT(epoch FROM  
  duration(trip_span))/3600) Hr from Analytics_trips_3 group by vehicleReg  
  union  
  Select 4 as month_id,vehicleReg, Sum(EXTRACT(epoch FROM  
  duration(trip_span))/3600) Hr from Analytics_trips_4 group by vehicleReg  
  union  
  Select 5 as month_id,vehicleReg, Sum(EXTRACT(epoch FROM  
  duration(trip_span))/3600) Hr from Analytics_trips_5 group by vehicleReg  
  union  
  Select 6 as month_id,vehicleReg, Sum(EXTRACT(epoch FROM  
  duration(trip_span))/3600) Hr from Analytics_trips_6 group by vehicleReg  
  union  
  Select 7 as month_id,vehicleReg, Sum(EXTRACT(epoch FROM  
  duration(trip_span))/3600) Hr from Analytics_trips_7 group by vehicleReg  
  union  
  Select 8 as month_id,vehicleReg, Sum(EXTRACT(epoch FROM  
  duration(trip_span))/3600) Hr from Analytics_trips_8 group by vehicleReg  
  union  
  Select 9 as month_id,vehicleReg, Sum(EXTRACT(epoch FROM  
  duration(trip_span))/3600) Hr from Analytics_trips_9 group by vehicleReg  
  union  
  Select 10 as month_id,vehicleReg, Sum(EXTRACT(epoch FROM  
  duration(trip_span))/3600) Hr from Analytics_trips_10 group by vehicleReg  
  union  
  Select 11 as month_id,vehicleReg, Sum(EXTRACT(epoch FROM  
  duration(trip_span))/3600) Hr from Analytics_trips_11 group by vehicleReg  
  union  
  Select 12 as month_id,vehicleReg, Sum(EXTRACT(epoch FROM  
  duration(trip_span))/3600) Hr from Analytics_trips_12 group by vehicleReg  
  ) x
```

### Step 02: Extract monthly hours

```
SELECT month_id,sum(hr)::numeric::integer as Hrs_Drive from  
analytics_trips_time group by month_id order by month_id
```









