

# Space & Time Databases

Independent Benchmark  
Q3 2022

# The Space & Time Database Benchmark

03

## Overview & Takeaways

Benchmark overview, database selection, data sets, and key takeaways

08

## Geospatial Analytics

Analysis and results of queries that integrate with geospatial functions

11

## Graph Analytics

Analysis and results of queries that integrate event data with graph functions

14

## Spatio-Temporal Analytics

Analysis and results of time-series queries integrated with event location data

17

## Streaming Data

Results of streaming IoT data ingestion and configuration

18

## Visualizing Data

Results of geospatial data visualizations generated in-database at scale

19

## Links and Resources

Links to data sets and SQL scripts used in the benchmark

20

## Appendix

Performance data results

# Benchmark Overview

With the proliferation of devices capable of transmitting their location, the amount of location-enriched IoT data has increased exponentially.

This machine data permeates our world – in consumer goods, health care, manufacturing, defense, transportation, energy, retail, and other industries. While IoT devices and use cases vary, they have two data components in common: *space and time*.

In this context, **Space** means that IoT data is increasingly tagged with a longitude and latitude or some other measure of location. **Time** refers to the data is created in given intervals that capture temporal changes.

Analyzing spatial and time series data presents new challenges to data analytics practitioners and vendors. Spatial and time series analysis requires non-traditional joins, filters, and specialized analytic functions. Despite the growth in data volumes and commercial interest, there was no framework in the marketplace to help organizations evaluate database technologies suitable for these workloads.

Radiant Advisors independently designed and performed the Space & Time Database Benchmark to accelerate knowledge in this area so that enterprises can more quickly and efficiently realize value from location-enriched real-time and historical IoT data.



## Benchmark Data

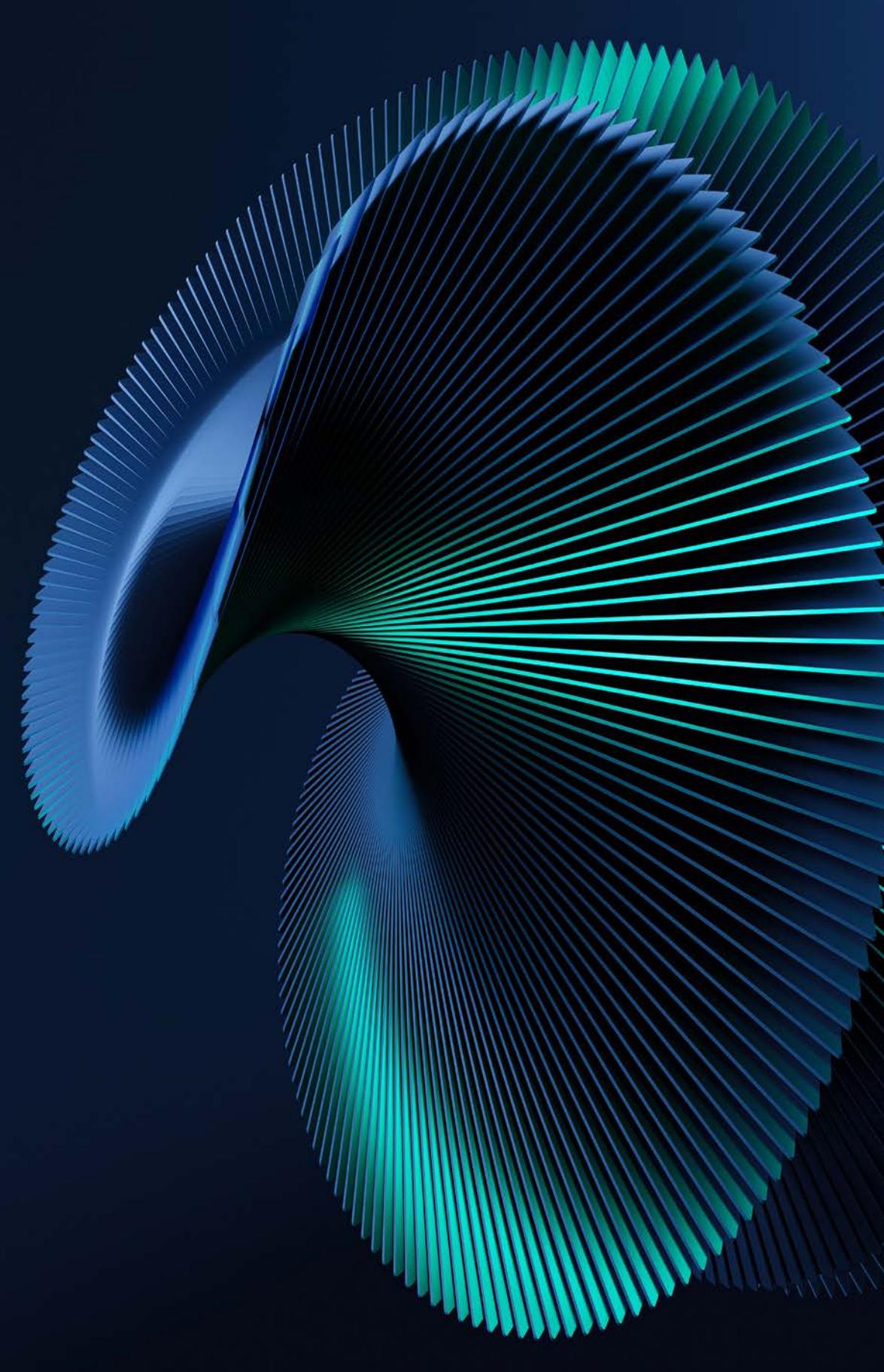
Citi Bike data from New York was chosen for its open public data and years of historical data for bike trips, station locations, and real-time API access to its bike stations. This is a good proxy for other use cases that track items through space and time such as logistics or routing.

Bike trips generate high volumes of daily event data with each bike being checked out and back in at different station locations throughout New York, each with timestamps. We loaded historical data for bike trips into sets of 10, 50, and 100 million records for benchmark testing.

start time	stop time	start station latitude	start station longitude	end station latitude	end station longitude
6/1/13 0:00	6/1/13 0:11	40.7423543	-73.98915076	40.74317449	-74.00366443
6/1/13 0:00	6/1/13 0:11	40.7423543	-73.98915076	40.74317449	-74.00366443
6/1/13 0:00	6/1/13 0:35	40.69512845	-73.99595065	40.69512845	-73.99595065
6/1/13 0:01	6/1/13 0:03	40.73524276	-73.98758561	40.6917823	-73.9737299
6/1/13 0:01	6/1/13 0:26	40.70569254	-74.01677685	40.68926942	-73.98912867
6/1/13 0:01	6/1/13 0:35	40.75038009	-73.98338988	40.69512845	-73.99595065
6/1/13 0:02	6/1/13 0:36	40.73454567	-73.99074142	40.7104512	-73.960876
6/1/13 0:03	6/1/13 0:09	40.7454973	-74.00197139	40.75096735	-73.99444208
6/1/13 0:03	6/1/13 0:34	40.72229346	-73.99147535	40.68216564	-73.95399026

*Sample of bike trips event data with timestamps and location-enriched station data.*

Location data for New York is also publicly available and used to define street geospatial locations, speed limits, and outlining areas and boroughs. Historical weather data was incorporated to add more depth to the analytics questions.



## Space & Time Database Selection

The following databases were selected initially based on having a SQL analytics for geospatial and temporal data.

[Kinetica](#) and [HEAVY.AI](#) (formerly OmniSci and MapD) are both focused on analyzing and visualizing geometric and time-series data types from real-time data streams. Even though both Kinetica and HEAVY.AI apply innovations from GPUs that have known computational advantages for spatial and time-series calculations, the benchmark only used their CPU versions in order to create a like-for-like comparison with other databases that are CPU only. HEAVY.AI declined to provide a trial distributed license for the benchmark. While this prevented benchmarking HEAVY.AI, functionality testing was conducted using HEAVY.AI's non-distributed trial license.

[Azure Cosmos DB](#) is Microsoft's proprietary globally distributed database service generally classified as a NoSQL database. Cosmos provides support for [geospatial](#) and [time-series](#) functions.

[PostGIS](#) - is an open-source software program that adds support for geographic objects to the PostgreSQL object-relational database. PostGIS is widely considered the standard for spatial analysis.

This is certainly not an exhaustive list of databases with support for spatial and time-series capabilities. Our hope is that others in the community will evaluate other databases and share the results. All data files and scripts are publicly available at <https://github.com/RadiantAdvisors/SpaceTimeBenchmark>

## Key Takeaways

1. Kinetica outperformed PostGIS in every query and was the only database to pass all feasibility tests across geospatial, time-series, graph, and streaming.
2. While some vendors have time series and geospatial capabilities, they cannot execute joins between them, forcing a significant restriction on use cases and incurring higher data engineering costs. (For e.g., Cosmos could not do simple joins between database containers and failed functional criteria.)
3. As IoT data which is inherently time series becomes geo-encoded, spatial-only or time-series-only databases will become less relevant when compared to multi-modal spatio-temporal databases.

## Research Notes

Ease of use (interface, connectors, collaboration) was a considerable part of the benchmark for our database engineers working with data and building, tuning, and validating SQL. Kinetica has a significant advantage for database developer and team efficiency.

PostGIS is not a distributed database and demonstrated its lack of linear scalability. To handle forecasted data growth and high-volume event data, such as IoT data, only distributed databases are capable of linear performance scalability.



*Spatial analysis allows you to solve complex location-oriented problems and better understand where and what is occurring in your world.*

*It goes beyond mere mapping to let you study the characteristics of places and the relationships between them. Spatial analysis lends new perspectives to your analysis and decision-making.*

# Geospatial Analytics – SQL Feasibility

**Business Goal:** Ensure that bikes are available in right areas and stations each day and determine how Operations can optimize the cost to redistribute bikes as needed.

Business Questions:

1. How many rides originated from an area?
2. How long were the rides that originated in an area?
3. How many rides originated and ended in an area?

SQL Functions	Kinetica	PostGIS	HEAVY.AI	Cosmos
1 Filter trips by Area	✓ STXY_INTERSECTS	✓ ST_INTERSECTS	✓ ST_INTERSECTS	X Unable to join two containers
2 Sum trips by Area	✓ TIMESTAMPDIFF STXY_INTERSECTS	✓ EXTRACT(EPOCH) ST_INTERSECTS	✓ TIMESTAMPDIFF ST_INTERSECTS	X Unable to join two containers
3 Count trips by Joined Areas	✓ STXY_INTERSECTS	✓ ST_INTERSECTS w/ ST_POINT	✓ CAST() ST_INTERSECTS() ST_POINT()	X Unable to join two containers

\* We do not consider Cosmos an appropriate database where there is a need to join spatial data sets.

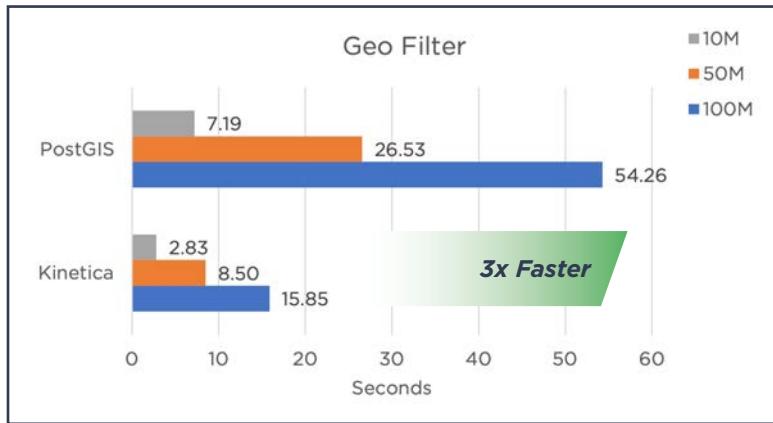
## Analysis and Notes

These business questions require integrating large amounts of bike trips start and stop data with the location attributes of the bike stations and the defined areas of various New York City boroughs.

SQL statements require functions that can filter and join based on geospatial data as a component of analytics for counting and aggregating high volumes of bike trips relational data.

Spatial INTERSECTS functions and timestamp conversions are available in each database with slight variations. However, surprisingly the multi-modal Cosmos database cannot perform joins across its containers of different database types to perform the necessary SQL joins in our benchmark. Items within Cosmos container are required to be homogeneous to the container's database type.

# Geospatial Analytics - Performance Benchmark Results



How many rides originated from an area?



How long were the rides that originated in an area?



How many rides originated and ended in an area?

*Various indexes, spatial indexes, and database tuning parameters were tested but did not improve PostGIS performance.*

Tested: Kinetica Large: Four ESDv2 with 2TB premium SSD; PostGIS: Azure VM: Single M192 with 2TB premium SSD.

HEAVY.AI declined to provide a trial distributed license for performance benchmark.

We do not consider Cosmos an appropriate database where there is a need to join spatial data sets.

All data files and scripts are publicly available at <https://github.com/RadiantAdvisors/SpaceTimeBenchmark>

## Analysis and Notes

The Geo Filter queries demonstrated a linear performance for both PostGIS and Kinetica databases with increasing amounts of bike trips.

Results: Kinetica consistently performs 2.5x - 3.5x faster than PostGIS.

The Geo Join query was completed by both databases, however PostGIS took significantly longer in response time. Various indexes were tested, including spatial indexes, that did not improve performance.

Distributed database architectures, like Kinetica, will deliver linear query performance when scaling data compared to symmetrical multi-processing database architectures, like Postgres, with an equivalent amount CPUs and memory. This is key for managing query performance as data volumes increase.



*Graph analysis is a key component for geospatial analytics as it efficiently models the surrounding environment as interconnected entities to explore the relationship between absolute and relative position of the entities.*

# Graph Analytics - SQL Feasibility

**Business Goal:** The business needs to perform trip analytics based on areas, boundaries, roads, and speed limits while customers may seek shortest path routes.

Business Questions:

1. Find all bike trips originating and ending in an area.
2. Find the shortest route from Empire State Building to the Brooklyn Bridge.
3. What was the shortest path taken of bike trips on 12/20/20?
4. Find the total length of shortest paths by all bikes on 8/30/21?

SQL Feasibility	Kinetica	PostGIS	HEAVY.AI	Cosmos
1 Graph Create	✓ CREATE GRAPH	✓ TOPOLOGY	✗ Not supported	✓ Create Graph
2 Shortest Path	✓ GRAPH_SOLVE	✓ PGR_DIJKSTRA	✗ Not supported	○ API for Gremlin
3 Graph Batch Solve	✓ GRAPH_SOLVE w/ sub query	✓ PGR_DIJKSTRA	✗ Graph cannot be incorporated	✗ Unable to join containers
4 Graph Solve & Aggregate	✓ GRAPH_SOLVE SHORTEST_PATH	✓ CREATE TABLE PATH multiplier	✗ Graph cannot be incorporated	✗ Unable to join containers

## Analysis and Notes

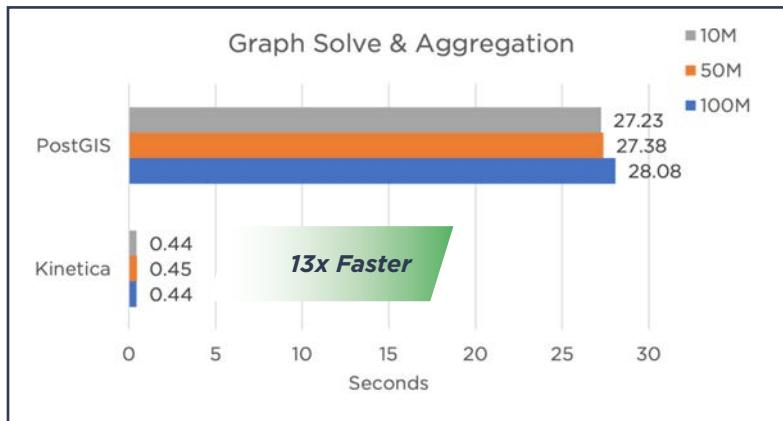
These business questions require integrating the large amounts of bike trips data by date ranges for their start and stop station locations with data loaded into graph databases to define areas such as the boroughs of New York City.

Routing routines such as “shortest path” and “Dijkstra” can inform a customer the best way to get anywhere from their current location on a mobile app.

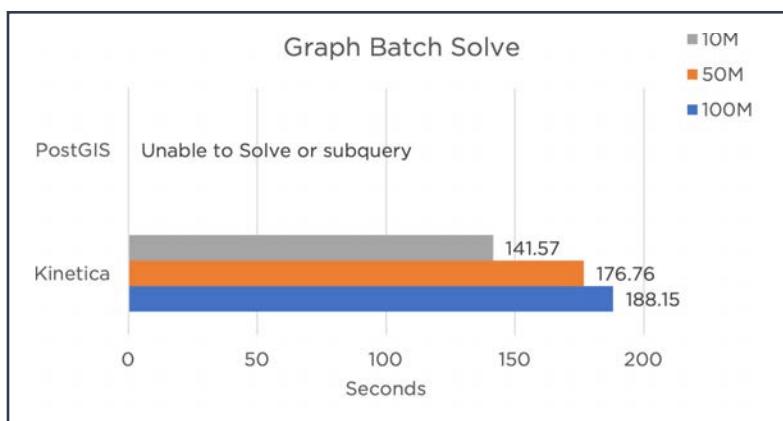
Creating a topology and adding GIS extensions to Postgres was straightforward and quick. HEAVY.AI does not have a native graph database like Kinetica for solving analytics. The Cosmos DB container for graph databases can solve a shortest path with the Gremlin APIs but couldn’t join to bike trip data in another Cosmos container.

# Graph Analytics – Performance Benchmark Results

Graph databases can be used to define an area (polygon) or roads to solve for optimal geospatial routes. However, integrating graph data with large event data, such as bike trips, can be challenging. *PostGIS and HEAVY.AI do not have a native graph.*



Find all bike trips originating and ending in an area



What was the shortest path taken of bike trips on 12/20/20?

Tested: Kinetica Large: Four ESDv2 with 2TB premium SSD; PostGIS: Azure VM: Single M192 with 2TB premium SSD. HEAVY.AI declined to provide a trial distributed license for performance benchmark.

All data files and scripts are publicly available at <https://github.com/RadiantAdvisors/SpaceTimeBenchmark>

©2022 Radiant Advisors All Rights Reserved.

## Analysis and Notes

Graph databases easily demonstrate their consistently great performance with little variation at 5x and 10x scale of bike trips data.

Results: Kinetica was 13x faster than PostGIS when filtering the bike trips data by the polygon area.

However, as the business analytics got more complex, PostGIS and HEAVY.AI does not have a native graph database that can solve all bike trips paths taken on a single date to determine which was likely the shortest path taken. This straightforward SQL statement requires a sub-query which PostGIS cannot support.

Likewise, PostGIS and HEAVY.AI are unable to solve a native graph to determine the shortest path of all bike trips taken before aggregating those lengths or sorting for the shortest or longest of all paths taken for calculating revenue and cost as total ride time.



*Temporal analysis, also known as time series analysis, compares the results of the same metric for different time periods, or temporal snapshots of the spatial data.*

# Temporal Analytics – SQL Feasibility

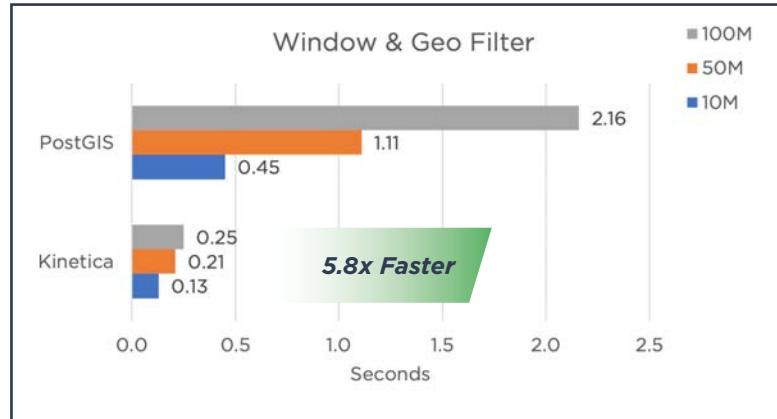
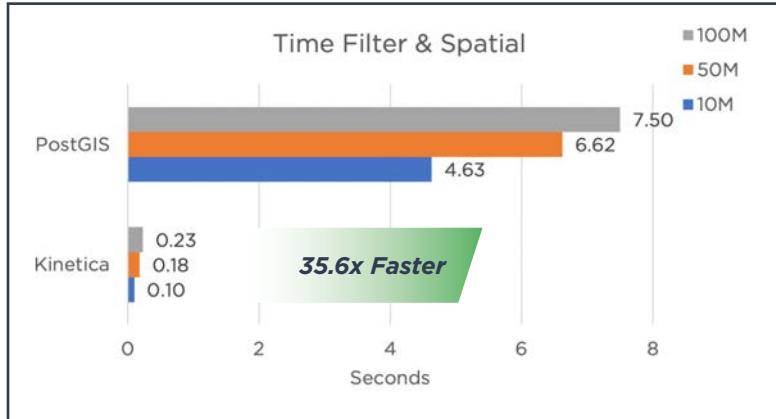
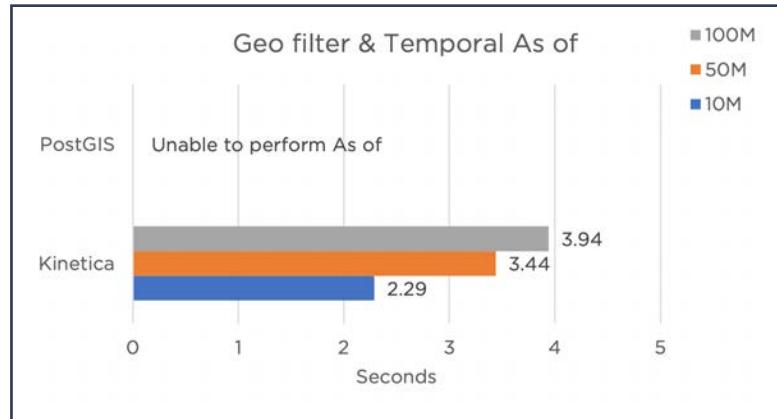
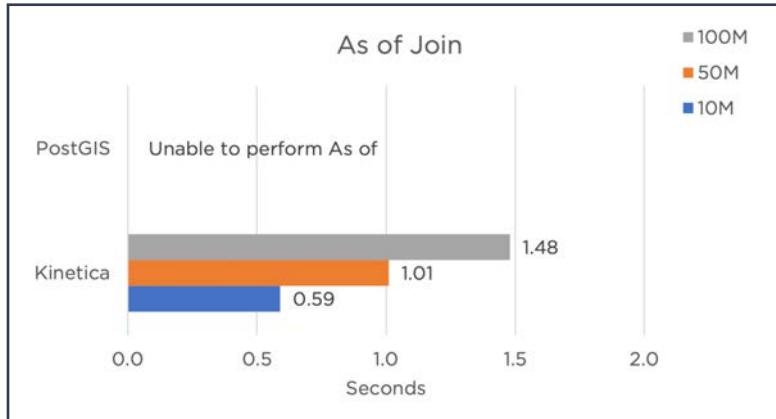
Spatio-Temporal SQL statement were straightforward except for the “as of” function which only Kinetica could perform. As a workaround, temporary tables of temperature data were created to join to in PostGIS and HEAVY.AI but would take between two to six hours to refresh and therefore was not feasible.

SQL Feasibility	Kinetica	PostGIS	HEAVY.AI
1 AS OF Join	✓ LEFT JOIN ON ASOF(1)	✗ Times table created but refresh too long	✗ Times table created but refresh too long
2 Time Filter & Spatial	✓ JOIN ON STXY_INTERSECTS w/ ST_DISSOLVE	✓ JOIN ON ST_INTERSECTS w/ ST_UNION	JOIN ON ST_INTERSECTS w/ ST_UNION v5.2
3 Moving Window Function	✓ RANGE BETWEEN	✓ EXTRACT(EPOCH) W/ RANGE BETWEEN	○ CAST(EXTRACT()) used in an equation
4 Geo filter & Temporal ASOF	✓ LEFT JOIN ON ASOF STXY_INTERSECTS JOIN w/ polygon	✓ Times table created but refresh too long	✗ Times table created but refresh too long
5 Geo filter & Window	✓ RANGE BETWEEN w/ STXY_INTERSECTS	✓ EXTRACT(EPOCH) w/ RANGE BETWEEN w/ ST_INTERSECTS	✓ EXTRACT(EPOCH) w/ RANGE BETWEEN w/ ST_INTERSECTS
6 Geo join & Temporal Filter	✓ JOIN w/ STXY_INTERSECTS	✓ JOIN w/ ST_INTERSECTS	✓ JOIN w/ ST_INTERSECTS

## Business Questions

- Join the bike events to the temp readings based on “as of” time and interval +/- 1 hour.
- Count all events that occurred between two given date times that intersect an area polygon
- For every ride, find the moving demand over 10-minute sliding window
- Get the temperature within an hour of the ride starting in the given area polygon. Count the number of trips that started in the area polygon where the temp was above 60 degrees
- Find the demand of trips for a particular day that fell within the GIANT\_POLYGON
- Join events to boundaries where the trip duration was less than 10 minutes

# Temporal Analytics - Performance Benchmark Results



Tested: Kinetica Large: Four ESDv2 with 2TB premium SSD; PostGIS: Azure VM: Single M192 with 2TB premium SSD.

HEAVY.AI declined to provide a trial distributed license for performance benchmark.

All data files and scripts are publicly available at <https://github.com/RadiantAdvisors/SpaceTimeBenchmark>

## Analysis and Notes

The “as of” time function was only possible with Kinetica, and our database engineers attempted several work arounds such as joining to derived tables. However, creating and updating these tables took hours to complete and wouldn’t be valuable.

**Results: Kinetica consistently performs 5x - 42x faster than PostGIS.**

Important to note is the faster and predictable linear performance of Kinetica at each data volume which is a characteristic of shared-nothing architectures.

Despite both databases having 192 CPUs available, Kinetica performed notably faster at all data volumes while PostGIS demonstrated non-linear performance despite tuning. HEAVY.AI is expected to have similar distributed shared-nothing performance but could not be verified without their enterprise license.



*The ability to ingest streaming IoT sensor data into databases is critical for near-real time analytics that can drive immediate decisions and actions for optimized operations and customer satisfaction.*

# Streaming Data Ingestion Feasibility



**Business Goal:** Need real-time updates from bike stations for business operations, alert notifications, and improved customer experience from bike availability.

Business Questions:

1. Configure streaming data ingestion of Citi Bike JSON data.
2. Utilize linear regression or machine learning (ML) to send real-time notifications bike shortages at stations.

	SQL Functions	Kinetica	PostGIS	HEAVY.AI	Cosmos
1	Consume data from Confluent Kafka on Azure	✓ CREATE TABLE as Kafka Consumer	○ Could not make Confluent Connector work w/ JSON & Postgres	○ Could not get Kafka consumer program to insert JSON records	Not tested
2	Linear Regression/ML	✓ Materialized views auto refreshes	○ Has LR/ML but materialized views need manual refresh or procedure	✗ LR/ML not supported	Not tested

## Analysis and Notes

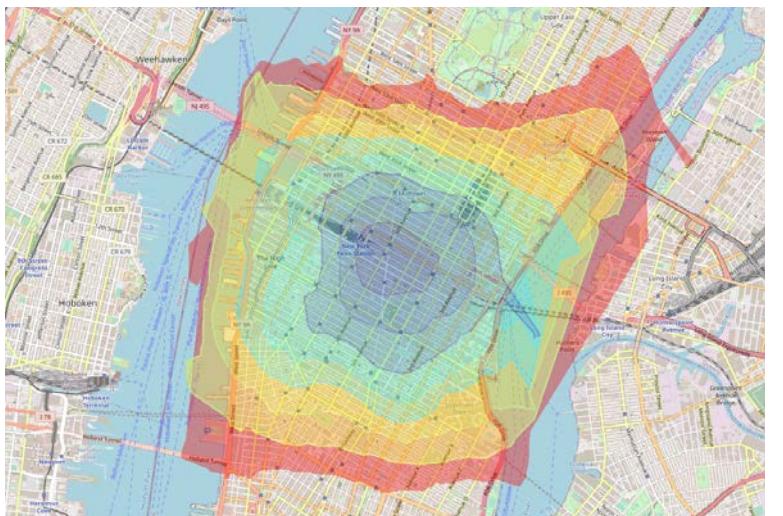
A Python Producer program was used to fetch real-time Citi Bike station data by calling their REST API then publishing the JSON data to a Confluent Cloud on Azure topic. Each database could subscribe and load this data for near real-time analytics.

Kinetica took minutes to configure with a table defined with Kafka consumer properties and security.

Postgres documentation recommends using the Kafka Connector which after two weeks of Confluent Support could not be successfully set up. Without a HEAVY.AI Kafka Connector available, another Python Consumer program had to be written and deployed to read from the Kafka topic and insert records into the database.

For real-time warning notifications of low inventory, a materialized view with linear regression or machine learning was needed.

# Streaming Data and Visualization



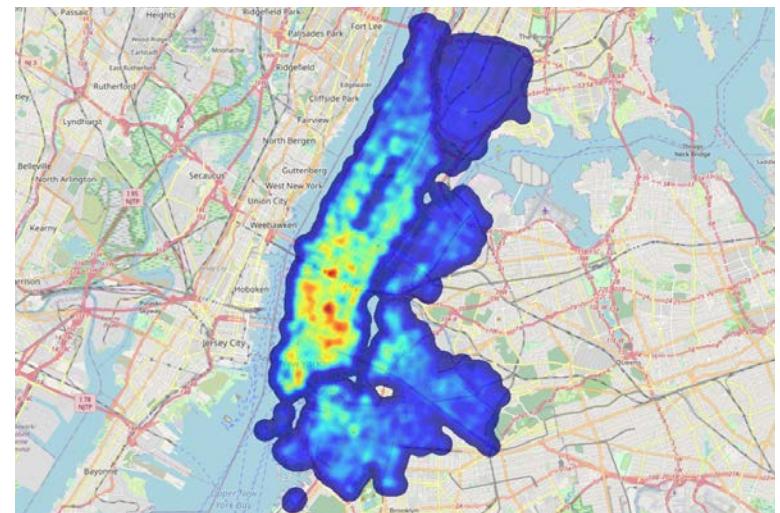
0.9 seconds to execute at 10M, 50M, 100M

Kinetica generated *in-database* isochrone visualizations under one second leveraging the graph data for a point, roads, and speed limits. A REST API delivers the PNG.

Left is a 6-level isochrone of five-minute distances from the Empire State Building.

Kinetica generated *in-database* heatmap visualizations of historical bike trips data under one second.

This avoids sending tens of millions rows of data to any data visualization tool.



HEAVY.AI declined to support performance testing with a trial distributed enterprise license.

## Analysis and Notes

The greater the amount of data, the better the resolution of data visualizations can be. However, sending tens or hundreds of millions of rows of data to can be time consuming and costly, not to mention impacting the user experience in analysis.

We created these spatio-temporal data visualizations in Kinetica to leverage its computing resources and avoid data transfer time and cost.

Creating and refreshing the visualizations can be scheduled in the database with a corresponding REST API that allow users and applications to request the PNG file many times without having to render the image over and over.

In-database visualization is critical to high volume data analysis in near real-time.



Radiant Advisors independently designed and performed the Space and Time Database Benchmark working with a database engineering team from [MResult](#).

[Kinetica](#) sponsored this independent research and Azure resources.

[Confluent Cloud](#) support for Azure was utilized for streaming data testing.

The [PostGIS](#) Community provided online support and performance tuning.

[HEAVY.AI](#) declined to support or provide a trial enterprise license necessary for multi-node performance testing.

Images from [Unsplash](#).

All test data and scripts are publicly available in the [Radiant Advisors GitHub repository](#).

# Appendix: Space and Time Database Benchmark Results



Ref	Benchmark Test	10M bike trips			50M bike trips (5x data)			100M bike trips (10x data)		
		Kinetica	PostGIS	Factor	Kinetica	PostGIS	Factor	Kinetica	PostGIS	Factor
1	Geo Filter	2.83	7.19	2.54	8.50	26.53	3.12	15.85	54.26	3.42
2	Geo Filter & Sum	2.83	7.29	2.58	8.50	26.78	3.15	16.02	54.13	3.38
3	Geo Join & Count	8.84	3,779.03	427.49	30.39	11,113.07	365.68	64.04	17,285.01	269.91
4	Graph creation	99.27	1,237.67	12.47	96.00	1,237.67	12.89	94.10	1,237.67	13.15
5	Graph Solve & Aggregation	0.44	27.23	61.89	0.45	27.38	60.84	0.44	28.08	63.82
6	Graph Batch Solve	141.57	X	can't solve graph or subquery	176.76	X	can't solve graph or subquery	188.15	X	can't solve graph or subquery
7	Graph Solve & Aggregation	0.37	X	import Kinetica graph & index	0.75	X	import Kinetica graph & index	1.37	X	import Kinetica graph & index
8	AS OF Join	0.59	X	not feasible w/ temp table	1.01	X	not feasible w/ temp table	1.48	X	not feasible w/ temp table
9	Time Filter & Spatial	0.10	4.63	46.30	0.18	6.62	36.78	0.23	7.50	32.61
10	Moving Window Function	0.10	0.45	4.50	0.15	1.10	7.33	0.16	2.23	13.94
11	Geo filter & Temporal ASOF	2.29	X	not feasible w/ temp table	3.44	X	not feasible w/ temp table	3.94	X	not feasible w/ temp table
12	Window & Geo Filter	0.13	0.45	3.46	0.21	1.11	5.29	0.25	2.16	8.64
13	Geo-join & Temporal Filter	0.14	4.76	34.00	0.23	13.62	59.22	0.27	9.28	34.37
14	Generate Isochrone	0.92	X	not possible	0.96	X	not possible	0.90	X	not possible

HEAVY.AI declined to support performance testing with a trial distributed enterprise license.