The Most Scalable Vector Database for AI

JaguarDB Cloud Administrator and User Manual

捷豹向量数据库云管理和开发手册

Administrador en la Nube de JaguarDB y Manual del Usuario

JaguarDB クラウド管理者およびユーザーマニュアル

जैगुआरडीबी क्लाउड प्रशासक और उपयोगकर्ता मैनुअल

Administrateur de Cloud JaguarDB et Manuel de l'Utilisateur

JaguarDB Cloud-Administrator und Benutzerhandbuch

(Release 3.4.3 02/04/2025)

Contents

Introduction

JaguarDB is a vector database equipped with its distinctive ZeroMove hashing technology, enabling seamless scalability to handle extensive volumes of vector data and media content. Artificial intelligence (AI) often relies on vector databases for various tasks such as natural language processing, information retrieval, recommendation systems, and similarity matching. The use of vector databases is particularly relevant in the context of machine learning models that leverage embeddings, which are numerical representations of data elements in a continuous vector space.

Vectors provide a compact and efficient representation of complex data structures. By transforming data elements into vectors with thousands of dimensions, AI systems can work with numerical representations that are more amenable to mathematical operations and analysis. Vector databases enable the computation of similarity or distance metrics between vectors, such as cosine similarity or Euclidean distance. These metrics are fundamental for tasks like similarity matching, nearest neighbor searches, and clustering, which are essential in recommendation systems, content retrieval, and data exploration.

In the fields of generative AI, the exponential growth of data is inevistore. From voluminous vector data to vast collections of photos and videos, the potential for information generation knows no bounds. However, efficiently managing this diverse and ever-expanding data landscape poses a significant challenge for traditional database and storage systems. AI-generated data can quickly accumulate and consume significant storage space. Storing and managing this massive amount of data requires robust and scalable infrastructure. Organizations need to invest in adequate storage solutions, such as cloud storage or distributed file systems, to accommodate the growing data volumes.

Traditional databases rely on consistent hashing techniques, which, unfortunately, lead to excessive data migration. During the constant expansion of data systems, incremental scaling operations often require data migration for almost every piece of data and impose substantial costs on the system. These costs manifest in various forms,

including increased power consumption, hardware wear and tear, and degraded performance.

The innovative ZeroMove technique is employed in JaguarDB that offers a revolutionary solution. In contrast to the consistent hashing algorithm, which requires data migration when scaling out the system, ZeroMove enables scaling without the need to move data between computers. Data is intelligently tagged with encoded identifiers to facilitate efficient host location. These encoded identifiers serve as unique markers that enable swift and accurate retrieval of data within the system. Our approach ensures that data remains in the host where it is hashed, thereby increasing availability, and improving system performance.
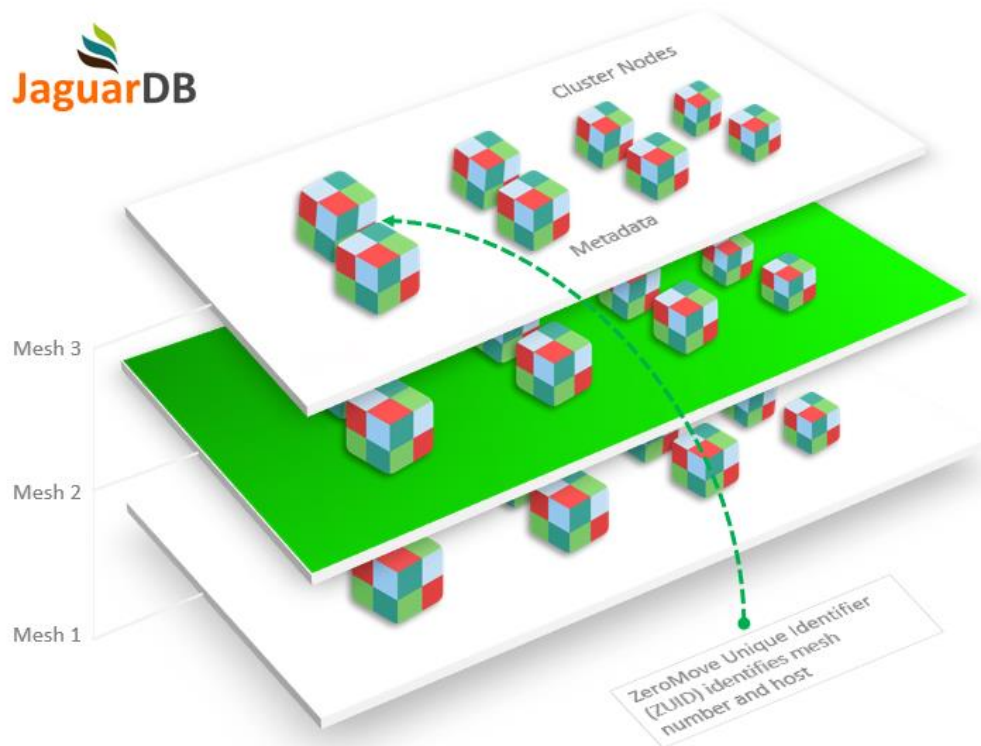
An AI datalake is equally crucial for AI applications, as media data like images and videos tend to occupy more space compared to structured data. The ZeroMove technology is particularly potent when it comes to efficiently scaling AI data systems. Geospatial search plays a significant role in enhancing the capabilities of AI, especially in robotic applications. Self-driving cars, drones, and robotics heavily rely on geospatial

data for navigation and obstacle avoidance. AI models can forecast future trends and outcomes by analyzing historical time series data. This is vital for financial predictions, stock market analysis, and demand forecasting.

JaguarDB is a pioneering platform that seamlessly brings together vector data, time series, and location data into a unified ecosystem. With JaguarDB, users can effortlessly manage and leverage vector data and diverse data types in a single, comprehensive solution. This convergence in JaguarDB eliminates the need for multiple disjointed systems. This streamlined approach not only simplifies data management but also proves cost-effective, particularly for large-scale AI data systems.

Programing Jaguar Vector Database

Guides and Concepts

Pods

A Pod serves as a versatile repository designed to house and manage multiple stores within an organization. This structural unit not only functions as a robust database but also excels in promoting streamlined and effective data management. Its primary role lies in facilitating the organized and efficient handling of data, ensuring clear data structures and swift data accessibility.

Stores

A store in JaguarDB bears a resemblance to the stores found in the relational databases, but with the added capability of accommodating multiple vector indexes. This feature grants it with versatility, allowing users to store not just scalar data, but also vector data and raw binary data, making it a multi-purpose storage solution. Multiple stores can exist in a pod.

Vector Indexes

A vector index is a data structure or technique used to efficiently store and retrieve high-dimensional vectors. The Hierarchical Navigable Small World (HNSW) is a type of data structure and algorithm used for efficient approximate nearest neighbor search

in high-dimensional spaces. It is integrated and extended in JaguarDB to address the challenges of performing nearest neighbor searches in spaces with a large number of dimensions, where traditional search methods become less effective.

Scalar Indexes

A scalar is a store column in database to represent a data attribute. A scalar index is a data structure that improves the speed of scalar data retrieval operations on a database store. It is an integral part of JaguarDB and plays a crucial role in optimizing query performance, on both vector search and scalar search. Scalar indexes together with vector indexes can quickly locate and access data records that satisfy a specific condition.
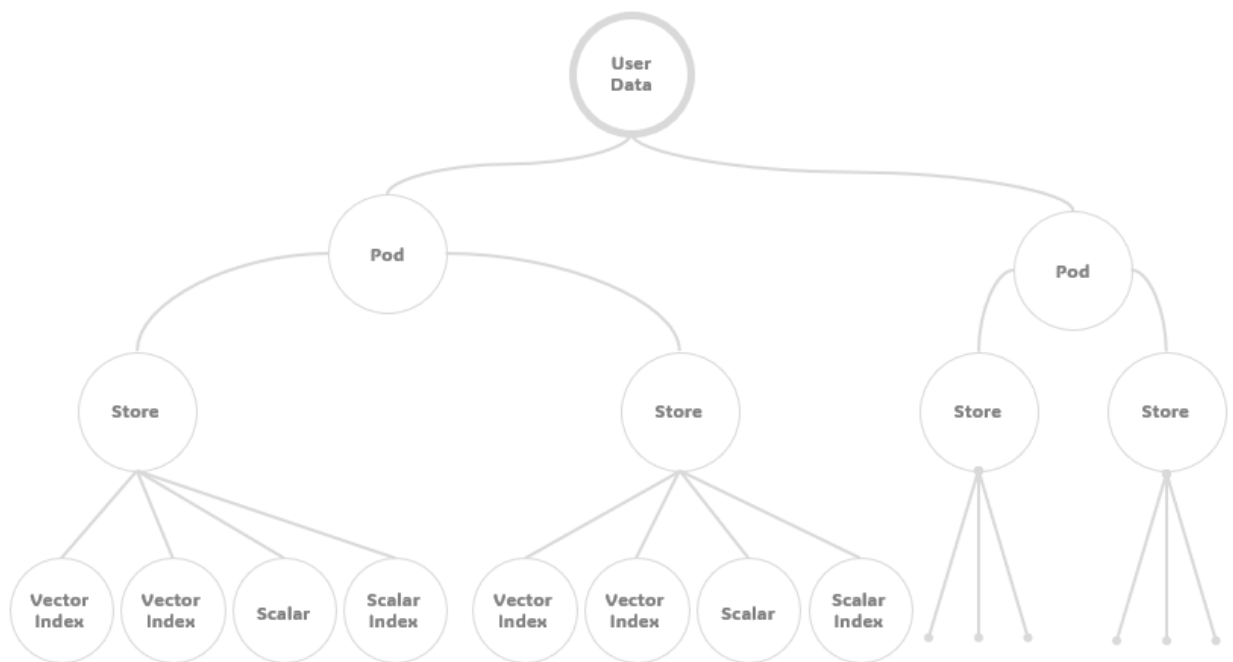
Python Example of Full Text Vector Search

The following Python example illustrates the integration of JaguarDB into AI applications for the benefit of software engineers and data scientists. In this demonstration, the focus lies on the seamless storage of textual data, the creation of embeddings, and the execution of similarity searches within the text data corpus. The process entails identifying texts that closely correspond to a given query text. Notably,

this operation is solely reliant on vector embeddings, rendering the inclusion of explicit keywords or search cues unnecessary.

The following Python code connects to JaguarDB instance:

```
jag = jaguarpy.Jaguar()

host = "127.0.0.1"
port = sys.argv[1]
user = "admin"
apikey = "myapikey"
vectordb = "vdb"

rc = jag.connect( host, port, apikey, vectordb )
print ("Connected to JaguarDB server" )
```

Next, a store containing vector column and other related data is created:

```
jag.execute("create store textvec ( key: zid zuid, value: v vector(1024,
'cosine_fraction_short'), text char(2048), source char(32) )")
```

In this statement, the "zid" field stands as an automatically generated unique identifier. The "v" field represents a vector, comprising two primary elements: an integer vector ID and an array of vector components. Notably, the dimension of the vector is set at 1024. The inclusion of "cosine" within the string "cosine_fraction_short" signifies the intention to employ the cosine distance metric for similarity searches conducted on the vector. The term "fraction" alludes to the anticipated fractional-format input data. It's worth noting that JaguarDB vector storage implements distinct quantization levels. Specifically, the short quantization mode leverages 16-bit quantization techniques to efficiently store vector data. Ther is no limit on the number of vectors in a store. Multiple vectors can be created on the same store, to capture various types of vectors for the same object. The "text" field can store text data for an object, with a maximum capacity of 2048 bytes.

When a store is created with a vector column, a scalar index is automatically created to link the vector ID and the zeromove unique id. The name of the scalar index is "DB.store.veccol_zid_idx", where DB is the name of the database or pod, store is the name of the store or store, and "veccol" is the name of vector column. User can create other scalar indexes based on specific requirements.

With JaguarDB, users can store various types of vectors, such as feature vectors and embedding vectors. An embedding vector, often simply referred to as an "embedding," is a mathematical representation of a discrete item, such as a word, phrase, image, or any other entity, in a continuous vector space. This technique is commonly used in various fields, including natural language processing (NLP), computer vision, recommendation systems, and more. The primary idea behind embedding vectors is to capture semantic relationships between items by placing similar items closer together in the vector space.

In this example, we use the "BAAI/bge-large-en" pre-trained embedding model to generate embeddings for the text data. A pre-trained embedding model is a machine learning model that has been trained on a large dataset to create meaningful representations (embeddings) of items in a continuous vector space. These embeddings capture semantic relationships and contextual information about the items. Pre-training involves training the model on a specific task, such as language modeling or image classification, with the goal of learning general features and patterns from the data. These learned features can then be fine-tuned or used as-is for various downstream tasks. Pre-trained embedding models are especially popular in natural language processing (NLP) and computer vision. The model "BAA/bge-large-en" requires a dimension of 1024 on the vectors, which was specified in the statement when we created the store and the vector field.

```
model = SentenceTransformer('BAAI/bge-large-en')
```

There are some simple required steps to setup and use the model. They are described in the github project github.com/fserv/jaguardb, in embedding ➜ text ➜ baai-bge-large ➜ README.md.

Next, we store a group of text data in the vector database store:

```
text = "Human impact on the environment (or anthropogenic environmental impact) refers
to changes to biophysical environments and to ec
osystems, biodiversity, and natural resources caused directly or indirectly by
humans."
    zuid1 = storeText( jag, model, text, "wiki" )

    text = "a group of people involved in persistent interpersonal relationships, or a
large social grouping sharing the same geographical or social territory, typically
subject to the same political authority and dominant cultural expectations. Human
societies are characterized by patterns of relationships (social relations) between
```

individuals who share a distinctive culture and institutions; a given society maybe described as the total of such relationships among its constituent members."
    zuid2 = storeText(jag, model, text, "wiki")

    text = "In 1768, Astley, a skilled equestrian, began performing exhibitions of trick horse riding in an open field called Ha'Penny Hatch on the south side of the Thames River, England. In 1770, he hired acrobats, tightrope walkers, jugglers and a clown to fill in the pauses between the equestrian demonstrations and thus chanced on the format which was later named a circus. Performances developed significantly
 over the next fifty years, with large-scale theatrical battle reenactments becoming a significant feature. "
    zuid3 = storeText(jag, model, text, "wiki")

    text = "Astley had a genius for trick riding. He saw that trick riders received the most attention from the crowds in Islington. He had an idea for opening a riding school in London in which he could also conduct shows of acrobatic riding skill. In 1768, Astley performed in an open field in what is now the Waterloo area of London, behind the present site of St John's Church. Astley added a clown to his shows t o amuse the spectators between equestrian sequences, moving to fenced premises just south of Westminster Bridge, where he opened his riding school from 1769 onwards and expanded the content of his shows. He taught riding in the mornings and performed his feats of horsemanship in the afternoons."
    zuid4 = storeText(jag, model, text, "google")

    text = "After the Amphitheatre was rebuilt again after the third fire, it was said to be very grand. The external walls were 148 feet long which was larger than anything else at the time in London. The interior of the Amphitheatre was designed with a proscenium stage surrounded by boxes and galleries for spectators. The general structure of the interior was octagonal. The pit used for the entertainers and ri ders became a standardised 43 feet in diameter, with the circular enclosure surrounded by a painted four foot barrier. Astley's original circus was 62 ft (~19 m) in diameter, and later he settled it at 42 ft (~13 m), which has been an international standard for circuses since."
    zuid5 = storeText(jag, model, text, "wiki")

    text = "According to the Big Bang theory, the energy and matter initially present have become less dense as the universe expanded. Afte
r an initial accelerated expansion called the inflationary epoch at around 10−32 seconds, and the separation of the four known fundamental forces, the universe gradually cooled and continued to expand, allowing the first subatomic particles and simple atoms to form. Dark matter gradually gathered, forming a foam-like structure of filaments and voids under the influence of gravity. Giant clouds of hydrogen and heli um were gradually drawn to the places where dark matter was most dense, forming the first galaxies, stars, and everything else seen today."
    zuid6 = storeText(jag, model, text, "wiki")

    text = "By comparison, general relativity did not appear to be as useful, beyond making minor corrections to predictions of Newtonian gravitation theory. It seemed to offer little potential for experimental test, as most of its assertions were on an astronomical scale. Its mathematics seemed difficult and fully understandable only by a small number of people. Around 1960, general relativity became central to p hysics and astronomy. New mathematical techniques to apply to general relativity streamlined calculations and made its concepts more easily visualized. As astronomical phenomena were discovered, such as quasars (1963), the 3-kelvin microwave background radiation (1965), pulsars (1967), and the first black hole candidates (1981), the theory explained their attributes, and measurement of them further confirmed the t heory."
    zuid7 = storeText(jag, model, text, "wiki")

    text = "In astronomy, the magnitude of a gravitational redshift is often expressed as the velocity that would create an equivalent shift through the relativistic Doppler effect. In such units, the 2 ppm sunlight redshift corresponds to a 633 m/s receding

velocity, roughly of the same magnitude as convective motions in the sun, thus complicating the measurement. The GPS satellite gravitational blueshift velo city equivalent is less than 0.2 m/s, which is negligible compared to the actual Doppler shift resulting from its orbital velocity."
```
    zuid8 = storeText( jag, model, text, "google" )
```

text = "Turn on the sprinkler system. In order to locate the break or leak in the sprinkler system, you need to run water through it. Turn on the sprinkler system to activate the flow of water. Allow the water to run for about 2 minutes before you check the lines. Do this in the daytime, when you'll have an easier time spotting the leak. If your sprinkler system is separated into zones, activate the zones one at a time so you can identify the break or leak more easily."
```
    zuid9 = storeText( jag, model, text, "wiki" )
```

text = "Check for water bubbling up from the soil. If you see a pool of water or water coming from the soil, then there's a leak in the sprinkler line buried underneath. Mark the general location of the leak or break so you can identify it when the water is turned off. Place an item like a shovel or a rock on the ground near the leak. Turn off the sprinkler system after you've found the leak. If you've found the signs of a leak and located the region where the line is leaking or broken, turn off the water so you can repair the line. Use the shut-off valve in the control box to stop the flow of water through the system."
```
    zuid10 = storeText( jag, model, text, "wiki" )
```

text = "In fact, Antarctica is such a good spot for meteorite hunters that crews of scientists visit every year, searching for these otherworldly rocks, driving around the surface until they spot a lone dark rock on an otherwise unbroken expanse of white. However, you don't always have to travel to the other side of the world to find a meteorite. Sometimes meteorites will come to you. Keep an eye open for local reports of brilliant fireballs lighting your region's sky. Debris from such displays scatters across the ground and sometimes hits structures or vehicles. Watch for information about fireballs in your area on the websites of the American Meteor Society or the International Meteor Organization."
```
    zuid11 = storeText( jag, model, text, "wiki" )
```

text = "Most tornadoes are found in the Great Plains of the central United States – an ideal environment for the formation of severe thunderstorms. In this area, known as Tornado Alley, storms are caused when dry cold air moving south from Canada meets warm moist air traveling north from the Gulf of Mexico. Tornadoes can form at any time of year, but most occur in the spring and summer months along with thunderstorms. May and June are usually the peak months for tornadoes. The Great Plains are conducive to the type of thunderstorms (supercells) that spawn tornadoes. It is in this region that cool, dry air in the upper levels of the atmosphere caps warm, humid surface air. This situation leads to a very unsstore atmosphere and the development of severe thunderstorms."
```
    zuid12 = storeText( jag, model, text, "google" )
```

The function storeTex is implemented with the following program:

```
def storeText(jag, model, text, src):
    sentences = [ text ]
    embeddings = model.encode(sentences, normalize_embeddings=False)
    vstr = ",".join( [str(x) for x in embeddings[0] ])

    istr = "insert into textvec values ('" + vstr + "','" + text + "','"+src+"')"
    jag.execute( istr )
    return jag.getLastZuid()
```

Now we can use a query and get similar texts from database:

```
query_text = "More recently, that focus has shifted eastward by 400 to 500 miles. In
the past decade or so tornadoes have become prevalent in eastern Missouri and
Arkansas, western Tennessee and Kentucky, and northern Mississippi and Alabama—a new
region of concentrated storms. Tornado activity in early 2023 epitomized the trend."

    K = 3;

    retrieveTopK( jag, model, query_text, K )
```

The result is show below:

```
Retrieved similar texts:


Rank: 1

Vector ID: 1168196459548885000

Distance: 83010886

Text: Most tornadoes are found in the Great Plains of the central United States – an
ideal environment for the formation of severe thunderstorms. In this area, known as
Tornado Alley, storms are caused when dry cold air moving south from Canada meets warm
moist air traveling north from the Gulf of Mexico. Tornadoes can form at any time of
year, but most occur in the spring and summer months along with thunderstorms.
May and June are usually the peak months for tornadoes. The Great Plains are conducive
to the type of thunderstorms (supercells) that spawn tornadoes. It is in this region
that cool, dry air in the upper levels of the atmosphere caps warm, humid surface air.
This situation leads to a very unsstore atmosphere and the development of severe
thunderstorms.




Rank: 2

Vector ID: 1168196459122135000

Distance: 132491843

Text: In fact, Antarctica is such a good spot for meteorite hunters that crews of
scientists visit every year, searching for these otherworldly rocks, driving around
the surface until they spot a lone dark rock on an otherwise unbroken expanse of
white. However, you don't always have to travel to the other side of the world to find
a meteorite. Sometimes meteorites will come to you. Keep an eye open for local reports
of brilliant fireballs lighting your region's sky. Debris from such displays scatters
across the ground and sometimes hits structures or vehicles. Watch for information
about fireballs in your area on the websites of the American Meteor Society or the
International Meteor Organization.




Rank: 3

Vector ID: 1168196458028280000
```

```
Distance: 145542249
```

Text: In astronomy, the magnitude of a gravitational redshift is often expressed as the velocity that would create an equivalent shift through the relativistic Doppler effect. In such units, the 2 ppm sunlight redshift corresponds to a 633 m/s receding velocity, roughly of the same magnitude as convective motions in the sun, thus complicating the measurement. The GPS satellite gravitational blueshift velocity equivalent is less than 0.2 m/s, which is negligible compared to the actual Doppler shift resulting from its orbital velocity.

Then we can have another query and get similar texts from database:

```
    query_text = "Think of designing a landscape for the bare lot surrounding your new
home as an adventure in creativity. Perhaps your property needs only a few small,
easily doable projects to make it more attractive. Either way, it's important to
consider how each change will relate to the big picture. Stand back from time to time
to see the entire landscape and how each part fits into it."

    K = 3;

    retrieveTopK( jag, model, query_text, K )
```

```
Retrieved similar texts:
```

```
Rank: 1
```

```
Vector ID: 1692034183297995000
```

```
Distance: 135676737
```

Text: Check for water bubbling up from the soil. If you see a pool of water or water coming from the soil, then there's a leak in the sprinkler line buried underneath. Mark the general location of the leak or break so you can identify it when the water is turned off. Place an item like a shovel or a rock on the ground near the leak. Turn off the sprinkler system after you've found the leak. If you've found the signs of a leak and located the region where the line is leaking or broken, turn off the water so you can repair the line. Use the shut-off valve in the control box to stop the flow of water through the system.

```
Rank: 2
```

```
Vector ID: 1168196459122135000
```

```
Distance: 137667225
```

Text: In fact, Antarctica is such a good spot for meteorite hunters that crews of scientists visit every year, searching for these otherworldly rocks, driving around the surface until they spot a lone dark rock on an otherwise unbroken expanse of white. However, you don't always have to travel to the other side of the world to find a meteorite. Sometimes meteorites will come to you. Keep an eye open for local reports of brilliant fireballs lighting your region's sky. Debris from such displays scatters across the ground and sometimes hits structures or vehicles. Watch for information about fireballs in your area on the websites of the American Meteor Society or the International Meteor Organization.

```
Rank: 3

Vector ID: 1692034180102679000

Distance: 142799019

Text: Human impact on the environment (or anthropogenic environmental impact) refers
to changes to biophysical environments and to ecosystems, biodiversity, and natural
resources caused directly or indirectly by humans.
```

The full listing of Python3 programs is shown below.

```python
def searchSimilarTexts(jag, model, queryText, K):

    sentences = [ queryText ]

    embeddings = model.encode(sentences, normalize_embeddings=False)

    comma_separated_str = ",".join( [str(x) for x in embeddings[0] ])


    qstr = "select similarity(v, '" + comma_separated_str

    qstr += "', 'topk=" + str(K) + ",type=cosine_fraction_short')"

    qstr += " from  textvec"


    jag.query( qstr )


    jsonstr = ''

    while jag.fetch():

        jsonstr = jag.json()


    return jsonstr



def getTextByVID(jag, vid):

    qstr =" select zid from test. textvec. textvec_idx where v='" + vid + "'"

    zid = ''

    jag.query( qstr )

    while jag.fetch():

        zid = jag.getValue("zid")
```

```python
    qstr = "select text from  textvec where zid='" + zid + "'"

    jag.query( qstr )

    txt = ''

    while jag.fetch():

        txt = jag.getValue("text")


    return txt


def retrieveTopK( jag, model, query_text, K ):

    print("Query: " + query_text )

    json_str = searchSimilarTexts( jag, model, query_text, K )

    json_obj = json.loads(json_str)


    i = 0;

    print("\n")

    print("Retrieved similar texts: ")

    for rec in json_obj:

        dat = rec[str(i)]

        print("\n")

        print("Rank: " + str(i+1))

        vid = dat["id"]

        print("Vector ID: " + vid )

        print("Distance: " + dat["distance"] )

        txt = getTextByVID( jag, vid )

        print("Text: " + txt )

        i += 1


    print("\n\n")
```

Furthermore, extending beyond text embeddings, the capability exists to generate image and video embeddings. These embeddings serve as efficient tools for rapid image and video searches using vector-based techniques. This advancement empowers users to swiftly locate relevant images and videos by exploiting the inherent characteristics captured within the embedding vectors. As a result, the need for

intricate keyword-based searches or complex metadata is significantly reduced, enhancing the speed and accuracy of the search process.

Combing Vector Search and Exact Search

JaguarDB empowers users with a unique synergy of similarity search and exact predicate search. In the context of this integration, consider the following Python illustration: it finds textual instances similar to a given input text while concurrently sifting through records that adhere to specific criteria. The outcome of this combined endeavor is the assignment of similarity values to the retrieved records, a direct consequence of the similarity search's operation. It is noted that the governing criterion, in this case, relates to the source of the text. However, in practical implementation, a number of predicates can be applied.

```
def retrieveTopKWithCriteria( jag, model, queryText, src, K ):
    print("Query: " + queryText )

    sentences = [ queryText ]
    embeddings = model.encode(sentences, normalize_embeddings=False)
    comma_str = ",".join( [str(x) for x in embeddings[0] ])

    qstr = "select similarity(v, '" + comma_str
    qstr += "', 'topk=" + str(K) + ",type=cosine_fraction_short')"
    qstr += " from textvec"
    qstr += " where source='" + src + "'"

    jag.query( qstr )

    print("\n")
    print("Result: ")
    while jag.fetch():
        print('zid={}'.format(jag.getValue("zid")) )
        print('v={}'.format(jag.getValue("v")) )
        print('vectorid={}'.format(jag.getValue("vectorid")) )
        print('rank={}'.format(jag.getValue("rank")) )
        print('distance={}'.format(jag.getValue("distance")) )
        print('source={}'.format(jag.getValue("source")) )
        print('text={}'.format(jag.getValue("text")) )
        print("\n")
```

Anomaly Vector Search

Anomaly detection plays a crucial role in identifying outliers that may indicate instances of fraud, network hacking activities, and various types of attacks such as network intrusions and DDoS attacks. Utilizing vectors as an innovative method to thoroughly examine and assess these anomalies is first provided by JaguarDB.

The following example demonstrates how text, images, videos, or any type of data that can be represented by a vector, can be detected if the input data is anomalous:

```
select anomalous(v, '1,2,3,1,0.3',
                'type=euclidean_whole_float, activation=[1.5:30;2:35;3:20]')
from tab;
```

```
Result:

{"anomalous":"YES","percent":"52;45;23","activation":"1.5:30;2:35;3:20"}]
```

RAG Example Integrating LLM and JaguarDB

Retrieval-Augmented Generation (RAG) emerges as a groundbreaking solution to address the limitations of Large Language Models (LLMs), particularly their propensity to produce hallucinated or factually inaccurate content. RAG operates by ingeniously blending the strengths of retrieval-based models, which excel in sourcing and providing accurate, context-specific information from a vast dataset, with the innovative capacity of generative models known for their ability to create fluent and coherent responses. In this synergistic framework, the retrieval component first fetches the most relevant information pertinent to a given query or context. This information is then seamlessly integrated into the generative model's process, guiding it to produce responses that are not only creative and contextually coherent but also anchored in factual accuracy. Consequently, RAG significantly enhances the reliability and quality of the outputs from LLMs, making them more effective for investment tasks requiring high factual correctness and detailed context understanding.

The following Python example illustrates documents broken into chunks that are stored in JaguarDB which compliments the LLM for answering questions from users. The texts in JaguarDB are fed to the LLM which will analyze all available data and provide a coherent and logical final answer.

```python
from langchain.document_loaders import TextLoader
from langchain.embeddings.openai import OpenAIEmbeddings
from langchain.text_splitter import CharacterTextSplitter
#from langchain.vectorstores.jaguar import Jaguar
from jaguar import Jaguar ## use local class file
from langchain.chains import RetrievalQAWithSourcesChain
from langchain.llms import OpenAI
from langchain.schema.output_parser import StrOutputParser
from langchain.schema.runnable import RunnablePassthrough
from langchain.prompts import ChatPromptTemplate
from langchain.chat_models import ChatOpenAI


loader = TextLoader("./state_of_the_union.txt")
documents = loader.load()
text_splitter = CharacterTextSplitter(chunk_size=1000, chunk_overlap=300)
docs = text_splitter.split_documents(documents)

'''
Create a jaguar vector store
This should be done only once
If the store is already created, you do not need to do this.
'''
url = "http://192.168.3.88:8080/fwww/"
embeddings = OpenAIEmbeddings()

pod = "vdb"
store = "langchain_rag_store"
vector_index = "v"
vector_type = "cosine_fraction_float"
vector_dimension = 1536

vectorstore = Jaguar(pod, store, vector_index,
    vector_type, vector_dimension, url, embeddings
)

vectorstore.login("demouser")

''' create vector on the database
This should be called only once
'''
metadata = "category char(16)"
text_size = 4096
vectorstore.create(metadata, text_size)

# add texts to vectorstore
vectorstore.add_documents(docs)

retriever = vectorstore.as_retriever()
# if use metadata:
#retriever = vectorstore.as_retriever(search_kwargs={"where": "a='123' and
b='xyz'"})
```

```
template = """You are an assistant for question-answering tasks. Use the
following pieces of retrieved context to answer the question. If you don't
know the answer, just say that you don't know. Use three sentences maximum
and keep the answer concise.
Question: {question}
Context: {context}
Answer:
"""
prompt = ChatPromptTemplate.from_template(template)

#LLM = ChatOpenAI(model_name="gpt-3.5-turbo", temperature=0)
LLM = ChatOpenAI(model_name="gpt-4", temperature=0)
rag_chain = (
    {"context": retriever, "question": RunnablePassthrough()}
    | prompt
    | LLM
    | StrOutputParser()
)

query = "What did the president say about Justice Breyer?"
print(f"Question: {query}\n")

r = rag_chain.invoke(query)
print("Answer:")
print(r)
```

The above example depends on the LangChain stack and the jaguar.py store file. You can visit github.com/fserv/jaguar-sdk RAG directory for a complete example.

Jaguar Vector Database API

JaguarDB offers a comprehensive set of application programming interfaces (APIs) tailored to various development needs. These APIs can be seamlessly employed within the jql.bin client terminal or seamlessly integrated into programming languages such as Java, Python, Go, and Node.js. This flexibility empowers developers to interact with JaguarDB using their preferred environment, ensuring a smooth and versatile development experience.

Creating a store for Vectors

```
create store store (
```

```
    key: …KEY…,

    value: VECCOL vector(dimension,'DISTANCE_INPUT_QUANTIZATION'),

    …other_fields…

)
```

The symbol "VECCOL" designates the name of the vector column, while "dimension" denotes the count of components within a vector. Standard dimensions often include values like 768, 1024, 1536, etc. The string "DISTANCE_INPUT_QUANTIZATION" is a vector definition that serves to specify the nature of the distance, input data type, and level of quantization employed in the vector storage and search of similarity between vectors. This comprehensive approach accommodates various distance types, which encompass:

Euclidean Distance

The Euclidean distance, also known as the L2 distance or the Euclidean norm, is a measure of the straight-line distance between two points in a multi-dimensional space. It's commonly used to quantify the similarity between vectors.

$$\text{dist} = \sqrt{\sum_{i=1}^{n}(A_i - B_i)^2}$$

Cosine Distance

Cosine distance is a measure used to quantify the dissimilarity between two vectors in a multi-dimensional space. Unlike the Euclidean distance that measures the direct geometric distance between vectors, the cosine distance focuses on the angle between the vectors.

$$\cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}$$

InnerProduct

Inner product similarity is useful for similarity search in scenarios where the magnitudes of vectors are important in addition to their directions.

$$\vec{A} \cdot \vec{B} = \sum_{i=0}^{n} A_i B_i$$

Manhatten Distance

Manhattan distance is a distance metric between two points in a multi-dimensional vector space. It is the sum of absolute difference between the measures in all dimensions of two points.

$$\text{dist} = \sum_{i=1}^{n} |A_i - B_i|$$

Chebyshev Distance

Chebyshev distance is a metric defined on a vector space where the distance between two vectors is the greatest of their differences along any coordinate dimension.

$$\text{dist} = \max_i (|A_i - B_i|)$$

Hamming Distance

The Hamming distance between two vectors is the number of positions at which the corresponding components are different.

$$\text{dist} = \sum_{i=1}^{n} \Delta(A_i, B_i)$$

Jeccard Distance

The Jeccard distance between two vectors is computed by taking the ratio of Intersection over Union of the two vectors.

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

Minkowski Half

In general, the Minkowski distance of order p is given by:

$$\text{dist} = \left(\sum_{i-1}^{n} |A_i - B_i|^p \right)^{1/p}$$

In JaguarDB, Minkowski Half distance refers to the Minkowski distance where $p = 0.5$.

The input type in JagaurDB refers to the expected data format in the input vectors. There are two input types: fraction and whole. JaguarDB excels not only in managing vector embeddings but also in handling a diverse range of feature vectors. These vectors can include various types and forms, whether they are normalized or unnormalized, presented in fractional or full original formats. This versatility underscores JaguarDB's capability to accommodate a wide array of data formats.

Fraction Input Format

Each component of a vector is in the range of [-1.0, +1.0], inclusive. An example of a such a vector would be: "0.1, 0.02, -0.04, -0.5, 0.12, 0.53".

Whole Input Format

Components of a vector are not limited to the range of [-1.0, +1.0]. They can be in any range. However, they could be trimmed and converted to the range that is required by the quantization level as described below.

Quantization Level

There are two quantization levels in JaguarDB: byte and short. The process of quantizing input vectors yields efficient memory utilization within the system. While storing a float number demands 4 bytes, employing fewer bytes for storing vector components can yield substantial memory savings. When components are stored as signed integers, memory savings can reach 50%, while utilizing only a single byte for vector components can result in an impressive 75% reduction in memory usage. This approach is termed "short quantization level" for the utilization of signed integers and "byte quantization level" for the use of a single byte. The quantization of input vectors aligns with the level specified by the user during vector creation, optimizing memory consumption while maintaining data integrity.

With byte (8-bit) quantization level, the number of quantized hyper cubes in a 1024-dimensional hyperspace is 256^{1024} which is already a large number and vector distribution would be sparse. With a short (16-bit) quantization level, the number of available hypercubes is even larger. In rare application scenarios, the vectors could be densely populated around clusters. A 16-bit quantization may provide higher resolution of differentiating vectors than an 8-bit quantization. It is a trade-off between storage size and accuracy in searching nearest neighbors. If no quantization is used, then the float type can be selected. An 8-bit or 16-bit quantization tend to work well to vectors of lower dimensions.

Multiple Search Types

During the creation of a vector store, the second argument within the "vector()" field description, or key definition, offers the flexibility to incorporate multiple instances of "DISTANCE_INPUT_QUANTIZATION". For instance, it can appear as a series of "cosine_fraction_byte, hamming_whole_short". This allows users to specify multiple distance types and quantization levels, albeit limited to a single input type for the same distance and quantization level. Notably, distinct vector data stores are managed for each unique combination of the three types, ensuring the effective organization of data based on these parameters.

List of Key Definitions

Key Definition	Distance	Input (component x)	Quantization
euclidean_fraction_short	Euclidean	-1.0 <= x <= +1.0	16-bit integer
euclidean_fraction_byte	Euclidean	-1.0 <= x <= +1.0	8-bit integer
euclidean_whole_short	Euclidean	-32767 <= x <= 32767	16-bit integer
euclidean_whole_byte	Euclidean	-127 <= x <= 127	8-bit integer
cosine_fraction_short	Cosine	-1.0 <= x <= +1.0	16-bit integer
cosine_fraction_byte	Cosine	-1.0 <= x <= +1.0	8-bit integer
cosine_whole_short	Cosine	-32767 <= x <= 32767	16-bit integer
cosine_whole_byte	Cosine	-127 <= x <= 127	8-bit integer
innerproduct_fraction_short	Inner Product	-1.0 <= x <= +1.0	16-bit integer
innerproduct_fraction_byte	Inner Product	-1.0 <= x <= +1.0	8-bit integer
innerproduct_whole_short	Inner Product	-32767 <= x <= 32767	16-bit integer
innerproduct_whole_byte	Inner Product	-127 <= x <= 127	8-bit integer
manhatten_fraction_short	Manhatten	-1.0 <= x <= +1.0	16-bit integer
manhatten_fraction_byte	Manhatten	-1.0 <= x <= +1.0	8-bit integer
manhatten_whole_short	Manhatten	-32767 <= x <= 32767	16-bit integer
manhatten_whole_byte	Manhatten	-127 <= x <= 127	8-bit integer
hamming_fraction_short	Hamming	-1.0 <= x <= +1.0	16-bit integer
hamming_fraction_byte	Hamming	-1.0 <= x <= +1.0	8-bit integer
hamming_whole_short	Hamming	-32767 <= x <= 32767	16-bit integer
hamming_whole_byte	Hamming	-127 <= x <= 127	8-bit integer
chebyshev_fraction_short	Chebyshev	-1.0 <= x <= +1.0	16-bit integer
chebyshev_fraction_byte	Chebyshev	-1.0 <= x <= +1.0	8-bit integer
chebyshev_whole_short	Chebyshev	-32767 <= x <= 32767	16-bit integer
chebyshev_whole_byte	Chebyshev	-127 <= x <= 127	8-bit integer
minkowskihalf_fraction_short	MinkowskiHalf	-1.0 <= x <= +1.0	16-bit integer
minkowskihalf_fraction_byte	MinkowskiHalf	-1.0 <= x <= +1.0	8-bit integer
minkowskihalf_whole_short	MinkowskiHalf	-32767 <= x <= 32767	16-bit integer
minkowskihalf_whole_byte	MinkowskiHalf	-127 <= x <= 127	8-bit integer
jeccard_fraction_short	Jeccard	-1.0 <= x <= +1.0	16-bit integer
jeccard_fraction_byte	Jeccard	-1.0 <= x <= +1.0	8-bit integer
jeccard_whole_short	Jeccard	-32767 <= x <= 32767	16-bit integer

jeccard_whole_byte	Jeccard	-127 <= x <= 127	8-bit integer
euclidean_fraction_float	Euclidean	float	32-bit float
euclidean_whole_float	Euclidean	float	32-bit float
cosine_fraction_float	Cosine	float	32-bit float
cosine_whole_float	Cosine	float	32-bit float
innerproduct_fraction_float	InnerProduct	float	32-bit float
innerproduct_whole_float	InnerProduct	float	32-bit float
manhatten_fraction_float	Manhatten	float	32-bit float
manhatten_whole_float	Manhatten	float	32-bit float
hamming_fraction_float	Hamming	float	32-bit float
hamming_whole_float	Hamming	float	32-bit float
chebyshev_fraction_float	Chebyshev	float	32-bit float
chebyshev_whole_float	Chebyshev	float	32-bit float
minkowskihalf_fraction_float	MinkowskiHalf	float	32-bit float
minkowskihalf_whole_float	MinkowskiHalf	float	32-bit float
jeccard_fraction_float	Jeccard	float	32-bit float
jeccard_whole_float	Jeccard	float	32-bit float

Adding Vectors

JaguarDB can integrate all application and vector data, facilitating streamlined data management for real-world scenarios. It enables the incorporation of vector data alongside other pertinent information related to business objects, allowing for comprehensive and cohesive data representation.

```
insert into store ( …, VECCOL, …) values (…, 'VECTOR_STRING', … )
insert into store values (…, 'VECTOR_STRING', … )
```

Where VECTOR_STRING is a list of comma-separated components of the vector. In the second statement, the values must be provided according to the correct order of the

columns in the store. Once the vector is added, the value of the field for VECCOL will be replaced with an integer as the unique identifier for the vector. With a vector ID, the components of the vector can be retrieved from the vector database.

Similarity Search

Similarity search using JaguarDB vectors involves the process of finding vectors within the database that are most similar to a given query vector. This search is conducted based on predefined similarity metrics, such as cosine similarity or Euclidean distance similarity, which quantify the resemblance between vectors. The API for similarity search is as follows:

```
select
similarity(v, 'QUERY_VECTOR',
'topk=K,type=DIST_INPUT_QUANT,with_score=yes,with_text=yes')
from store;
```

where QUERY_VECTOR is a list of comma-separated component values of the vector. The number "K" specifies the number of most similar vectors to be found and returned for the query vector. The returned result is in the JSON format and the developer can call the json() function to parse the JSON format and retrieve the ID and distance values.

As an example, the following statement returns the top 5 most similar vectors to the query vector:

```
select similarity(v, '0.1, 0.2, 0.3, 0.4, 0.5, 0.3, 0.1',
'topk=5,type=manhatten_fraction_byte') from vec1;
```

Multimodal Similarity Search

In various application scenarios, there arises a need for users to perform targeted queries on a dataset, ensuring that the retrieved data records not only adhere to certain criteria but also exhibit a certain level of similarity to a provided data sample. This intricate task demands the identification of vectors that are both closely related and satisfy specific prerequisites. With the innovative capabilities of JaguarDB, this complex process can be streamlined into a single step. Through the integration of similarity search alongside selective criteria, JaguarDB facilitates the discovery of nearest neighbors that fulfill predefined qualifications. This advanced functionality empowers users to seamlessly locate a subset of data records and subsequently assess their likeness to a reference vector, resulting in the assignment of similarity rankings. By encompassing both the aspects of similarity and tailored selection, this approach significantly mitigates the potential for inaccuracies, making it particularly well-suited for environments characterized by stringent business requirements.

JaguarDB's unique amalgamation of similarity-based search and tailored qualification selection brings unprecedented efficiency to the intricate task of querying and comparison. Once a cohort of relevant data records is extracted, their alignment with a given vector is precisely evaluated, generating a hierarchy of similarity rankings. This integrated approach is instrumental in refining the matching process, ensuring that data records not only exhibit the desired attributes but also possess a designated degree of resemblance to a reference sample. This holistic functionality carries substantial benefits, especially in high-stakes scenarios where precision is paramount. By converging the twin challenges of similarity and criterion-based filtering, JaguarDB effectively minimizes the potential for inaccuracies, offering a robust solution for industries demanding precise data retrieval and analysis. Through this innovative approach, JaguarDB empowers users to navigate the complexities of data exploration with enhanced accuracy and confidence, establishing itself as a pivotal tool in the pursuit of data-driven excellence.

The following similarity search statement is extended with the "where clause" to filter the nearest neighbors of the input query vector:

```
select
similarity(v, 'QUERY_VECTOR',
'topk=K,fetch_k=N,type=DIST_INPUT_QUANT,with_score,metadata=col3&col4')
from store
```

```
      where attribute1 = … and attribute2 = …;
```

For example:

```
select similarity(v, '0.1, 0.2, 0.3, 0.4, 0.5, 0.3, 0.1',
'topk=10,fetch_k=100,type=manhatten_fraction_float,metadata=zip')
from vec1
where customer_region='NE' and marriage_status='single';
```

In this illustrative scenario, the foremost consideration involves the establishment of a fetch_k=100 similar records. Simultaneously, another set of data is determined by the criteria enlisted in the "where" clause. The intersection of these two sets of data, with a maximum of topk=10 records, is returned to the user. The number of fetch_k should be large enough to produce topk records that are qualified by the where clause.

Time Decayed Similarity Search

Time-decayed similarity search of vectors involves adjusting semantic similarity based on time-weighted factors. This means that vectors representing older data, in terms of vector creation time, have a reduced impact on similarity scores compared to more recently created vectors, assuming their semantic similarity is identical. The degree of time decay is quantified by a decay rate, which can be calculated on various time scales such as weekly, daily, hourly, or by the minute.

When computing the effect of time decay, there are two available modes: power mode and exponential mode. These modes are specified as "decay_mode=P" for power mode or "decay_mode=E" for exponential mode. By default, the power mode is used.

In the power decay mode, the distance and similarity score values are adjusted as follows:

```
r = 1.0 – decay_rate
```

```
adjusted_distance = (2.0 * distance)/(1 + pow(r, n))


adjusted_score = score * pow(r, n)
```

where "n" is the number of intervals passed between now and the creation time of the vectors; "pow" is the power operation, that is, the value of r^n.

In the exponential decay mode, the distance and similarity score are adjusted according to the following transformations:

```
f = decay_rate
adjusted_distance = (2.0 * distance)/(1 + 1.0/exp(f * n))
adjusted_score = score/exp(f * n)
```

where "exp" is the base-e exponential function.

An example of applying time-decay in similarity search is:

```
select similarity(v, '0.1, 0.2, 0.3, 0.4, 0.5, 0.3, 0.1',

'topk=10,fetch_k=100,type=manhatten_fraction_float,day_decay_rate=0.01,
decay_mode=P')

from vec1
```

In addition to daily decay considerations, users can use "week_decay_rate", "hour_decay_rate", "minute_decay_rate" , and "second_decay_rate" for weekly, hourly, by minute, by second decay adjustments.

It should be noted that the decay adjustments are performed on the vectors retrieved with the "fetch_k" number of vectors. A relatively large number of fetch_k is recommend to obtain more accurate results. To fully capture the dynamic variations in time dimension and semantic dimension, it would require the system to rebuild the

vector data structure (HNSW graph) constantly as time progresses. In practice this is hard or simply impossible. The solution is using a large number of fetch_k and then adjusting the similarity according to the elapsed time of the selected vectors.

In essence, time-decayed similarity search enables the adjustment of semantic similarity by considering the temporal dimension, ensuring that older vectors contribute less to similarity calculations compared to newer vectors with the same semantic similarity. This approach is valuable for applications where the relevance of data diminishes over time.

Time Cutoff

In specific scenarios, applications may choose to disregard exceptionally old vectors when conducting similarity searches. JaguarDB offers a solution for this by introducing the time cutoff parameter in its search functionality. Vectors with creation times preceding the specified values are automatically excluded from the search results. The available time cutoff parameters include week_cutoff, day_cutoff, hour_cutoff, minute_cutoff, and second_cutoff allowing for precise control over the temporal scope of the search.

```
select similarity(v, '0.1, 0.2, 0.3, 0.4, 0.5, 0.3, 0.1',

'topk=10,fetch_k=100,type=manhatten_fraction_float,day_cutoff=365,day_d
ecay_rate=0.01,decay_mode=P')

from vec1
```

The cutoff option can be used independently or together with the decay rate values. In the above example, vectors that are 366-days older or earlier are ignored in the search.

Anomaly Detection

Jaguar vector database is revolutionizing the way businesses approach anomaly detection. It provides a structured and efficient means of storing and querying data, enabling organizations to analyze patterns and deviations with remarkable precision. This innovative technique not only streamlines the process of anomaly detection but

also enhances the accuracy of identifying potential threats. As the business landscape continues to evolve in an increasingly digital world, leveraging vector databases for anomaly detection has become a strategic imperative for enterprises seeking to safeguard their operations and data from malicious activities.

The API for analyzing anomaly is shown below:

```
select
anomaly(VECCOL,
            'type=DISTANCE_INPUT_QUANTIZATION,[slices=N]')
from store
```

where the type specifies the distance type and quantization levels of vectors; the optional parameter slices is the number of slices that divides 4-standard deviations of the distribution of all the vectors in the vector store. The default value of slices is 20. Some examples of anomaly detection are listed as follows:

```
select
anomaly(vc,
            'type=euclidean_whole_float')
from myvector;
```

```
select
anomaly(vc,
            'type=euclidean_whole_float, slices=20')
from myvector;
```

Result:

[{"sigma":"0.1","prate":"0.9"},{"sigma":"0.3","prate":"0.9"},{"sigma":"0.5","prate":"0.7"},{"sigma":"0.7","prate":"0.7"},{"sigma":"0.9","prate":"0.7"},{"sigma":"1.1","prate":"0.6"},{"sigma":"1.3","prate":"0.6"},{"sigma":"1.5","prat

e":"0.6"},{"sigma":"1.7","prate":"0.6"},{"sigma":"1.9","prate":"0.6"},{"sigma":"2.1","prate":"0.5"},{"sigma":"2.3","prate":"0.4"},{"sigma":"2.5","prate":"0.4"},{"sigma":"2.7","prate":"0.4"},{"sigma":"2.9","prate":"0.4"},{"sigma":"3.1","prate":"0.4"},{"sigma":"3.3","prate":"0.4"},{"sigma":"3.5","prate":"0.4"},{"sigma":"3.7","prate":"0.2"},{"sigma":"3.9","prate":"0.2"}]

The API for detecting anomalousness is shown below:

```
select

anomalous(VECCOL,

'type=DISTANCE_INPUT_QUANTIZATION,activation=[sigma1:percent1&sigma2:percent2&sigma3:percent3&…]')

from store
```

where the type specifies the distance type and quantization levels of vectors; the parameter activation specifies one or more "sigma:percent" pairs instructing how much percent of vector components must be greater than the sigma value in order to be classified as anomalous. For instance, if activation is "0.3:70&1:50&1.5:30&3:10", then at 0.3 times of Sigma, there must be more than 70 percent of vector components that exceed this 0.3*sigma value; and at one Sigma, there must be more than 50 percent of vector components that exceed this one sigma value; and finally at three Sigma, there must be more than 10 percent of vector components that exceed this one sigma value. If any condition is not met, then the query vector is not classified as an anomalous vector.

```
select

anomalous(vc,

'type=euclidean_whole_float,activation=[0.3:60&1:50&1.5:30&3:10]')

from myvector;
```

Result:

{"anomalous":"YES","percent":"74&60&40&12","activation":"0.3:60&1:50&1.5:30&3:10"}

Retrieving Vectors

In cases where users need to retrieve the component values of a vector, the following API can be used:

```
select
vector(VECCOL, 'type=DISTANCE_INPUT_QUANTIZATION')
from store
where KEY=…
```

For example,

```
select vector(v, 'type=manhatten_fraction_short')
from vec1
where fid='ANjf848223@01'
```

The utilized KEY in the query must uniquely identify a record housing the vector, typically involving the exclusive use of the ZeroMove unique ID.

Updating Vectors

The vector components can be updated with two approaches:

```
update store
set VECCOL:vector='VECTOR_STRING'
where KEY=…
```

```
update store

set VECCOL:vector='VECTOR_ID:VECTOR_STRING'

where 1
```

where VECTOR_ID is the integer value of the vector ID, and VECTOR_STRING is a list of comma-separated component values.

Deleting Vectors

The vector components cannot be deleted separately without deleting the record containing the vector. A store record can be deleted with the following command:

```
Delete from store
where KEY=…
```

The KEY in the above statement must uniquely identify a record housing the vector, typically the ZeroMove unique ID. In addition, dropping or truncating a store will delete the associated vectors as well.

Environment

JaguarDB consists of both Server and Client packages. You may install Server and Client packages either on the same host or on multiple hosts.

System Requirements for Jaguar Server

Hardware : CPU 8 Core, 2GHz, 4GB RAM, 500GB HDD or SSD
Software : Linux, CentOS, RedHat, Ubuntu, x86_64, Docker
File systems : ext4, XFS, ZFS

System Requirements for Jaguar Client

Hardware : CPU 4 Core, 2GHz, 1GB RAM, 256GB HDD or SSD
Software : Linux, CentOS, RedHat, Ubuntu, x86_64, Docker
File systems : ext4, XFS, ZFS

Cloud Framework

JaguarDB is a versatile vector database solution that offers comprehensive support for both public and private cloud environments. Organizations can seamlessly deploy and manage JaguarDB in either setting, accommodating multiple tenants or business units within their structure.

Within each tenant, there is the potential for multiple member users. Importantly, the data of different tenants remains isolated and independent, ensuring data security and integrity. Conversely, data within a given tenant is shared among all members of that particular tenant, fostering collaboration and data accessibility. This cloud-based framework strikes an optimal balance between flexibility and security.

Moreover, within each tenant, there is an essential administrative structure. An admin account is designated to carry out administrative tasks exclusively for the members within that tenant. Additionally, a superadmin oversees the entire system of the Jaguar distributed database, operating and maintaining its overall functionality. Both admin account types possess their own distinct and secure API keys.

It is crucial to emphasize that all member users are uniquely identified through their secret API keys, which are associated with their respective tenants. The concept of a "tenant" is a generalized term that can refer to a business unit, department, or any distinct group within an organization, providing adaptability to various organizational structures.
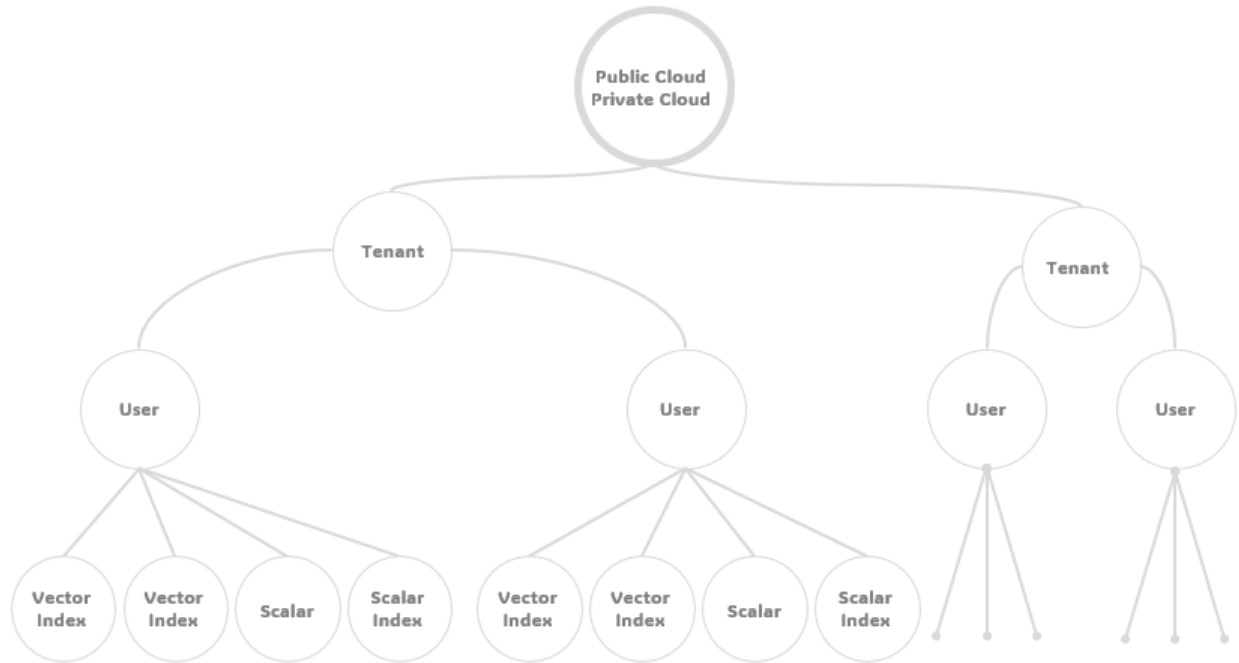
In addition to the multi-tenant and multi-cloud framework, JaguarDB presents an additional framework tailored to internet users. Within this personalized framework, JaguarDB automatically generates a distinct tenant for each user upon their signup for a JaguarDB account or any related service. Essentially, every user becomes a self-contained tenant in their own right. This approach ensures that the data of different users remains entirely isolated and independent, upholding the highest standards of security and privacy.

Given the potential for a large number of users in this scenario, the distributed databases may necessitate the sharding of user accounts as per the specific requirements of cloud operators. Sharding enables efficient management and scaling of data to meet the unique demands of the system, contributing to a robust and adaptable user experience.
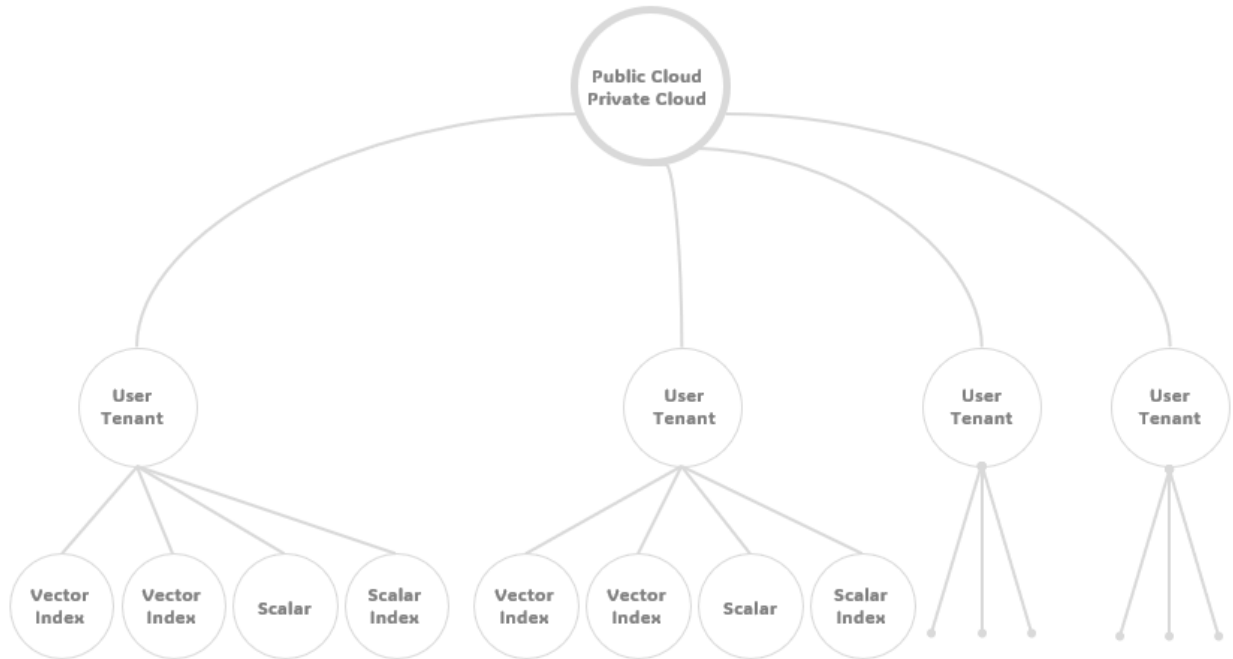
JaguarDB Installation

You can use docker pull to install jaguardb image, and docker run to start running jaguardb in your docker container. You could also download all binary packages for 64-bit machines of Jaguar vector database software and then you can install Jaguar with just one script. You may install the binaries either on the same machine or different servers. Jaguar Server will listen on TCP/IP port 8888, and the process name is "jaguar.bin". The process can be started by any user, each having a different listening port. Jaguar provides shell scripts to start and stop the Jaguar servers.

Operating Systems

JaguarDB supports Linux 64-bit operating systems. There are Linux jaguar server and client libraries in the downloaded files.

Linux System

In the docker image, latest Ubuntu system is used. In a Linux 64-bit system (such as Centos 7 or Ubuntu), you can open a terminal (or putty, xterm, etc.) and saved the downloaded file in any directory.

JaguarDB Server and Client Setup

There are three methods for setting up JaguarDB on your Linux system.

Docker Container Deployment: The first method involves deploying JaguarDB using Docker containers. This approach offers ease of management and scalability.

Host-Based Installation and Cluster Configuration: The second approach is to install JaguarDB individually on each host and then establish connectivity between these hosts by sharing a common cluster configuration file. This method provides more control over the installation process and allows for fine-tuned cluster management.

Public Key Authentication and Cluster-Wide Installation Script: The third method involves setting up public keys on all hosts within the cluster. Subsequently, you can run an installation script from one node within the cluster to deploy JaguarDB on all the nodes in the cluster. This approach streamlines the deployment process while maintaining security through key-based authentication.

Consider these options to select the most suistore approach for your specific requirements when setting up JaguarDB on your Linux system.

Installation Method One

For a quick setup of JaguarDB in one docker container, you can run the following commands:

```
sudo docker pull jaguardb/jaguardb
```

Then you can start up the JaguarDB process in the docker container:

```
docker run -d -p 8888:8888 --name jaguardb jaguardb/jaguardb
```

Once the JaguarDB server process is started in the docker container, you can use the JaguarDB client terminal to talk to the JaguarDB server in the container:

```
docker exec -it jaguardb /home/jaguar/jaguar/bin/jag
```

The above command logs into an admin account which can create tenants and user accounts.

```
docker exec -it jaguardb /home/jaguar/jaguar/bin/jag -apikey demouser
```

The Jaguar docker container includes a HTTP gateway server and a vector database server. In addition, there are programming API in Java, Python, Golang, Nodejs to query the JaguarDB vector databases. The JaguarDB docker container is based on Ubuntu 22.04.

Installation Method Two

If you are on macOS or Windows, you must use the Docker image to get JaguarDB up and running. However, if you are on a Linux system or inside a Linux-based Docker container, the following command can quickly install and launch the JaguarDB vector store along with its HTTP gateway server by downloading, installing, and starting the necessary components.

```
curl -fsSL http://jaguardb.com/install.sh | sh
```

The JaguarDB component is installed under $HOME/jaguar directory, and the Http gateway is installed under $HOME/fwww directory.

Installation Method Three

In this method, basically you can download and copy the jaguar-n.n.n.tar.gz program and install it manually on each host of your database cluster, and then configure the $JAGUAR_HOME/conf/cluster.conf file which should be shared by all the hosts in the

system. The download link for JaguarDB software is www.jaguardb.com/download.html
and you can follow the link to download the tar ball to your host.

On each host in a cluster, you can perform the following steps:

1) Copy the package jaguar-n.n.n.tar.gz to each host
2) $ tar -zxf jaguar-n.n.n.tar.gz

3) Then related files will be unzipped into jaguar-n.n.n directory:

   $ cd jaguar-n.n.n
   $ ./install.sh

   The script install.sh takes an option "-d <INSTALLATION_PATH>" in order to
   install Jaguar to an alternative directory instead of the default $HOME/jaguar
   location.

4) In the conf/cluster.conf file, place the IP addresses of all your hosts and copy
   this file to the conf directory on every host in the JaguarDB system. You can
   start up JaguarDB on all the hosts with the script:

   $ $JAGUAR_HOME/bin/jaguarstart_on_all_hosts.sh

Installation Method Four

This method uses a downloaded tar ball and a script to install JaguarDB software on
all hosts for your database cluster. To implement this, ensure that you employ
identical account credentials across all hosts. This process will establish trusted public
authorized keys for seamless SSH login, eliminating the need for repetitive password
input.

If you use Linux hosts, on any server in your cluster, you can execute the following
script:

```
$ tar -zxf jaguar-n.n.n.tar.gz
```

Then related files will be unzipped into jaguar-n.n.n directory:

```
$ cd jaguar-n.n.n
```

If you are operating within a Linux environment with an active sshd server and ssh client, deploying JaguarDB across the entire cluster of servers can be achieved effortlessly using a single command:

```
$ ./install_jaguar_database_on_all_hosts.sh  -f HOSTFILE
```

You must create the HOSTFILE which should contain the IP address of all hosts in the Jaguar database cluster.

Example of HOSTFILE:

```
Ip_of_node1
Ip_of_node2
Ip_of_node3
Ip_of_node4
Ip_of_node5
```

The Ip_of_nodeN means the IP (v4) address of the N-th node. **Prior to installing Jaguar on all hosts with this approach, please make sure you have a user account on all the hosts that have the same password for the user.** The HOSTFILE file is very important for setting up your database cluster. This installation method will set-up trusted public keys on the hosts that will be included in the cluster. If you accidently make an error in the installation process, you can execute the following command to uninstall JaguarDB on all the hosts you have prepared in the HOSTFILE:

```
$ ./uninstall_jaguar_database_on_all_hosts.sh
```


The script `install_jaguar_database_on_all_hosts.sh` also takes "-d <TARGETDIRECTORY>" command option to have Jaguar installed on a different directory other than $HOME/. If a different directory is used to install Jaguar, then the JAGUAR_HOME environment variable in the bin/jaguarstart, bin/jaguarstatus, bin/jaguarstop scripts is set to this directory. By default, the JAGUAR_HOME directory as an environment variable is set to $HOME of the user who is installing Jaguar. Once it is set, $HOME/.jaguarhome file will contain the path value of $ JAGUAR_HOME.

If you have installed jaguar in the past, later when you are upgrading jaguar with new releases, the "-f HOSTFILE" will not be required.

File $HOME/.jagsetupssh is created when user's public keys have been setup on all hosts in the cluster. If this file does not exist, "setupsshkeys -f CONFFILE" command is executed to set up the public keys. The setupsshkeys program can be executed anytime to have the public keys installed in the cluster.

The above scripts will copy config file *server.conf* to $JAGUAR_HOME/conf/ and jaguar programs to $JAGUAR_HOME/bin/. You should set up your $PATH environment variable to include the directory $JAGUAR_HOME/bin.

Configuration file $JAGUAR_HOME/*conf/server.conf* includes the following parameters:

■ PORT is the listening port number of Jaguar server.

■ LISTEN_IP is the IP address that the server will use if there are multiple network interfaces on the same server host. If there is only one IP address on the server host, this parameter should be commented out and ignored.

■ MEMORY_MODE specifies whether more or less memory will be used by jaguar server. If high is specified, then a little more memory is used by Jaguar. If low is given, then less memory is used by Jaguar. Default value is high.

■ REPLICATION is the number of copies for each data record. For every data record, it is replicated to multiple hosts. The default value is 3. If the number of servers is less than 3, then the replication number is equal to the number of servers. Once the system is up and running, the parameter cannot be changed. For free-trial version, this parameter is always one, i.e., data is not replicated.

■ BUFF_READER_BLOCKS   When scanning a store, blocks of underlying file are loaded into a buffer which size is specified by this number. Higher number can boost performance during join or any scan operations. Default value is 4096.

■ JAG_LOG_LEVEL   Lower number (min is 0) makes the server generate less logging messages. A higher number (max is 9) makes the server generate more debugging information.

■ LOCAL_BACKUP_PLAN Specifies when and how data is backed up. There are five types of intervals when duplicate data is saved: 15MIN, HOURLY, DAILY, WEEKLY, and MONTHLY. When data is saved, it can be either SNAPSHOT or OVERWRITE mode. SNAPSHOT means each and separate copy of data is saved with a timestamp (uses more storage space as times goes on). OVERWRITE means only one copy of data is saved. The format for LOCAL_BACKUP_PLAN is frequency:policy|frequency:policy|… where frequency is one of 15MIN, HOURLY, DAILY, WEEKLY, and MONTHLY, and policy is one of SNAPSHOT or OVERWRITE. If no value is provided for BACKUP_PLAN, then no data is saved as backup.

■ REMOTE_BACKUP_SERVER and REMOTE_BACKUP_INTERVAL: These parameters specify remote backup server IP address and backup interval in seconds. The remote backup server can be a SAN storage server and must have enough capacity. If these parameters are provided, all servers in the cluster will periodically send local data to the remote server for backup.

Configuration file $JAGUAR_HOME/*conf/cluster.conf* is created from the HOSTFILE when executing the `install_jaguar_database_on_all_hosts.sh` script and it includes the following parameters:

    Host IP of server 1
    Host IP of server 2
    Host IP of server 3
    Host IP of server 4
    ……

For example (conf/cluster.conf):

    192.168.1.101
    192.168.1.102
    192.168.1.103
    192.168.1.104

Make sure that cluster.conf is the same on all server hosts. Please note that you cannot modify cluster.conf. If you want to add more hosts in cluster.conf, you must use the "addcluster" command. The file server.conf should be the same on all hosts (except that LISTEN_IP is different in case it is used).

Jaguar Server Startup

Linux System

On a Linux system, you may start Jaguar server on all hosts with this command:

```
$ $JAGUAR_HOME/bin/ jaguarstart_on_all_hosts.sh
```

Then Jaguar server will listen on port 8888. After Server is started up, you can login using the "admin" account and "jaguarjaguarjaguar" as password.   It is recommended that you change the password for admin account. You may create more Databases and User Accounts. The server log file will be in $JAGUAR_HOME/log/ directory.

You may also check the status of Jaguar on all hosts:

```
$ $JAGUAR_HOME/bin/ jaguarstatus_on_all_hosts.sh
```

All Jaguar server processes can be stopped with:

```
$ $JAGUAR_HOME/bin/ jaguarstop_on_all_hosts.sh
```

HTTP Gateway Setup

In the installation methods 1 and 2 as described in the installation section of the manual, the HTTP gateway and the vector database components are installed and deployed automatically. In the methods 3 and 4, the HTTP gateway server needs to be manually installed separately.

In tandem with the JaguarDB server, the HTTP gateway provides a universal endpoint for clients to seamlessly interact with Jaguar servers. This gateway serves as a proficient proxy, facilitating the relay of commands from clients to the backend database servers. Consequently, clients are liberated from the constraints of relying on a specific Linux platform. The HTTP gateway software can be conveniently obtained free of charge from the following link:

```
http://www.jaguardb.com/download.html
```

Users are encouraged to adhere to the installation instructions bundled within the package. To set up the Jaguar HTTP gateway on any Linux system, users can follow these straightforward steps:

1) Use the 'wget' command to acquire the necessary files:

```
wget http://www.jaguardb.com/download/fwww_n.n.n.tar.gz
```

2) Execute the './install.sh' script to initialize the installation process
3) Configure the settings within '$HOME/fwww/conf_dir/fwww.conf'
4) Navigate to the '$HOME/fwww/bin_dir' directory and run './start_all_servers.sh'

It is important to note that the gateway process should ideally run on the same host as one of the Jaguar database servers, although it can also be deployed on separate hosts within a local area network for enhanced flexibility. The value of configuration parameter JAGUAR_SUPER_ADMIN_API_KEY in the fwww.conf file should be read from the configuration file "$JAGUAR_HOME/conf/server.conf" on one of a JaguarDB

servers. When you open the file server.conf, you will see SERVER_TOKEN and you can use its value for the JAGUAR_SUPER_ADMIN_API_KEY entry in the fwww.conf configuration file. Note that JaguarDB server should be installed prior to setting up the HTTP gateway. If JaguarDB server is restarted, you should restart the http server.

Once the http gateway is setup, you point your browser at:

```
http://<IP>:8080/login.html
```

to login to your account and manage your data, where <IP> is the IP address of the server where the fwww http server is installed.
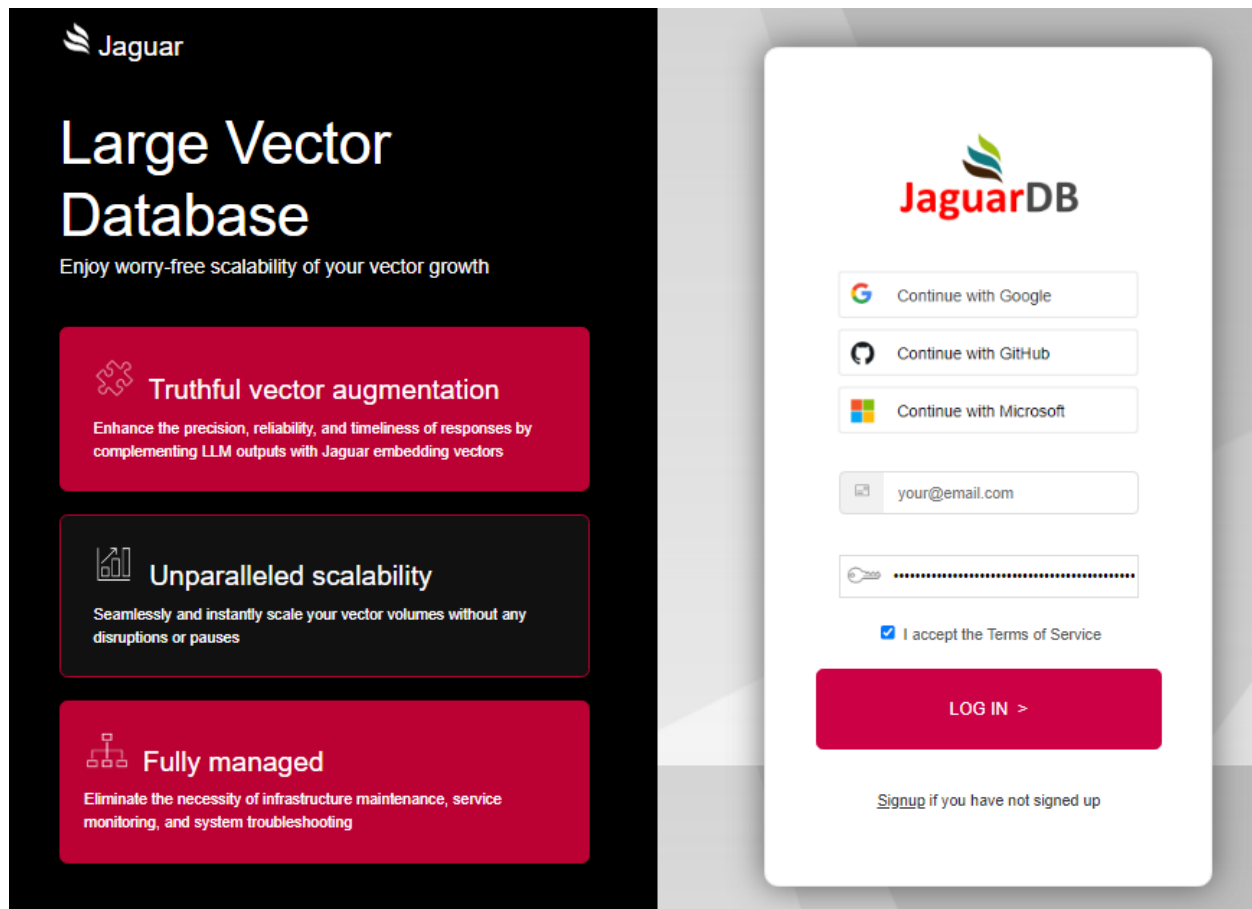
You can contact your tenant admin to get our API key. The admin can use the following command in jql.bin (or jag) terminal program to create user accounts for their users:

```
$JAGUAR_HOME/bin/jag

jaguardb> makeapikey;

jaguardb> createuser <APIKEY> <EMAIL> <LEVEL> <TENANT>;
```

The value of LEVEL must be one of 100, 1000, 2000, 3000, 4000, and 5000. The value of TENANT must be the same as that of the admin API key. If the admin is the superadmin, then the TENANT can be any (if it does not exist, the system will create a new tenant). The superadmin account can also create a new tenant with the following command:

```
jaguardb> create tenant NEW_TENANT;
```


The "jaguardb-http-client" package is readily accessible to clients operating on a multitude of platforms, including macOS, Windows, Linux, iOS, and Android. All they need to do is install machine learning packages on their local systems. To connect to

the JaguarDB server, clients simply need to interface with an HTTP gateway server, which effectively functions as a proxy for the JaguarDB server. This design confers substantial flexibility to clients, accommodating those operating on diverse systems.

Jaguar Architecture

The Distributed Jaguar Vector Database system boasts an immensely scalable design founded on a cluster-based architecture. The scaling of JaguarDB revolves around clusters, each representing a coherent group of nodes. The system seamlessly accommodates the addition of new clusters at any point, imparting flexibility without disruptions. To fortify data reliability, three replicas are available, ensuring redundancy. For optimal write and read speeds, data distribution occurs across server nodes through sharding. Notably, any Jaguar client can effortlessly establish connections to any Jaguar server.

Data synchronization occurs in real-time across all Jaguar servers, sustaining updated information consistency. The core strength of the Jaguar system lies in its linear scalability; deploying additional server hosts yields proportional increments in storage capacity and performance, following a nearly linear trajectory.
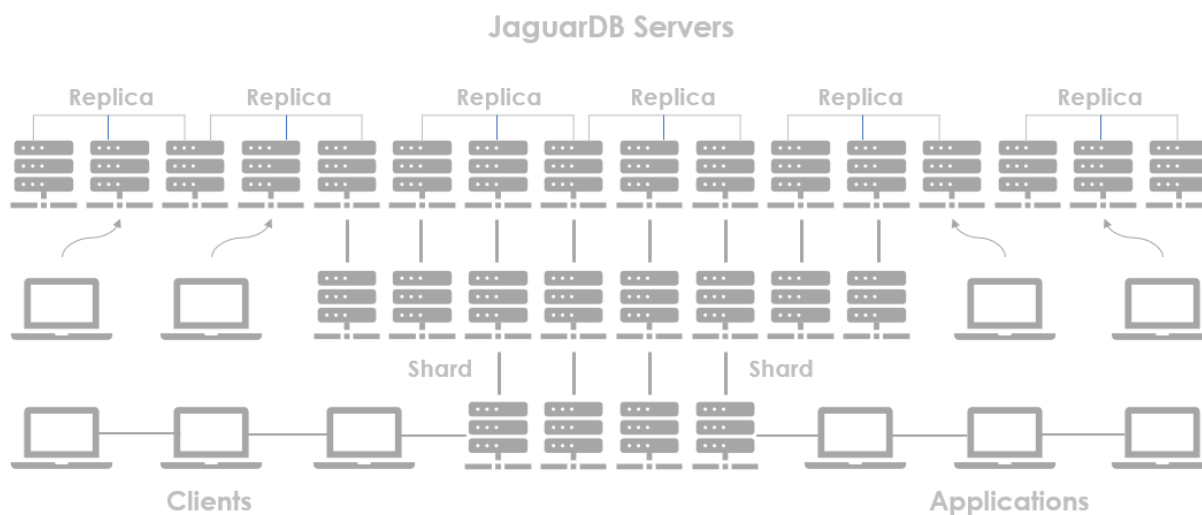
The Shared-Nothing Master-Master architecture in JaguarDB empowers all hosts to efficiently receive, store, and facilitate data reads. This innovative design fosters a distributed data environment where every host participates actively, culminating in

enhanced data input/output speeds and seamless linear scalability. By leveraging the collective capabilities of multiple hosts, JaguarDB optimizes AI data management, ensuring smoother operations and accommodating growing data demands, either structured data or large volume unstructured data such as media files.

JaguarDB Server is versatile and can be effortlessly deployed on a wide range of Linux platforms, including but not limited to Ubuntu, Red Hat, Fedora, and others. The server software is compatible with any 64-bit Linux distribution. Clients, on the other hand, enjoy flexibility and compatibility as they can initiate connections from various platforms, provided that Python 3 is installed, and the jaguardb-http-client package is available. The python package can be installed with the pip command.
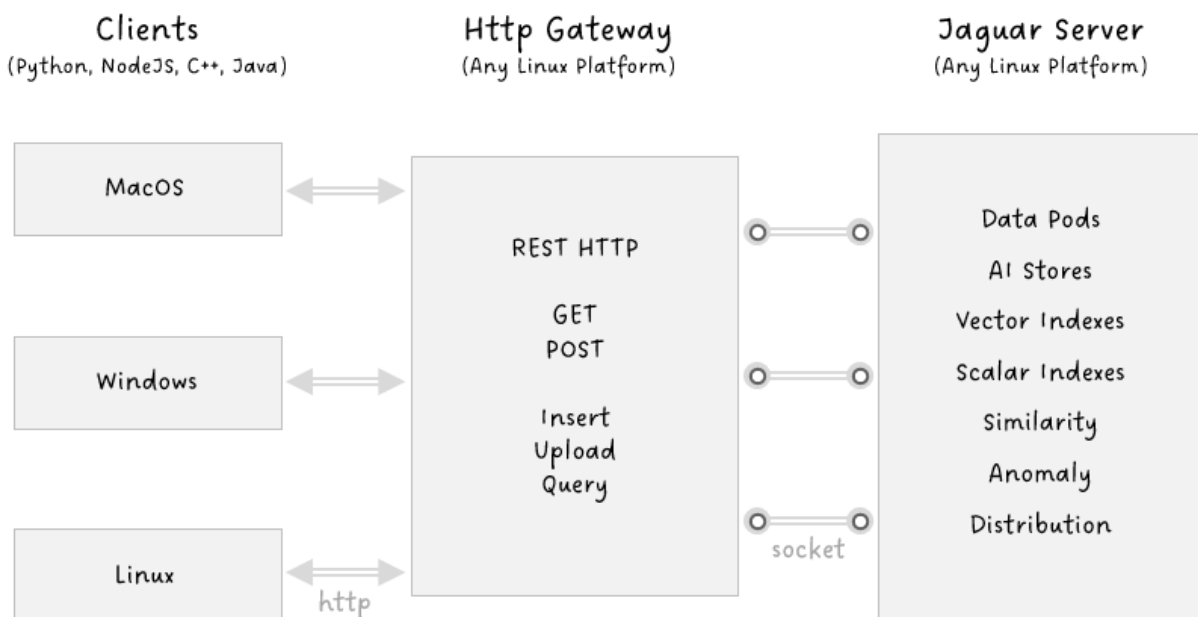
Applications have the flexibility to bypass the HTTP gateway and establish direct connections with the JaguarDB server when they are operating on the Ubuntu 22.04 platform. Additionally, libraries are provided to empower clients with the capability to communicate directly with the JaguarDB server for seamless data access and management.

Client programs that focus on AI applications should install the jaguardb-socket-client package with the pip install command. The package requires the client to be based on Ubuntu 22.0 platform as of JaguarDB release 3.3.8.

In a JaguarDB cluster, each node operates in a precise sequence of steps. Initially, JLog records incoming vector data, followed by DLog that tracks changes in vector data prepared for replication nodes, offering resilience against system or network

Clients
(Python, NodeJS, C++, Java)

Jaguar Server
(Any Linux Platform)

Insert
Select
Update
Delete
Upload
Show Info
API Keys

Data Pods
AI Stores
Vector Indexes
Scalar Indexes
Similarity
Anomaly
Distribution
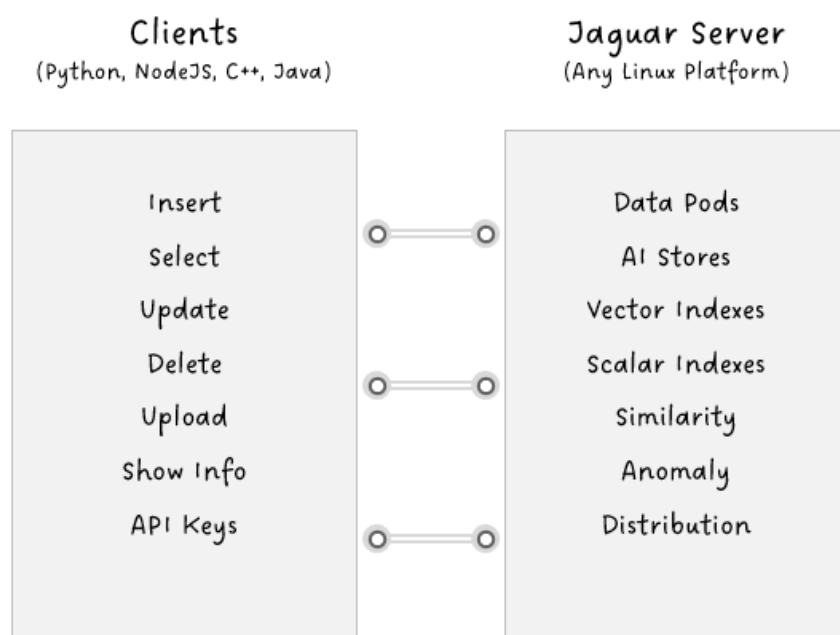
disruptions. Meanwhile, JMem, an in-memory structure, temporarily stores vector in a data store of HNSW structure, which stands for "Hierarchical Navigable Small World". The HNSW store is a data structure and algorithm used for building and searching in approximate nearest neighbor search (ANN) systems. The concept of HNSW is designed to efficiently perform similarity searches in high-dimensional spaces. HNSW stores and other data caches are frequently flushed to vector database files. Jstore encapsulates both indexed database files and their fundamental file management elements. While not depicted in the figure, JIndex corresponds to Jstore, allowing a single store object to own multiple indexes. Introducing JFile, a distinct

module, is designed for user-initiated media file uploads. JFile accommodates external user files, maintaining independence from internal files associated with stores and indexes.

Server Topology

All Jaguar server hosts are specified in the conf/cluster.conf file which contains all the IP addresses of the server hosts. Each Jaguar server maintains network communication channels to other servers for schema changes. Among the servers control messages are exchanged for synchronization of server status. Each server manages store data locally for data writes and reads.
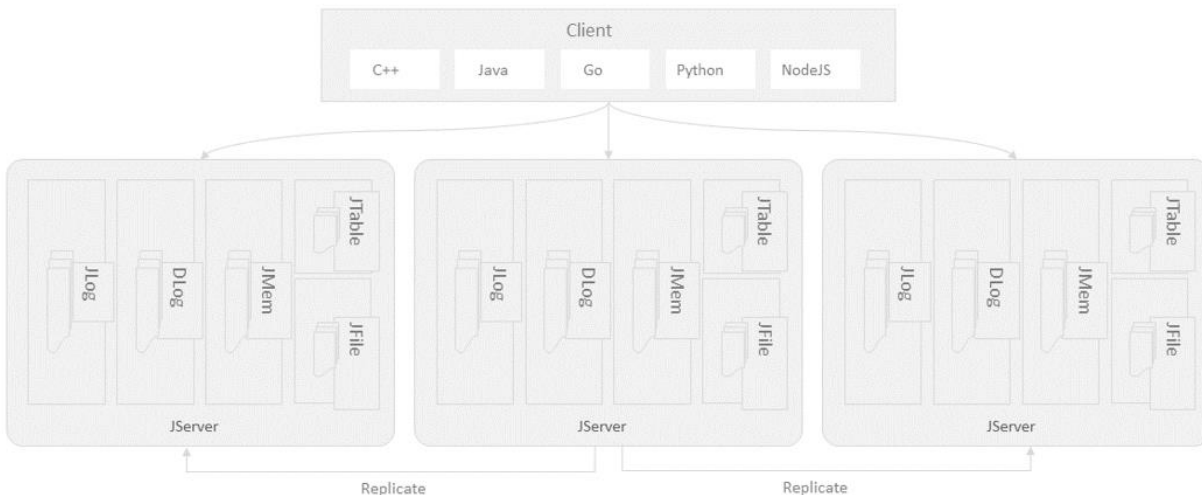
High Availability

In a multi-node deployment configuration, primary nodes assume responsibility for a specific data record and are complemented by replica nodes, forming a fault-tolerant setup. Should the primary node encounter an outage, the replicas retain an identical copy of the data record. Subsequently, when the primary node is rejuvenated and reverts to its standard operational state, the replicas initiate a data recovery process to synchronize any divergences. This data recovery encompasses both alterations in metadata and changes in raw data, encompassing a provision for maintaining up to

three replicas for each individual data record. Throughout instances of primary node downtime, the system upholds its regular functionality for data read and write operations, thereby preserving data access continuity.

Architected with resilience in mind, the system adheres to predefined replication factors for enhanced fault tolerance. For instance, with a replication factor set at two, the system demonstrates robustness by tolerating failures of up to 33% of all nodes. Similarly, a replication factor of three extends this resilience to tolerate failures of up to 66% of all nodes. It is critical to underscore the significance of proactive monitoring and prompt remediation in the event of node failures. Given that the data recovery process is inherently iterative and time-intensive, promptly restarting failed nodes is essential to streamline the catch-up process and reinstate the system's full operational capacity within an acceptable timeframe.

System Configuration

Mount noatime

File Input and Output (IO) is one of the most important performance indicators for Database. We suggest that you turn off the *access time* option for your file system. You may disable this in */etc/fstab* as *root* :

    defaults,noatime

Resource limits

Maximum Number of Open Files

Small number of maximum open files and number of processes and threads is a common problem in Linux systems. We suggest you increase the parameters by adding the following lines with root account to /etc/security/limits.conf

```
    *         hard    nofile     1000000
    *         soft    nofile     1000000
```

Maximum Number of Threads or Processes Per User
/etc/security/limits.conf:

```
    *           hard    nproc       500000
    *           soft    nproc       500000
```

Maximum Kernel Threads
/etc/sysctl.conf:
```
    kernel.threads-max = 1000000
```

Maximum Number of Process IDs
/etc/sysctl.conf:
```
    kernel.pid_max = 1000000
```

Please note the if there are config files in /etc/security/limits.d/ directory. The settings in the config files under this directory will override the settings in the /etc/security/limits.conf file. Please make sure you make the changes in the files under /etc/security/limits.d/. Please do not set nproc to an extremely high number or to "unlimited" which could cause users unable to login to the system.

Once you save the file, please reboot the system. The parameters will take effect after reboot. (If you wish not to reboot the system, please execute "sysctl -p" and close the old terminal and open a new window terminal)

Installation Verification

After you install the Jaguar Server, please make sure :
1) No other service or processes use port 8888
2) Directory *$JAGUAR_HOME/* was created
3) Following files exist and is executable:

```
    $JAGUAR_HOME/bin/jaguar.bin
    $JAGUAR_HOME/bin/jaguarstart (start local jaguar)
    $JAGUAR_HOME/bin/jaguarstop  (stop local jaguar)
```

Test Run

Test Approaches

There are two ways to interact with Jaguar servers:

1. Interaction between Jaguar Client and Server side:

Test by running the $JAGUAR_HOME/bin/jag client, typing SQL-like commands. Then the Server will respond when receiving queries.

2. APIs calls

Test by writing programs which calls Jaguar APIs to perform related data Select, Insert operations. Client API bindings include Java, C++, Python, PHP, Go, NodeJS languages.

Programming Guide

There are example programs in $JAGUAR_HOME/doc directory which can be used as a guide for developers.

Shell

$ $JAGUAR_HOME/bin/jag –u USERNAME –p PASSWORD –h HOST:PORT –d DATABASE

Example: `$ $JAGUAR_HOME/bin/jag -apikey <KEY> -h hostip:8888 -d mydb`

```
jaguardb> create store photos (
    key: zid zuid,
    value: vector v(1024, 'cosine_fraction_short,euclidean_fraction_byte')
          fname char(128) );


jaguardb> select similarity(v,'0.1,0.2,0.4',
                            'topk=10,type=euclidean_fraction_byte')
        from tvec;
```

Curl

Using the "curl" commands necessitates configuring your HTTP server with the Jaguar fwww package. Initially, you must execute a "login" request to obtain a valid token for communication with the server. It is advisable to perform a "logout" request afterward for enhanced security and efficient resource utilization. In the interim, you can execute various requests using the "query" command.

```bash
#!/bin/bash

#apikey="20231119211753587j05a208561d6e87fdb3fafd065390f5be0@000"
apikey=`cat ~/.jagrc`


echo "login ..."
r=`curl -s --request POST --url "http://192.168.1.88:8080/fwww/" \
   -d "{\"query\": \"login\", \"apikey\": \"$apikey\" }"`
echo "$r"


#r={"access_token":"906305a5c645dcf83eaf05068b63b1ee139b0","token_type":"Bearer"}
token=`echo $r|cut -d'"' -f4`
echo "token=[$token]"


echo "drop store myteststore ..."
curl -s --url "http://192.168.1.88:8080/fwww/"  --request POST \
     --header "Authorization: Bearer $token" \
     -d "{\"query\": \"drop store if exists myteststore\" }"


echo
echo "create store myteststore ..."
curl -s --url "http://192.168.1.88:8080/fwww/"  --request POST \
     --header "Authorization: Bearer $token" \
     -d "{\"query\": \"create store myteststore ( v vector(1024,
'cosine_fraction_float'), v:text char(1024), a int)\" }"


echo
```

```
echo "insert ..."

curl -s --url "http://192.168.1.88:8080/fwww/"  --request POST \

     --header "Authorization: Bearer $token" \

     -d "{\"query\": \"insert into myteststore values ( '0.1,0.2,0.3','text 1 here',
'100')\" }"


echo

echo "insert ..."

curl -s --url "http://192.168.1.88:8080/fwww/"  --request POST \

     --header "Authorization: Bearer $token" \

     -d "{\"query\": \"insert into myteststore values ( '0.8,0.1,0.2','text 2 here',
'200')\" }"


echo

echo "select similarity ..."

curl -s --url "http://192.168.1.88:8080/fwww/"  --request POST \

     --header "Authorization: Bearer $token" \

     -d "{\"query\": \"select similarity( v,
'0.6,0.2,0.3','topk=1,type=cosine_fraction_float') from myteststore\" }"


echo

echo "select similarity with filter(where) ..."

curl -s --url "http://192.168.1.88:8080/fwww/"  --request POST \

     --header "Authorization: Bearer $token" \

     -d "{\"query\": \"select similarity( v,
'0.6,0.2,0.3','topk=1,type=cosine_fraction_float') from myteststore where a >=
'100'\" }"


echo

echo "logout ..."

curl -s --url "http://192.168.1.88:8080/fwww/"  --request POST \

     --header "Authorization: Bearer $token" \

     -d "{\"query\": \"logout\" }"
```

C++/C

```
#include <JaguarAPI.h>
JaguarAPI  jdb;
jdb.connect(  host, port, userapikey, dbname );
jdb.execute(  "insert into photos values ( '0.1,0.2,0.3', 'photo1.jpg') " );
jdb.query( "select * from photo1;" );
while ( jdb.fetch() ) {
        jdb.printRow();
        char *p = jdb.getValue( "v" );
        printf("vid=%s\n", p );
        free( p );
      p = jdb.json();
      printf("JSON string=[%s]\n", p );
}
```

Java

```
System.loadLibrary("JaguarClient");
Jaguar jdb = new Jaguar();
boolean rc = jdb.connect( "127.0.0.1", 8888, "testapikey", "test");
jdb.execute("insert into tab (uid, addr) values ( 'Jill', '333 B Ave, CA' );");
jdb.query("select * from tab;");
while( jdb.fetch() ) {
      val = jdb.getValue("uid");
      m1 = jdb.getValue("m1");
      System.out.println( "uid: " + val + " m1: " + m1  );
 }
```

```
jdb.close();
```

Java JDBC

```
DataSource ds = new JaguarDataSource("127.0.0.1", "8888", "mydb");
Connection connection = ds.getConnection("testuserapikey");
Statement statement = connection.createStatement();
statement.executeUpdate("insert into tab (uid, addr) values ( 'Jill', '333 B Ave, CA' );");
Statement statement = connection.createStatement();
ResultSet rs = statement.executeQuery("select * from tab;");
String val;
String m1;
while(rs.next()) {
      val = rs.getString("uid");
      m1 = rs.getString("m1");
      System.out.println( "uid: " + val + " m1: " + m1  );
 }
rs.close();
statement.close();
```

Scala

```
import com.jaguar.jdbc.internal.jaguar._
System.loadLibrary("JaguarClient");
val jdb = new Jaguar();
val rc = jdb.connect( "127.0.0.1", "8888", "testapikey", "test", "", 0 );
jdb.execute("insert into tab (uid, addr) values ( 'Jill', '333 B Ave, CA' )");
```

```
jdb.query("select * from tab;")

while( jdb.fetch() ) {

      val u = jdb.getValue("uid");

      val m1 = jdb.getValue("m1");

      println( "uid: " + val + " m1: " + m1  );

 }

jdb.close();
```

Python

This approach takes advantage of the C++ shared library in "jaguarpy" package and makes direct access to the JaguarDB server through the socket in the package.

Make sure the environment variable PYTHONPATH points to the directory where jaguarpy.so library file exists:

```
export PYTHONPATH=$JAGUAR_HOME/lib

export LD_LIBRARY_PATH=$JAGUAR_HOME/lib
```

Then in your python program:

```
import jaguarpy

jdb = jaguarpy.Jaguar()

rc = jdb.connect( "192.168.2.200", 8888, "apikey", "dbname" )

jdb.execute("insert into tab (uid, addr) values ( 'Jill', '333 B Ave,
CA' );");

jdb.query( "select * from t1; " );

while jdb.fetch():

      jag.printRow();

      u = jdb.getValue( "uid" );
```

```
a = jdb.getValue("addr");

ds = 'uid is ' + repr(u) + '  addr is ' + repr(a)

print( ds );
```

With jaguardb-socket-client Package

This approach works similarly to the direct access, except it facilitates the installation of client packages with a simple pip package named "jaguardn-socket-client". Users can install the package on any node that runs on an Ubuntu 22.04 platform.

The package can be installed:

```
pip install -U jaguardb-socket-client
```

export LD_LIBRARY_PATH=$HOME/.local/jaguardb

export PYTHONPATH=$HOME/.local:$LD_LIBRARY_PATH

```
apikey = jag.getApikey()

jag.connect(apikey, '127.0.0.1', 8888, 'vdb' )


q = "drop store vdb.mystore"

jag.execute(q)

q = "create store vdb.mystore ( key: zid zuid, value: v vector(1024,
'cosine_fraction_float'), v:f file, v:t char(1024) )"

jag.execute(q)

q = "select similarity(v, '" + comma_sepstr + "', 'topk=1,
type=cosine_fraction_float, with_score=yes, with_text=yes') "

q += " from vdb.mystore"

jag.query(q)

jag.fetch()

jag.close()
```

With jaguardb-http-client Package

The package can be installed:

```
pip install -U jaguardb-http-client
```

As described in the "curl" commands, this approach requires the fwww http server running to handle http requests. It first requires a "login" request and finally a "logout" request. For smaller-sized commands, consider using the get() method, as GET requests are typically faster than the POST method employed by the query() function.

```
import requests, json, sys
from sentence_transformers import SentenceTransformer
from jaguardb.JaguarHttpClient import JaguarHttpClient

url = http://192.168.10.88:8080/fwww/
jag = JaguarHttpClient( url )
apikey = jag.getApikey()    # or you can use "demouser"
token = jag.login(apikey)
query = "create store vdb.mystore ( key: zid zuid, value: v vector(1024,
'cosine_fraction_float'), v:f file, v:t char(1024) )"
response = jag.get(query, token)
jag.logout(token)
```

For more information, please refer to https://github.com/fserv/jaguar-sdk web site.

PHP

To program PHP with Jaguar, please use root or sudo and copy conf/jaguar.ini to /etc/php.d directory (Centos), or to /etc/php5/mods-available (Ubuntu), or to other PHP

required directory. Also copy lib/jaguarphp.so and lib/libJaguarClient.so to proper directory.

$ php -i | grep additional   Gives directory where jaguar.ini should be copied to.

$ php -i |grep extension_dir   Gives directory where jaguarphp.so should be copied to.

Example:

```
Centos    # cp -f  conf/jaguar.ini  /etc/php.d/

Centos    # cp -f  lib/jaguarphp.so  /usr/lib64/php/modules

Ubuntu    # cp -f conf/jaguar.ini /etc/php5/mods-available/

Ubuntu    # cp -f lib/jaguarphp.so  /usr/lib/php5/20121212

# cp -f lib/libJaguarClient.so  /usr/lib
```

```php
<?php

$jdb = new Jaguar();

$jdb->connect( "192.168.2.200", 8888, "apikey",  "dbname" );

$jdb->execute("insert into tab (uid, addr) values ( 'Jill', '333 B Ave, CA' );");

$jdb->query( "select * from t1;" );

While ( jdb.fetch() ) {

        $jag->printRow();

        $u = $jdb->getValue( "uid" );

        $a = $jdb->getValue("addr");

        print( "uid=$u addr=$a\n");

}

…

?>
```

NodeJS

To use Ruby client API, make sure lib/jaguarnode.node exist in the $JAGUAR_HOME/lib directory:

var homedir=process.env.JAGUAR_HOME;

var libname = homedir + "/jaguar/lib/jaguarnode";

const jaguarnode = require(libname)

var jaguar = new jaguarnode.JagAPI();

jdb.connect("127.0.0.1", 8888, "adminapikey", "test");

jdb.execute("insert into tab (uid, addr) values ('Jill', '333 B Ave, CA');");

jdb.query("select * from t1;");

while (jdb.fetch()) {

      jdb.printRow();

      var u = jdb.getValue("uid");

      var a = jdb.getValue("addr");

      process.stdout.write("uid: " + uid + " addr: " + addr + "\n");

end

Go

To use Go language client API, please go to the src/golang directory in github.com/datajaguar/jaguardb and read the readme file. The jaguargo directory contains the interface files between C++ and golang. The script compile.sh is a program to compile jaguargo package imported by the main.go program as an example.

main.go file:

```
package main
import(
  "jaguargo/jaguargo"
  "strconv"
  "flag"
  "fmt"
```

```go
    "time"

    "os"

)


func main() {

    flag.Parse()

    ports := flag.Arg(0)

    jdb := jaguargo.New()

    fmt.Printf("connecting to jaguardb 127.0.0.1 port=%s\n", ports )

    port, err := strconv.ParseUint(ports, 0, 64 )

    if err != nil {

        fmt.Printf("error\n" )

        os.Exit(1)

    }


    jdb.Connect("127.0.0.1", uint(port), "admin_api_key", "test" )

    jdb.Execute("drop store if exists gotab123")

    jdb.Execute("create store gotab123 (key: uid char(32), value: addr char(128) )" )

    jdb.Execute("insert into gotab123 values ( 'id1001', '123 W. Washington Blvd' )")

    jdb.Execute("insert into gotab123 values ( 'id1002', '225 E. Sunshine St' )")


    jdb.Query("show databases" )

    fmt.Printf("List of databases:\n")

    for {

            rc := jdb.Fetch()

            if rc > 0 {

                jdb.PrintRow()

            } else {

                break

            }

    }

    jdb.Query("show stores" )

    fmt.Printf("List of stores:\n")

    for {
```

```
        rc := jdb.Fetch()

        if rc > 0 {

            jdb.PrintRow()

        } else {

            break

        }

    }

    time.Sleep(1*time.Second)

    jdb.Query("select * from gotab123" )

    fmt.Printf("Data in store gotab123:\n")

    for {

            rc := jdb.Fetch()

            if rc > 0 {

                jdb.PrintRow()

            } else {

                break

            }

    }

    jdb.Close()
}
```

To execute the main.go program:

```
export LD_LIBRARY_PATH=$HOME/jaguar/lib:/home/jaguar/jaguar/lib:/usr/local/gcc-7.1.0/lib64
unset GOPATH
export GO111MODULE=on
go run main.go 8888
```

Query with Index

Suppose store mystore contains key: uid and value: v1, v2, v3. If you need to query data in mystore according to a non-key column (or several columns), then you can create an index on the column(s) and query mystore by using the index. For example:

```
create index mystore_idx23 on mystore ( v2, v3 );
```

Shell

```
jaguar> select * from mystore_idx23 where v2='somevalue' and
v3='somevalue';
```

C++/C

jdb.query("select * from mystore_idx23 where v2 >= 'somevalue' ; ");

while (jdb.fetch( )) {

      jdb.printRow();

      char *p = jdb.getValue("uid");

      printf("uid=%s\n", p); free(p);

        p = jdb.json();

        printf("JSON string=[%s]\n", p);

}

Java JDBC

Statement statement = connection.createStatement();

ResultSet rs = statement.executeQuery("select * from mystore_idx23 where v2 >= 'myvalue';");

String val;

String m1;

while(rs.next()) {

    val = rs.getString("uid");

```
        m1 = rs.getString("m1");

        System.out.println( "uid: " + val + " m1: " + m1  );

    }

  rs.close();

statement.close();
```

Client API Reference

The following methods are supported for C++, Java, Scala, Python, PHP, NodeJS and other API calls:

1. bool connect(String host, int port, String userapikey, String db)
   Connects to server. Returns True for success, False for failure.

2. bool execute(String command)
   Execute a data modification command string such as create store, drop store, insert commands. The command must not contain a "select" query string where multiple rows may be expected. Multiple statements, delimited by the ';' character, can be placed in the execute command. For example, execute("insert into t123 values ('1', '2'); insert into t345 values ('3', '5'); update t888 set v='3' where uid='234'; delete from t69 where uid='333';"). Allowed commands include insert, update, delete, alter, drop, and truncate.

3. bool query(String query)
   Select data from server. This is where the "select" statement should be used.

4. Bool fetch()
   Return result data to the client. With a while loop around this call, you can obtain the selected result data row by row. When there is no more data, the fetch() call returns false.

5. void printRow()
   Print out row data on standard output.

6. void close()
   Closes the connection to server and frees up relevant memory resources.

7. String getDatabase()
   Returns the database name of current client session.

8. bool hasError()
    Tests if there is error from the query command.

9. String error()
    If hasError() is true, this call returns the error string.

10. String getLastZuid()
    This function returns the UUID string of the last insert operation if the store has zuid type in its first column and the column is a key field. If the first column is not a key field, then this function returns empty string.

11. String getNthValue(int col)
    Returns the value of the N-th column (starting from 1) in the current row (inside the reply while loop).

12. String getValue(String columnName)
    Returns the value of a column of name columnName. For example, is "uid" is the column name of a store, then getValue("uid") returns the value of uid column in the current row.

13. String getMessage( )
    Return the output data in the current row. Sometimes the current row data does not have any column structure, with only a raw message. For example, "desc store;" will output a text message describing the format of a store. In such cases, getMessage() should be called.

14. String json()

    Return the JSON string for a data record or a group of records.

15. long getLong(String columnName)
    If the column is known to be long integer type, this method returns the long integer value.

16. double getFloat(String columnName)
    If the column is known to be numerical double type, this method returns the double value.

17. int getColumnCount()
    Returns the number of columns in current row.

18. String getColumnName(int col)
    Returns the string name of the col-th column (starting from 1).

19. int getColumnType(int col)
Return the numeric column type of col-th column (per JDBC definition)

20. String getColumnTypeName(int col)
Return the string column type of col-th column (per JDBC definition)

21. String getstoreName(int col)
Return the store name of col-th column

CURL API

Login to get a valid session token:

```
curl --request POST --url "http://<IP>:8080/fwww/" \
    -d "{\"query\": \"login\", \"apikey\": \"$myapikey\" }"`
```

Make any requests:

```
curl --url "http://<IP>:8080/fwww/"  --request POST \
    --header "Authorization: Bearer $token" \
    -d "{\"query\": \"any SQL-like query here\" }"
```

Logout of the session:

```
curl  --url "http://<IP>:8080/fwww/"  --request POST \
   --header "Authorization: Bearer $token" \
   -d "{\"query\": \"logout\" }"
```

Python REST API

```
import requests, json, sys
from sentence_transformers import SentenceTransformer
from jaguardb.JaguarHttpClient import JaguarHttpClient
```

Create Client Object:

```
jag = JaguarHttpClient( url )
```

Read in ApiKey:

If API key is not provided by the user, the following method can be called to find available API key on the current host.

```
apikey = jag.getApikey()
```

Login:

The following function call authenticate the API key and returns a valid session token:

```
token = jag.login(apikey)
```

Requests:

With the session token obtained from the "login" request, any requests can be made to the server:

```
    q = "any create/insert/update/delete/select commands here"
    response = jag.query(q, token)
```

Upload File:

```
jag.postFile(token, fpath, position )
```

Before executing an "insert" statement in which there are files to be uploaded to the Jaguar server, this command first reads and sends the file to the server. The parameter "position" specifies the position of the file column in the insert values, starting from 1. Then in the insert command:

```
q = "insert into vdb.store values ('...', '...', '…' )"
jag.post(q, token, True)  // True tells there are files to be transferred
```

Logout:

Remember to logout for security and resource reuse:

```
jag.logout(token)
```

LangChain Integration

LangChain is a framework for building LLM-powered applications. It helps developers chain together interoperable components and third-party integrations to simplify AI application development — all while future-proofing decisions as the underlying technology evolves. The JaguarDB vector store is integrated with the LangChain development frame. The main URL for the LangChain github is

```
https://github.com/langchain-ai/langchain
```

and the Jaguar vector store is under:

```
https://github.com/langchain-ai/langchain
```

```
->libs/community/langchain_community/vectorstores/jaguar.py
```

The documentation is under:

```
https://github.com/langchain-ai/langchain

-> docs/docs/integrations/vectorstores/jaguar.ipynb
```

LLamaIndex Interation

LlamaIndex (GPT Index) is a data framework for your LLM application. Building with LlamaIndex typically involves working with LlamaIndex core and a chosen set of integrations (or plugins). The main github repository is at:

```
https://github.com/run-llama/llama_index
```

The Jaguar vector store is at:

```
llama-index-integrations/vector_stores/llama-index-vector-stores-
jaguar/llama_index/vector_stores/jaguar/base.py
```

NodeJS API

JaguarDB provides API to Node.js developers. The JaguarNodeClient class and methods can be installed with the following command on a Linux system:

```
npm install jaguardb-node-client
```

Developers are recommended to read the documents in github.com/fserv/jaguar-sdk javascript directory.

Operation

Remote Backup

Setup on the first Jaguar server host

The data stored in all the servers of Jaguar cluster can be backed up in a remote server (such as a high-capacity storage server) frequently. In conf/server.conf, you can assign values to the REMOTE_BACKUP_SERVER and REMOTE_BACKUP_INTERVAL parameters to enable this feature. REMOTE_BACKUP_SERVER should point to the IP address of the remote backup server, and REMOTE_BACKUP_INTERVAL is the time interval (in seconds) specifying how often the backup is performed. This configuration needs to be completed on the first jaguar server host only. It is not necessary to set it up on other jaguar server hosts. The file conf/syncpass.txt (chmod 600 as user jaguar) should just contain the password (single word) of jaguar user to connect to the remote backup server host. This password can be different from jaguar's system account password.

```
File conf/syncpass.txt:
mypassword888
```

Setup on the remote backup server host

On the remote backup server, rsync daemon should be setup correctly. The config file /etc/rsyncd.conf should have the following information:

```
uid = jaguar
gid = jaguar
use chroot = yes
max connections = 1000
pid file = /var/run/rsyncd.pid
log file = /var/log/rsyncd.log
```

```
    exclude = lost+found/

    transfer logging = yes

    timeout = 900

    ignore nonreadable = yes

    dont compress   = *.gz *.tgz *.zip *.z *.Z *.rpm *.deb *.bz2

    read only = false

    write only = false


    [jaguardata]

            path = /home/jaguar/jaguarbackup

            comment = Jaguar repository (requires authentication)

            auth users = jaguar

            strict modes = false

            secrets file = /etc/rsyncd.secrets
```

where "`[jaguardata]`" must be kept exactly as it is shown above but "path =
/home/jaguar/jaguarbackup" can be any directory you wish to use. This directory should be
owned by 'jaguar' user.

```
    # mkdir -p /home/jaguar/jaguarbackup

    # chown -R jaguar.jaguar /home/jaguar/jaguarbackup
```

In the file /etc/rsyncd.secrets (chmod 600 as root), you should have the password for jaguar
user (username:password format):

```
        jaguar:mypassword888
```

In /etc/rsyncd.secrets there can be many lines specifying username and password for rsync
daemon server to authenticate. If someuid takes the value of jaguar, i.e., rsync daemon will be
run as user jaguar. The password 'mypassword888' is just an example. You should use your
own password and is the same as the one in Jaguar server's `conf/syncpass.txt.`

Restart the rsync daemon server on this host after you have made the changes. On
CentOS/Redhat systems, the command to restart rsync daemon server as root is:

```
# systemctl restart rsyncd
```

Data Types

Currently Jaguar supports these data types:

1. Character string
   char(length) -- it is a fixed length character string in key columns. It is a variable length string in the value columns. It is same as varchar(length).

2. Boolean
   boolean -- one byte integer field containing a single digit

3. Integer
   int or integer - integer between -9999999999 and +9999999999

4. Big integer
   bigint -- integer between -9999999999999999999 and +9999999999999999999

5. Small integer
   smallint -- integer between -99999 and +99999

6. Tiny integer
   tinyint – integer between -999 and +999

7. Medium integer
   mediumint – integer between -9999999 and +9999999

8. Float
   float(L,d) -- a float decimal number with total of L digits and d number of digits after the decimal point.

9. Double
   double(L,d) -- similar to float except in internal representation and calculation, it is treated as double precision float number.

   numeric(L,d) – same as double(L,d)
   decimal(L,d) -- same as double(L,d)

10. LongDouble
   longdouble(L,d) -- similar to double except in internal representation and calculation, it is treated as a long double column.

11. DateTime
   datetime – a 16 digits time value in terms of microseconds. When a time data is loaded or inserted into Jaguar, the following format must be used:

   YYYY-MM-DD hh:mm:ss[.uuuuuu] [+HH:MM]
   YYYY-MM-DD hh:mm:ss[.uuuuuu] [-HH:MM]

   Where YYYY is the 4-digit year symbol, such as 2025
   MM is the month (1-12), such as 10
   DD is the date in 1-31, such 04
   hh:mm:ss is hour:minute:seconds such as 02:23:21
   .uuuuuu is optional fractional seconds (or microseconds)

   +HH:MM and -HH:MM are optional time zone difference from GMT standard time. If no time zone information is given, then the input string is taken as local time of the client. If the client just wants to insert local time string, then the time zone string is not necessary. The time zone info is only used when the client wants to insert time string of another time zone.

   Example:

   From California, USA:

   ```
   insert into sa (uid, sttime) values (12,'2014-11-23 16:32:21 -08:00' );
   insert into sd (devid, ltime) values ( 1232, '2015-10-23 13:32:21.234019' );
   select * from sales where sdate > '2014-12-10 03:12:23';
   ```

12. DateTimeNano
   datetimenano – similar to datetime except this has granularity of nanoseconds. When a datetimenano column is loaded or inserted into Jaguar, the following format must be used:

   ```
   YYYY-MM-DD hh:mm:ss[.nnnnnnnnn] [+HH:MM]
   YYYY-MM-DD hh:mm:ss[.nnnnnnnnn] [-HH:MM]
   ```

13. DateTimeSec

   datetimesec – similar to datetime except this has granularity of seconds. When a datetimesec column is loaded or inserted into Jaguar, the following format must be used:

```
YYYY-MM-DD hh:mm:ss [+HH:MM]

YYYY-MM-DD hh:mm:ss [-HH:MM]

insert into sd (devid, dtcol) values ( 1232, '2022-10-23 13:32:21' );
select * from sales where dtcol > '2014-12-10 03:12:23';
```

14. DateTimeMill

   datetimemill – similar to datetime except this has granularity of milliseconds. When a datetimemill column is loaded or inserted into Jaguar, the following format must be used:

```
YYYY-MM-DD hh:mm:ss[.nnn] [+HH:MM]

YYYY-MM-DD hh:mm:ss[.nnn] [-HH:MM]

insert into sd (devid, dtcol) values ( 1232, '2022-10-23 13:32:21.123' );
select * from sales where dtcol > '2014-12-10 03:12:23.123';
```

15. Date

   The date type has input and output format: YYYY-MM-DD
   Example:

```
insert into sales (uid, datecol) values (1234, '2015-03-12');
select * from sales where datecol='2015-12-23';
```

16. Time

   time – type for tracking hour, minute, second, and microsecond. The input format of time column is:

```
HH:MM:SS[.uuuuuu] -- where uuuuuu represents microseconds
```

17. TimeNano

   timenano --- type for tracking hour, minute, second, and nanosecond. The input format of timenano column is:

```
HH:MM:SS[.nnnnnnnnn] -- where nnnnnnnnn represents nanoseconds
```

18. Timestamp
   timestamp -- same as datetimestamp with precision of microseconds. Both can take input in 'yyyy-mm-dd HH:MM:SS.123456 HH:MM' format or simply a number representing microseconds since the epoch (1 January 1970 00:00:00), for example

1482000884000000. The difference between timestamp and datatime is that during insertion of this column data, if no value is provided by the user, the current local time with precision of microseconds is automatically generated and inserted into the store for the timestamp column.

19. TimestampNano

    timestampnano -- same as datetimestampnano with precision of nanoseconds. Both can take input in 'yyyy-mm-dd HH:MM:SS.123456789 HH:MM' format. The difference is that during insertion of this column data, if no value is provided by the user, the current local time with precision of nanoseconds is automatically generated and inserted into the store for the timestampnano column.

20. TimestampSec

    timestampsec -- same as datetimestampsec with precision of seconds. Both can take input in 'yyyy-mm-dd HH:MM:SS HH:MM' format. The difference is that during insertion of this column data, if no value is provided by the user, the current local time with precision of seconds is automatically generated and inserted into the store for the timestampsec column.

21. TimestampMill

    timestampmill -- same as datetimestampmill with precision of milliseconds. Both can take input in 'yyyy-mm-dd HH:MM:SS[.nnn] HH:MM' format. The difference is that during insertion of this column data, if no value is provided by the user, the current local time with precision of milliseconds is automatically generated and inserted into the store for the timestampmill column.

22. Real

    Real – the data type is same as a double(38,8), double of total 38 digits and 8 digits after the decimal point.

23. Text

    Text -- is the same as char(1024)

24. TinyText

    TinyText -- is the same as char(256)

25. MediumText

    MediumText -- is the same as char(2048)

26. LongText

    LongText -- is the same as char(10240)

27. Blob

Blob -- is the same as char(1024)

28. TinyBlob
    TinyBlob -- is the same as char(256)

29. MediumBlob
    MediumBlob -- is the same as char(2048)

30. LongBlob
    LongBlob -- is the same as char(10240)

31. String
    String – is the same as char(64)

32. Varchar
    Varchar(N) – is same as char(N)

33. Bit
    Bit – is one byte column, with value of 1 or 0

34. Eum

    COLUMN enum ('val1', 'val2', 'val3', …) -- A column can take certain values only

35. File
    File – is used to store any file (photo, image, audio, video, doc, ppt, pdf, etc). It has no limit in size.

36. Spatial Data Types
    Please refer the Spatial Data Management chapter in this manual.

37. Range

    Range(datetime)   -- a range of datetime with begin and end data
                        Format: "YYYY-MM-DD HH:MM:SS[.nnnnnn]"
    Range(datetimesec)   -- a range of datetime with begin and end data
                        Format: "YYYY-MM-DD HH:MM:SS"
    Range(datetimemill)   -- a range of datetime with begin and end data
                        Format: "YYYY-MM-DD HH:MM:SS[.nnn]"
    Range(datetimenano)   -- a range of datetime with begin and end data
                        Format: "YYYY-MM-DD HH:MM:SS.[nnnnnnnnn]"
    Range(date)   -- a range of date with begin and end data
                        Format: "YYYY-MM-DD", example "2018-09-12"

Range(time)   -- a range of time with begin and end

                        Format: "HH:MM:SS", example "13:21:01"

Range(double)   -- a range of double numbers with begin and end

Range(longdouble)   -- a range of longdouble numbers with begin and end

Range(float)   -- a range of float numbers with begin and end

Range(bigint)   -- a range of bigint numbers with begin and end

Range(int)   -- a range of integer numbers with begin and end

Range(medint)   -- a range of medint numbers with begin and end

Range(smallint)   -- a range of smallint numbers with begin and end

Range(tinyint)   -- a range of tinyint numbers with begin and end

38. ZUID

COLUMN zuid   -- COLUMN will be a string field with a 32-byte string that uniquely identifies the record in the store. The value of the UUID column must not be provided by the user in the insert statement since it will be generated automatically by the system during the insert operation. If the "key" keyword is not provided, the system automatically provides the default "zid zuid" leading column as the primary key for database records.

Examples:

Insert into a values (123, range('2010-01-01 00:00:00', '2020-12-31 23:59:59'));

Insert into b values (123, range('2010-01-01', '2020-12-31'));

Insert into c values (123, range('00:00:00', '13:00:00'));

Insert into d values ('abc', range(100, 500));

Insert into e values ('abc', range(2.34, 100.918));

Select * from e where within(r, range(10, 500));

Select a, b, r:begin, r:end from e where r:begin >= 300 and r:end <= 1000;

Functions within(), contain(), cover(), converedby(), intersect(), disjoint() are supported for the range data types.

Default Values

Any column in a store can take a one-byte default value. The timestamp and datetime columns can have default value of CURRENT_TIMESTAMP. Also upon update of a row, its

timestamp column can be automatically updated by entering "ON UPDATE CURRENT_TIMESTAMP" . For example:

```
Create store tab123 (

    Key:  zid zuid,

    Value:

        a int default  '0',

        b char(16) default 'b',

        bitv   bit default b'1',

        bitm  bit default b'0',

        tm1 timestamp DEFAULT CURRENT_TIMESTAMP,

        tm2  timestamp DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP ,

        tm3  timestamp ON UPDATE CURRENT_TIMESTAMP ,

        speed enum ('low', 'med', 'high' ) default 'med'

    );
```

Data Type Mapping Between Jaguar and Java

The following store specifies the mapping between Jaguar data types and Java data types:

Jaguar Type	Format	Java Type
bool	bool	boolean
char	char(length)	java.lang.String
char	char(length)	byte[]
int	int	int
smallint	smallint	int
tinyint	tinyint	int
mediumint	mediumint	int
bigint	bigint	long
double	double(m,n)	double
float	float(m,n)	float
timestamp	timestamp (in milliseconds)	java.util.Date
datetime	datetime (in milleseconds)	java.util.Date
datetimenano	datatimenano (in microseconds)	Java.sql.Timestamp
time	time	SimpleDateFormat
timenano	timenano	SimpleDateFormat

zuid	zuid	java.lang.String
vector		

Jaguar Functions

Jaguar supports a number of built-in functions, which can be operated on one or multiple columns from select statements or join statements. The following description illustrates how to use Jaguar functions.

Syntax:

```
SELECT FUNC( EXPR(COL) ) from store [WHERE] [LIMIT];
```

EXPR(COL):

Numeric columns: columns with arithmetic operation

+ addition

– subtraction)

* multiplication)

/ division

% modulo

^ power (exponential)

String columns: Concatenation of columns or string constants

string column + string column

string column + string constant

string constant + string column

string constant + string constant

string constant: 'some string'

FUNC(EXPR(COL)):

similarity(COL, '<VECTOR>', 'KEY_STR>')   -- find top K similar items

vector(COL, 'KEY_STR') -- find the vector coordinates for record(s)

```
min( EXPR(COL) )     -- minimum value of column expression

max( EXPR(COL) )     -- maximum value of column expression

avg( EXPR(COL) )     -- average value of column expression

sum( EXPR(COL) )     -- sum of column expression

count(1)             -- count number of rows

stddev( EXPR(COL) ) -- standard deviation of column expression

first( EXPR(COL) )  -- first value of column expression

last( EXPR(COL) )   -- last value of column expression

abs( EXPR(COL) )     -- absolute value of column expression

acos( EXPR(COL) )   -- arc cosine function of column expression

asin( EXPR(COL) )   -- arc sine function of column expression

ceil( EXPR(COL) )   -- smallest integral value not less than column expression

cos( EXPR(COL) )     -- cosine value of column expression

cot( EXPR(COL) )     -- inverse of tangent value of column expression

floor( EXPR(COL) )  -- largest integral value not greater than column expression

log2( EXPR(COL) )    -- base-2 logarithmic function of column expression

log10( EXPR(COL) )  -- base-10 logarithmic function of column expression

log( EXPR(COL) )     -- natural logarithmic function of column expression

ln( EXPR(COL) )      -- natural logarithmic function of column expression

mod( EXPR(COL), EXPR(COL) )  -- modulo value of first over second column expression

pow( EXPR(COL), EXPR(COL) )  -- power function of first to second column expression

radians( EXPR(COL) )   -- convert degrees to radian

degrees( EXPR(COL) )   -- convert radians to degrees

sin( EXPR(COL) )     -- sine function of column expression

sqrt( EXPR(COL) )   -- square root function of column expression

tan( EXPR(COL) )     -- tangent function of column expression

substr( EXPR(COL), start, length )  -- sub string of column expression

substr( EXPR(COL), start, length, 'UTF8' )  -- sub string of UTF8 encoded string
```

substring(EXPR(COL), start, length) same as substr() above

diff(EXPR(COL), EXPR(COL)) Levenshtein distance (edit distance) between two strings

diff(COL, 'stringconstant') Levenshtein distance (edit distance) between two strings

upper(EXPR(COL)) -- upper case string of column expression

lower(EXPR(COL)) -- lower case string of column expression

ltrim(EXPR(COL)) -- remove leading white spaces of string column expression

rtrim(EXPR(COL)) -- remove trailing white spaces of string column expression

trim(EXPR(COL))   -- remove leading and trailing white spaces of string column

length(EXPR(COL)) -- length of string column expression

second(TIMECOL)   -- value of second in a datetime column

minute(TIMECOL)   -- value of minute in a datetime column

hour(TIMECOL)   -- value of hour in a datetime column

date(TIMECOL)   -- value of date in a datetime column

month(TIMECOL)   -- value of month in a datetime column

year(TIMECOL)   -- value of year in a datetime column

datediff(type, BEGIN_TIMECOL, END_TIMECOL) -- difference of two datetime columns

> type: second (difference in seconds)
>
> type: minute (difference in minutes)
>
> type: hour   (difference in hours)
>
> type: day   (difference in days)
>
> type: month   (difference in months)
>
> type: year   (difference in years)

The result is the END_TIMECOL – BEGIN_TIMECOL.

dayofmonth(TIMECOL) -- the day of the month in a datetime column (1-31)

dayofweek(TIMECOL)   -- the day of the week in a datetime column (0-6)

dayofyear(TIMECOL)   -- day of the year in a datetime column (1-366)

curdate()   -- current date (yyyy-mm-dd) in client's local time

curtime()   -- current time (hh:mm:ss) in client's local time

now()   -- current date and time (yyyy-dd-dd hh:mm:ss) in client's local time

uuidtime(ZUIDCOL)   -- time string of a ZUID column with precision of microseconds.

Example:

```
   select similarity(v, '0.1,0.2,0.3', 'topk=5,type=cosine_fraction_short')
from tvec;

   select vector(vc, 'type=cosine_fraction_short') from tvec where
zid='1234';

   select sum(amt) as amt_sum from sales limit 3;

   select cos(lat), sin(lon) from map limit 3;

   select tan(lat+sin(lon)) as t, cot(lat^2+lon^2) as c from map;

   select uid, uid+addr, length(uid+addr)  from user limit 3;

   select price/2.0 + 1.25 as newp, lead*1.25 - 0.3 as newd from plan;


   select curdate() cd, curtime() ct, now() nw from _SYS_;

   select dayofweek("2021-09-29 03:03:12") v;
```

The _SYS_ is a special store that has only one column with only one character. It is used to query system related data such as current time and date on the server, which are not related to any other store structures. If a select query does not point to a store name, the client side will run the locally and returns the result to the user, without consulting the server. As shown in the last example of the above set of examples, the client will just compute the day of week from a constant string and return the day number to the user.

Jaguar SQL Statements

The commands and SQL statements supported by JaguarDB can be shown by the help command in the interactive shell jql program:

```
jaguar:test> help;
You can enter the following commands (ending with semicolon):


help admin        (how to for admin account)
```

```
help use          (how to use databases)

help desc         (how to describe stores)

help show         (how to show stores)

help create       (how to create stores)

help insert       (how to insert data)

help load         (how to load data from client host)

help copy         (how to copy data from server host)

help select       (how to select data)

help update       (how to update data)

help delete       (how to delete data)

help drop         (how to drop a store completely)

help alter        (how to alter a store and rename a key column)

help truncate     (how to truncate a store)

help func         (how to call functions in select)

help spool        (how to write output data to a file)
```

Please note that in a query command, keywords (such as create, store, select, where) can only be separated by blank spaces, '\t', '\r', '\n' characters. Other non-printable characters are not allowed and may cause parsing errors when executing the query.

Admin commands

These commands should be executed by the "admin" account to manage user accounts and databases.

```
create tenant  TENANANT; -- create a new tenant or business unit

createdb DBNAME;

dropdb DBNAME;

makeapikey;    -- The command will create a valid API key string

createuser <APIKEY> <EMAIL> <LEVEL> <TENANT>  -- APIKEY is the string created
from the above makeapikey command; EMAIL is the email address of a new user;
LEVEL is the plan or type of account for the user; TENANT is the name of a
tenant or business unit. If the tenant does not exist, it will be created.
```

```
dropuser apikey;

showusers;


Example:

    createdb mydb;

    dropdb mydb;

    makeapikey;

    createuser testapikey me@emailcom 3000 tenant1;

    dropuser testapikey;
```

Grant command

After admin has created a user account, permissions of the user should be granted by the admin. The grant command can be used in the following manner:

jaguar:test> help grant;

jaguar> grant all on all to user;

jaguar> grant PERM1, PERM2, ... PERM on DB.TAB.COL to user;

jaguar> grant PERM on DB.TAB.* to user;

jaguar> grant PERM on DB.TAB to user;

jaguar> grant PERM on DB to user;

jaguar> grant PERM on all to user;

jaguar> grant select on DB.TAB.COL to user [where TAB.COL1 > NNN and TAB.COL2 < MMM;

Only the admin account can issue this command.

PERM is one of: all/create/insert/select/update/delete/alter/truncate

All means all permissions.

The where statement, if provided, will be used to filter rows in select and join.

Example:

jaguar> grant all on all to user123;

jaguar> grant all on mydb.tab123 to user123;

jaguar> grant select on mydb.tab123.* to user123;

jaguar> grant select on mydb.tab123.col2 to user3 where tab123.col4>100;

jaguar> grant delete, update on mydb.tab123.col4 to user1;

Revoke command

Permissions of a user can be revoked with the following commands:

jaguar:test> help revoke;

jaguar> revoke al on all from user;

jaguar> revoke PERM1, PERM2, ... PERM on DB.TAB.COL from user;

jaguar> revoke PERM on DB.TAB.* from user;

jaguar> revoke PERM on DB.TAB from user;

jaguar> revoke PERM on DB from user;

jaguar> revoke PERM on all from user;

Only the admin account can issue this command.

PERM is one of: all/create/insert/select/update/delete/alter/truncate

All means all permissions. The permission to be revoked must exist already.

Example:

jaguar> revoke all on all from user123;

jaguar> revoke all on mydb.tab123 from user123;

jaguar> revoke select on mydb.tab123.* from user123;

jaguar> revoke select, update on mydb.tab123.col2 from user3;

jaguar> revoke update, delete on mydb.tab123.col4 from user1;

Describe command

Describe a store or index:

```
desc store;

desc INDEX;
```

Example:

```
desc usertab;

desc db.store.addr_index;
```

Show command

Show information about database system:

```
show databases    (display all databases in the system)

show stores       (display all stores in current database)

show indexes      (display all indexes in current database)

show currentdb    (display current database being used)

show task         (display all active tasks)

show indexes from/in store  (display all indexes of a store in currently
selected database)

show server version  (display Jaguar server version)

show client version  (display Jaguar client version)

show user            (display username of current session)

show cluster         (display clusters and nodes in each cluster)
```

Example:

```
show databases;
```

```
show stores;

show indexes from mystore;

show indexes;

show task;
```

Create command

Commands for creating store and index:

```
create store store3 ( key: KEY TYPE(size), ..., value: VALUE
TYPE(size), ...  );


create store store4 ( COL1 TYPE(size), COL2 TYPE(srid:ID,metrics:M), ...  );


create index INDEXNAEME on store(COL1, COL2, ...[, value: COL,COL]);
create index INDEXNAEME on store(key: COL1, COL2, ...[, value: COL,COL]);


Example:
   create store photos ( key: zid zuid, value: vector v(1024, '<SPECS>')
                         fname char(128), );


   create store user ( key: name char(32),
                      value: age int, address char(128), rdate date );
   create store sales ( key: name char(32), stime datetime,
                       value: author char(32) );
   create store sales ( key asc: id bigint, stime datetime,
                       value: member char(32) );
   create store users ( name char(32), age int, address char(128) );


   create index addr_index on user( address );
   create index addr_index on user( address, value: zipcode );
   create index addr_index on user( key: address, value: zipcode, city );
   create store media ( key: uid int, value: audio file, video file );
```

```
    create store ls( key: id int, value: s linestring(srid:4326,metrics:5) );

    create store cirm ( key: a int, value: c circle(metrics:2), d int );

    create store if mmetrics ( key: a int, value: pt point(srid:4326,
metrics:3), b int );
```

In creating store, if there is no key specified, an ZUID column is automatically added as a unique key to the store with the name "zid". If the ZUID column is the key, then no other columns can be a key.

When creating an index, you can add several value columns which will not be used as a key column in the index. It is purely for easy data access without going to the main store for retrieving the value columns. Creating an index from a store which has data already may take some time to complete, but it will be faster than the initial time spent on inserting the store data.

If the column type is geometric or geologic, the default value of SRID is zero. Metrics specifies the number of metrics associated with each point of raster shape or with a vector shape. There can be multiple metrics corresponding to each location point in a raster geometry.

In the command to create a vector store to store vectors and related data:

```
create store photos ( key: zid zuid, value: vector v(1024, '<KEYDEFS>')
                               fname char(128) );
```

The field 'zid' would store the auto-generated ZeroMove ZUID for each record. The column 'v' will be a vector field storing an auto-generated numeric ID for the vector. The number 1024 means the dimension of the vectors will be 1024. The string <KEYDEFS> is one or more key strings, each specifying the distance type, input value type, and storage quantization type.

For example:

```
create store photos (
    key: zid zuid,
```

```
      value: vector v(1024, 'cosine_fraction_short,euclidean_fraction_byte')

              fname char(128) );
```

The above command instructs that the database will be creating a vector store for cosine similarity search, with data input type of fractional numbers, storage quantization type of 16-bit integer. In addition, a vector store is also created for Euclidean distance type, fractional input values, and quantization type of 8-bit integer. In later searches of similarity, users can query top K nearest neighbors of a query vector by cosine or Euclidean distance measures.

Insert SQL Commands

```
insert into store (col1, col2, col3, ...) values ( 'va1', 'val2',
intval, ... );

insert into store values ( k1, k2, 'va1', 'val2', intval, ... );

insert into TAB1 select TAB2.col1, TAB2.col2, ... from TAB2 [WHERE] [LIMIT];

insert into TAB1 (TAB1.col1, TAB1.col2, ...) select TAB2.col1, TAB2.col2, ...
from TAB2 [WHERE] [LIMIT];


Example:

   insert into photos values ( '0.1, 0.2, 0.3, -0.1', 'photo1.jpg');

   insert into user ( fname, lname, age ) values ( 'David', 'Doe', 30 );

   insert into user ( fname, lname, age, addr ) values ( 'Larry', 'Lee', 40,
'123 North Ave., CA' );

   insert into member ( name, datecol ) values ( 'LarryK', '2015-03-21' );

   insert into member ( name, timecol ) values ( 'DennyC', '2015-12-23
12:32:30.022012 +08:30' );

   insert into t1 select * from t2 where t2.key1=1000;

   insert into t1 (t1.k1, t1.k2, t1.c2) select t2.k1, t2.c2, t2.c4 from t2
where t2.k1=1000;

   insert into media values ( 100, '/tmp/myaudio.aud', '/tmp/muvideo.mov');

   insert into mmetrics values ( 110, point( 0.2 0.3 'A' 'B' 'C' ), 234 );

   insert into mmetrics values ( 220, point( 0.2 0.3 '10' '00' '30' ), 43 );

   insert into cirm values ( 100, circle( 22 33 100 'PARK' 'tower' ), 209 );
```

```
   insert into cirm values ( 200, circle( 24 31 100 'SCHL' 'bank' ), 258 );
```

For the column that is a vector, the single quotes can be used to quote the vector components. More than one vector columns can be added in a store, allowing for multiple types embedding or feature vectors to be stored for a data item. Metrics data must be enclosed with single quotes or double quotes. The number of metrics data must be less than or equal to the number of metrics defined when creating the store with the columns that have metrics fields. Each metric is a string that has a length less than or equal to 8 characters or numbers. Metrics normally are used as tags or attributes describing a location or a shape. During store creation, the number of metrics can be as large as desired.

If there is a column of type "zuid", then its value must not be listed in the insert command. The database server will automatically generate a unique string (32 bytes) for the column and insert the whole record into database.

For datetime, datetimenano, timestamp fields, if no time zone information is provided, the input is considered from the client's local time zone.

Load command

Loading data in a file into database:

```
load /path/input.txt into store [columns terminated by C] [lines terminated
by N] [quote terminated by Q];
(Instructions inside [ ] are optional. /path/input.txt is located on client
host.)
Default values:
   columns terminated by: ','
   lines terminated by: '\n'
   column values can be quoted by singe quote (') character.


Example:
   load /tmp/input.txt into user columns terminated by ',';
```

```
The above load command can load a CSV file into the database.
```

Select SQL command

Data can be selected in various ways from the database:

```
SELECT  similarity(VECCOL, 'QUERY_VEC',
'topk=K,type=SPEC,with_score=yes') from TAB;

SELECT  vector(VECCOL, 'type=SPEC') from TAB WHERE …;

(SELECT) from store [WHERE] [GROUP BY] [ORDER BY] [LIMIT] [TIMEOUT N];

(SELECT) from INDEX [WHERE] [GROUP BY] [ORDER BY] [LIMIT] [TIMEOUT N];

select * from store;

select * from store limit N;

select * from store limit S,N;

select COL1, COL2, ... from store;

select COL1, COL2, ... from store limit N;

select COL1, COL2, ... from store limit N;

select COL1, COL2, ... from store where KEY='...' or KEY='...' and ( ... ) ;

select COL1, COL2, ... from store where (. . .) or ( ... and ... );

select COL1, COL2, ... from store where KEY='abc' and KEY2 like 'abc%';

select * from store where KEY='abc' and KEY2 match 'abc.*z';

select COL1, COL2, ... from store where KEY='key88' and VAL1 between m and n;

select COL1 as col1label, COL2 col2label, ... from store;

select count(*) from store;

select min(COL1), avg(COL3) as avg3, sum(COL4) sum4, count(1) from store;

select FUNC(COL1) fc1, FUNC(COL2) as x from store timeout 100;
```

If no limit is provided, a default of 10000 records is displayed on screen. Timeout parameter is optional and specifies the number of seconds for the server to timeout for the select operation. If no timeout is provided, server processing will timeout in 60 seconds for the select operation. Please be warned that in certain select operations, it will take a long time if your dataset is large. It is prudent to first try a timeout and check how long a query can take.

The match operation takes a regular expression enclosed with two single quotes. If the selected column matches the regular expression, then the test evaluates to true.

Examples:

```
SELECT similarity(v, '0.1,0.2,0.4','topk=10,type=euclidean_fraction_byte')
      from tvec;


select vector(v, 'type=cosine_fraction_short') from tvec where zid='1232323';

select * from user;

select * from user limit 100;

select * from user limit 1000,100;

select fname, lname, address from user;

select fname, lname, address, age from user limit 10;

select fname, lname, address  from user where fname='Sam' and lname='Walter';

select * from user where fname='Sam' and lname='Walter';

select * from user where fname='Sam' or ( fname='Ted' and lname like 'Ben%');

select * from user where fname >= 'Sam';

select * from user where fname >= 'Sam' and fname < 'Zack';

select * from user where fname >= 'Sam' and fname < 'Zack' and ( zipcode =
94506 or zipcode = 94507);

select * from user where fname >= 'Sam' and zipcode in ( 94506, 94582 );

select * from t1_index where uid='frank380' or uid='davidz';

select * from sales  where stime between '2014-12-01 00:00:00 -08:00' and
'2014-12-31 23:59:59 -08:00';

select avg(amt) as amt_avg from sales;

select sum(amt) amt_sum from sales where ...;

select sum(amt) amt_sum from sales group by key1, key2 limit 10;

select sum(amt+fee) as amt_sum from sales timeout 300;

select * from metrics1;

select a, pt:x, pt:y, pt:m1, pt:m2, pt:m3 from mmetrics;

select * from cirm;

select c:x, c:y, c:m1, c:m2 from cirm;
```

The c:m1 and c:m2 are the values of the metrics associated with the column that has metrics.

Getfile command

If there are some columns that are of type 'file', you can download the file data and save it into a local file on the client host. The syntax is:

```
Getfile COL into localfilapath from store where key=…;
```

Where localfilepath is file on client's computer and please make sure the "where" condition must specify the unique row that contains the file.

```
Getfile COL into stdout from store where key=…;
```

Where the stdout is a system keyword and represents the Linux standard output stream. The get file command does not save the file data into a file. However it streams the file data into the standard output stream which might be useful for playing media files into browsers.

You can also download multiple files from server into files on client side.

```
Getfile COL1 into fpath2, COL2 into fpath2 from store where key=…;
```

You can get the file size, file time, md5sum of files in a store:

```
Getfile col1 size, col2 time, col2 md5 from t123 where …;
Output is: col1.size:[38393] col2.time:[…] col2.md5:[IEdjJDDKKDnxnE]
```

To get the file type of a file, use the following command:

```
Getfile col1 type from t123 where …;
```

To get the hostname where a file is stored, use the following command:

```
Getfile col1 host from t123 where …;
```

To get the full path of a file, use the following command:

```
Getfile col1 fpath from t123 where …;
```

To get the host and full path of a file, use the following command:

```
Getfile col1 hostfpath from t123 where …;
```

To get the size of a file in MB (megabytes), use the following command:

```
Getfile col1 sizemb from t123 where …;
```

To get the size of a file in GB (gigabytes), use the following command:

```
Getfile col1 sizegb from t123 where …;
```

In getting the file attributes only, the "where condition" need not to be unique. The where clause can be constructed to include multiple rows which may contain one or more file columns. The attributes of the files in each row will be reported back to the client.

Managing files in JaguarDB is important for building data-lakes for AI. A data lake for AI refers to a scalable repository that stores a vast volume of raw, structured, semi-structured, and unstructured data in its native format. This approach allows organizations to accumulate a wide array of data types, including text, images, videos, logs, time series, and more. The data lake concept is particularly beneficial for AI applications due to its flexibility, scalability, and potential to facilitate advanced analytics and machine learning.

When retrieving or downloading the data of a file, the where clause must be a unique query condition, specifying the only one record that satisfies the where clause. You can download multiple files on the same record, but not multiple files from multiple records.

Update SQL Command

```
Update store set VCOL:vector='VID:VECTOR_STR';

Update store set VCOL:vector='VECTOR_STR' where …;

update store set VALUE='...', VALUE='...', ... where KEY1='...' and
KEY2='...', ... ;

update store set VALUE='...', VALUE='...', ... where KEY1>='...' and
KEY2>='...', ...;

update store set KEY='...', VALUE='...', ... where KEY='...' and
VALUE='...', ...;
```

```
Example:
```

```
update tvec set v:vector='11223344:0.1,0.2,0.3,0.4' where 1;

update tvec set v:vector='0.1,0.2,0.3,0.4' where zuid='ZMdjjrjrII8u@00';

update user set address='200 Main St., SR, CA 94506' where fname='Sam' and
lname='Walter';

update user set fname='Tim', address='201 Main St., SR, CA 94506' where
fname='Sam' and lname='Walter';
```

Delete SQL Command

```
delete from store;
delete from store where KEY='...' and KEY='...' and ... ;
delete from store where KEY>='...' and KEY<='...' and ... ;
```

```
Example:
   delete from tvec where fid='JDjruf3394JJ@00';
   delete from junkstore;
   delete from user where fname='Sam' and lname='Walter';
```

Drop command

stores or indexes can be dropped with the drop command:

```
drop store [if exists|force] store;
drop index INDEX on store;
```

```
Example:
   drop store user;
   drop index user_idx1 on user;
```

Truncate command

Data in a store can be deleted with the truncate command (schema is left untouched):

```
   truncate store store;
```

```
Data in store will be deleted, but the store schema still exists.
```

```
Example:
   truncate store mystore;
```

Alter command

The name of a key column can be changed to a different name:

```
alter store store rename OLDKEY to NEWKEY;
This command renames a key name in store store.
```

```
Example:
   alter store mystore rename mykey1 to userid;
```

Spool command

Send the output of a command to a file on client host:

```
spool LOCALFILE;
spool off;
```

```
Example:
   spool /tmp/myout.txt;
   (The above command will make the output data to be written to file
    /tmp/myout.txt)


   spool off;
   (The above command will stop writing output data to any file)
```

Group By Statement

Aggregation operation can be performed on numerical columns of a store or index with group by clause. The elements in the group by columns can be any column or columns. If they are all the keys or the left-subset of keys in the store or index, no sorting operation is performed so it would be faster than non-key group by.

If a non-numerical column is selected in the select clause without the "lastvalue", the value of any record is used and displayed.

```
Select [aggregation(COL)] from store/INDEX group by c1, c2, c3, … order by …
limit …;
```

Group By LastValue Statement

The last records of certain groups in a store or index can be selected with "group by lastvalue" statement.

```
Select [COL1, COL2, …] from store/INDEX group by lastvalue k1, k2, k3;
```

As a result of the above statement, the records are grouped according to the keys k1, k2, and k3, and the very last record of each such group is displayed.

Order By Statement

From the select results (which may contain group by statement), data can be further ordered by one or more columns:

```
order by COL1, COL2, COL3 [ASC/DESC]…
```

The default sorting order is ASC (meaning ascending). Descending order can be represented by DESC. The ordered columns have to be either all in ASC or all in DESC. Mixed ordering (one column in ASC but another column in DESC) is not supported. When DESC appears in an order by and a limit condition is used, the system actually returns the last greater [LIMIT] records.

Aggregation Statement

Aggregation functions can be applied to one more columns in a store or index in combination with other aggregation functions.

```
Examples include:
  Select sum(col1 + col2 ) + 2* avg(col3) from tab123 where …;
  select sum(x_coord + y_coord) as ss, 2*avg(minute1) as min2 from t123;
  select sum(x_coord + y_coord) as ss, 2*stddev(minute1) as std2 from t123;
```

System Limits

Limits of store Columns

A store can have a maximum of 4096 columns.

Limits of Vector Columns in a store

A store can have a maximum of 4096 vector columns.

Limits on Length of a Database Name

The name of a database can have a maximum of 64 characters.

Limits on Length of a Column Name

The name of a column can have a maximum of 32 characters.

Limits on Number of Bytes of a Row

Each record or row in a store can have a maximum of 2 billion bytes.

Schema Change

Use spare_ Column

When a store is created, a spare_ column with 30% extra space is allocated (which can be configurable in server.conf file). Users can add more columns to a store, using the extra spare_ column. If the spare_ column still has space, then the following command can be used to add a new column:

    alter store store add COLUMN TYPE;

    Example: alter store tab123 add spacex char(4);

store Change

When the schema of a store does need a major change (in early stage recommended), the following procedures are recommended:

1) Execute the jagexport command to export the store data
2) Drop the store
3) Re-create the store with new columns by following these rules:
   a) Some new columns can be added
   b) Some old columns can be dropped
   c) Smaller size columns can be changed to bigger size columns (int->bigint, wider char)
   d) Remaining column names should be kept the same
4) Execute the jagimport command
5) After SUCCESSFUL import, run the jag client program to cleanup the exported data:
   $ jag -u admin -p -d DB -h :8888
      jaguar> import into DB.store complete;

Fault Tolerance

In an operational Jaguar cluster, one or more Jaguar nodes can go offline but the cluster will still function. Data records are replicated to nodes that are alive. When the down-node is up again, data is restored from the live nodes. Keep in mind that you should always have one or more spare servers ready to be commissioned. The spare servers should be installed with the same version of Jaguar software and its $JAGUAR_HOME/data directory is empty. If one Jaguar node is completely broken (such as damaged hard drive, etc.), the spare server should be configured with the same IP address as the broken server and conf/cluster.conf file is updated. Then the spare server can just be connected to the Jaguar cluster network. Data will flow from other live nodes into this new server and everything will work normally. If a Jaguar server is temporarily disconnected from the rest of the nodes in the cluster, nothing needs to be done. When the network connection comes back up, data will be automatically restored to the node.

Expanding Jaguar Cluster

As data sizes continue to grow, the need to expand a Jaguar cluster arises, whether to accommodate more data or enhance overall cluster performance. Expanding or scaling out a Jaguar cluster is a straightforward process that involves just a few simple steps. Unlike other distributed databases that necessitate the time-consuming migration of data from old servers to

new ones, which can take hours or even days, Jaguar's scaling process is instant and demands no data migration among servers. Throughout the scaling process, the Jaguar cluster operates seamlessly, ensuring uninterrupted functionality before and after the expansion.

Here are the three simple steps to expand your current cluster:

1. Set up your new cluster like when you setup your existing cluster. The file conf/cluster.conf contains only the IP addresses of the hosts in the new cluster. (one IP address per line). Start all Jaguar servers of the new cluster.
2. Copy conf/cluster.conf in the new cluster to one of the hosts in the old cluster and name it as conf/newcluster.conf.
3. On the host which has the conf/newcluster.conf file, connect to Jaguar cluster with admin account and in exclusive mode:
   $JAGUAR_HOME/bin/jag -u admin -p -x yes -h 127.0.0.1:8888
   jaguar> addcluster;

After the command "addcluster" is executed, the new server hosts are accepted by the current cluster and will start to take read and write requests. When needed in the future, each new cluster of servers can be added with the same method.

The following example demonstrates how you can add a new cluster of hosts:

Suppose you have 192.168.1.10, 192.168.1.11, 192.168.1.12, 192.168.1.13 on your current cluster. You want to add a new cluster with new hosts: 192.168.1.14, 192.168.1.15, 192.168.1.16, 192.168.1.17 to the system. The following steps demonstrate the process to add the new cluster into the system:

Step 1. Provision the new hosts 192.168.1.14, 192.168.1.15, 192.168.1.16, 192.168.1.17 and install jaguardb on these hosts (cluster.conf can be empty)

Step 2. The new cluster is a blank cluster, with no database schema and store data

Step 3. Admin user should log in (or ssh) to a host in EXISTING cluster, e.g., 192.168.1.10

Step 4. On host 192.168.1.10, prepare the newcluster.conf file, with the IP addresses of the hosts on each line separately:

In $JAGUAR_HOME/conf/newcluster.conf file:

192.168.1.14

192.168.1.15

192.168.1.16

192.168.1.17

Step 5. Connect to local jaguardb server from the host that contains the newcluster.conf file

$JAGUAR_HOME/bin/jag -u adminapikey -h 192.168.1.10:8888 -x yes

Step 6. While connected to the jaguardb, execute the addcluster command:

jaguardb> addcluster;

The addcluster will take approximately one second to finish. All the new hosts are instantly added to current system which will function normally.

Note:

1. Never directly add new hosts in the file $JAGUAR_HOME/conf/cluster.conf manually

2. Any plan to add a new cluster of hosts must implement the addcluster process described here.

3. Execute addcluster command in the existing cluster, NOT in the new cluster.

4. It is recommended that existing clusters and new cluster contain large number of hosts. (dozens or hundreds).

For example, the existing cluster can have 30 hosts, and the new cluster can have 100 hosts.

5. Make sure JaguarDB is installed on all the hosts of the new cluster, and connectivity is good among all the hosts.

6. The server and client software must have the same version, on all the hosts of both existing clusters and the new cluster.

7. After adding a new cluster, all hosts will have the same cluster.conf file.

8. Make sure REPLICATION factor is the same on all the hosts.

During the operation of adding more clusters to the system, any client that is already connected to the system need not to disconnect and reconnect to the server it is currently connected to. The client can continue performing database operations as usual.

Jaguar Database Security

User data is considered extremely important in Jaguar database. Several measures can be taken to protect user data in Jaguar database system.

Network Protection

In the network or subnet where Jaguar is in operation, firewall or Security Policy can be setup for protecting the system against malicious attempts. In an on-premise environment, router firewall can be configured to allow only Jaguar database traffic. In a cloud environment, security policy can be configured to allow only Jaguar data communication. Even a database firewall can be employed to allow only legitimate SQL commands to be passed through, thus any threats such as SQL-injection or other attacks can be prevented.

Server System Protection

SELinux is a hardened Linux kernel that provides strong system security to Linux systems. With SELinux installed and enabled, user permission, process control, file control are better managed to achieve higher-level security.

User Privilege and File Permission

All files and data in Jaguar are owned by only one user (jaguar). Other users do not have the permission to read and write data in Jaguar database. The authorized user has password in the Linux system and we strongly recommend a strong-security password for the user. The credentials should be securely saved and protected. File permission should be strictly enforced and maintained across Jaguar database servers.

Database User Authentication

User accounts in Jaguar database are also required to have a password that is minimum of 16 characters long. Any shorter passwords are rejected by the system. The username and password should be kept properly by all users and developers of the system and they should be frequently updated with string-security content.

User Level Control

Users of Jaguar are classified into two categories: 1) administrator; 2) regular user. Only the administrator has the privilege to create and delete databases, regular user accounts. The regular users can only create and drop stores, indexes, insert and modify data records.

Server Communication Control

In a cluster of Jaguar database servers, messages between servers are frequently passed and processed. The servers use tokens (SERVER_TOKEN) to identify and authorized themselves to obtain permission to send request to other servers. The tokens are created during initial database installation process and are unique among Jaguar customers. This ensures the integrity of a Jaguar database cluster.

Access Control List

There are whitelist (conf/whitelist.conf) and blacklist (conf/blacklist.conf) control files that are used to limit client access to Jaguar servers. Only the clients whose IP address or IP segment is included in the whitelist are authorized to connect to Jaguar servers. For certain IP addresses in the whitelist, access can be denied if they belong to a blacklist. If no whitelist and blacklist are provided, then all client access is granted. We strongly recommend that the access control lists be used in Jaguar cluster for maximum system security.

Log Monitoring

Jaguar servers generate log entries for client connection and store management. Database administrator is recommended to regularly monitor the log information, and check for illegal access to the database or database store modifications.

Data Import and Synchronization

In Jaguar github web site there are programs to import and synchronize data between other databases and Jaguar database. There are also example programs on how to import data and synch data from Oracle, MySQL and other databases. The mechanism to synchronize data is: 1) Jaguar database must create same store as in other databases; 2) Other databases create changelog and triggers to capture changes in an original store or stores; 3) import all data from other databases to Jaguar; 4) start java sync server on a Jaguar server to monitor the records in the changelog stores and update the Jaguar stores.

Step One: Create stores on Jaguar

Suppose you have some stores on another database, you must first create the corresponding stores on Jaguar. **This must be performed on a Jaguar host.**

Example: Use github.com/datajaguar/jaguardb: importsync/databaseimport/from_oracle/ create_jaguar_store.sh to create stores on Jaguar from any Jaguar host. In example1 directory you can use create_jaguar_store_example1.sh as a reference. Note: please make sure you first compile the JDBC programs: cd importsync/jdbc; ./compile.sh

Step Two: Create Changelog Triggers

On other database system you must create changelog and triggers for the stores. **This step must be performed on the other database system.**

Please go to github and the following program to create changelog and triggers:

importsync/databasesync/oracle/OracleToJaguar/oracle_create_changelog_trigger.sh

Result：The changelog for store234 is created. If store234 has any insert, update, or delete, a new record in changelog is added.

Step Three: Importing Data

Importing data from other database to Jaguar database. This step must be executed on a Jaguar server.

Please goto github and find this program：

importsync/databaseimport/from_oracle/example1/ import_from_oracle.sh

Please note that in appconf.oracle you need to have correct source_jdbcurl , dest_jdbcurl, and other parameters.

Step Four: Updating Jaguar stores

On a Jaguar host, a java sync server needs to be started to monitor the changelog stores on the other database system. **This step must be performed on Jaguar host.**

Please use the following example program in jaguar github:
importsync/databasesync/oracle/OracleToJaguar/example1/start_sync_oracle_to_jaguar.sh

You need to change appconf.oracle to suite your own environment.

```
appconf.oracle:

source_jdbcurl=jdbc:oracle:thin:@//192.168.7.120:1522/test

 (192.18.7.120 is IP address of Oracle server)

source_store=store234|store345   (separate stores with vertical line)

source_user=test

source_password=test

sleep_in_millis=3000   (scan changelog store every 3 seconds)

keep_rows=10000          (keeping some records in changelog)


dest_jdbcurl=jdbc:jaguar://localhost:8888/test  (port of jaguar server)

dest_user=test

dest_password=test


### set true to stop java server anytime when java is running

# stop=true


## print more debug info

# debug=true
```

If you are just importing data from other database to Jaguar, then you need to execute only step one (creating jaguar store) and step three (importing data to jaguar). The java sync server can be stopped any time and restarted without affecting the synchronization. However, for real-time updates, it is recommended that the sync server be running all the time and a smaller sleep interval is desired.

Spark Data Analysis

Since Jaguar provides JDBC connectivity, developers can use Apache Spark to load data from Jaguar and perform data analytics and machine learning. The advantages provided by Jaguar is that Spark can load data faster, especially for loading data satisfying complex conditions, from Jaguar than from other data sources. The following code is based on two stores that have the following structure:

```
create store int10k ( key:  uid int(16), score float(16.3), value: city char(32) );
create store int10k_2 ( key: uid int(16), score float(16.3), value: city char(32) );
```

Scala program:

```
import org.apache.spark.SparkConf

import org.apache.spark.SparkContext

import org.apache.spark.SparkContext._

import scala.collection._

import org.apache.spark.sql._

import org.apache.spark.sql.types._

import org.apache.log4j.Logger

import org.apache.log4j.Level

import com.jaguar.jdbc.internal.jaguar._

import com.jaguar.jdbc.JaguarDataSource
```

```scala
object TestScalaJDBC {

  def main(args: Array[String]) {

    sparkfunc()

  }


  def sparkfunc()

  {

    Class.forName("com.jaguar.jdbc.JaguarDriver");

    val sparkConf = new SparkConf().setAppName("TestScalaJDBC")

    val sc = new SparkContext(sparkConf)

    val sqlContext = new org.apache.spark.sql.SQLContext(sc)

    import sqlContext.implicits._


    Logger.getLogger("org").setLevel(Level.OFF)

    Logger.getLogger("akka").setLevel(Level.OFF)


    val people = sqlContext.read.format("jdbc")

      .options(

        Map( "url" -> "jdbc:jaguar://127.0.0.1:8888/test",

            "dbstore" -> "int10k",

            "user" -> "test",

            "password" -> "test",

            "partitionColumn" -> "uid",

            "lowerBound" -> "2",

            "upperBound" -> "2000000",

            "numPartitions" -> "4",

            "driver" -> "com.jaguar.jdbc.JaguarDriver"

      )).load()
```

```
// work fine

people.registerTempstore("int10k")

people.printSchema()


val people2 = sqlContext.read.format("jdbc")

  .options(

      Map( "url" -> "jdbc:jaguar://127.0.0.1:8888/test",

          "dbstore" -> "int10k_2",

          "user" -> "test",

          "password" -> "test",

          "partitionColumn" -> "uid",

          "lowerBound" -> "2",

          "upperBound" -> "2000000",

          "numPartitions" -> "4",

          "driver" -> "com.jaguar.jdbc.JaguarDriver"

  )).load()

people2.registerTempstore("int10k_2")


// sort by columns


people.sort("score").show()

people.sort($"score".desc).show()

people.sort($"score".desc, $"uid".asc).show()

people.orderBy($"score".desc, $"uid".asc).show()


// select by expression

people.selectExpr("score", "uid"  ).show()
```

```
people.selectExpr("score", "uid as keyone"  ).show()

people.selectExpr("score", "uid as keyone", "abs(score)" ).show()


// select a few columns

val uid2 = people.select("uid", "score")

uid2.show();


// filter rows

val below60 = people.filter(people("uid") > 20990397 ).show()


// group by

people.groupBy("city").count().show()


// groupby and average

people.groupBy("city").avg().show()


people.groupBy(people("city"))
    .agg(
        Map(
            "score" -> "avg",
            "uid" -> "max"
          )
        )
    .show();


// rollup

people.rollup("city").avg().show()

people.rollup($"city")
```

```scala
      .agg(
        Map(
          "uid" -> "avg",
          "score" -> "max"
          )
        )
      .show();


// cube
people.cube($"city").avg().show()
people.cube($"city")
      .agg(
        Map(
          "uid" -> "avg",
          "score" -> "max"
          )
        )
      .show();


// describe statistics
people.describe( "uid", "score").show()


// find frequent items
people.stat.freqItems( Seq("uid") ).show()


// join two stores
people.join( people2, "uid" ).show()
people.join( people2, "score" ).show()
```

```
people.join(people2).where ( people("uid") === people2("uid")  ).show()

people.join(people2).where ( people("city") === people2("city")  ).show()

people.join(people2).where ( people("uid") === people2("uid") and people("city") ===
people2("city")  ).show()

people.join(people2).where ( people("uid") === people2("uid") && people("city") ===
people2("city") ).show()

people.join(people2).where ( people("uid") === people2("uid") && people("city") ===
people2("city") ) .limit(3).show()


// union
people.unionAll(people2).show()


// intersection
people.intersect(people2).show()


// exception
people.except(people2).show()


// Take samples
people.sample( true, 0.1, 100 ).show()


// distinct
people.distinct.show()


// same as distinct
people.dropDuplicates().show()


// cache and persist
people.dropDuplicates.cache.show()
```

```
people.dropDuplicates.persist.show()


//  SQL dataframe

val df = sqlContext.sql("SELECT * FROM int10k where uid < 200000000 and city between
'Alameda' and 'Berkeley' ")

df.distinct.show()
```

The class generated from the above Scala program can be submitted to Spark as follows:

```
/bin/spark-submit --class TestScalaJDBC  \

   --master spark://masterhost:7077  \

   --driver-class-path    /path/to/your/jaguar-jdbc-2.0.jar \

  --driver-library-path  $JAGUAR_HOME/lib \

   --conf spark.executor.extraClassPath=/path/to/your/jaguar-jdbc-2.0.jar  \

   --conf spark.executor.extraLibraryPath=$JAGUAR_HOME/lib   \

    /path/to/your_project/target/scala-2.10/testjdbc_2.10-1.0.jar
```

SparkR with Jaguar

Once you have R and SparkR packages installed, you can start the SparkR program by
executing the following command:

```
 #!/bin/bash

export JAVA_HOME=/home/jvm/jdk1.8.0_60
LIBPATH=/usr/lib/R/site-library/rJava/libs:$JAGUAR_HOME/lib
LDLIBPATH=$LIBPATH:$JAVA_HOME/jre/lib/amd64:$JAVA_HOME/jre/lib/amd64/server
JDBCJAR=$JAGUAR_HOME/lib/jaguar-jdbc-2.0.jar

sparkR \
–driver-class-path $JDBCJAR \
–driver-library-path $LDLIBPATH \
```

–conf spark.executor.extraClassPath=$JDBCJAR \
–conf spark.executor.extraLibraryPath=$LDLIBPATH

Then in the SparkR command line prompt, you can execute the following R commands:

library(RJDBC)
library(SparkR)

sc <- sparkR.init(master="spark://mymaster:7077", appName="MyTest")

sqlContext <- sparkRSQL.init(sc)

drv <- JDBC("com.jaguar.jdbc.JaguarDriver", "/home/jaguar/jaguar/lib/jaguar-jdbc-2.0.jar", "`")

conn <- dbConnect(drv, "jdbc:jaguar://localhost:8888/test", "test")

dbListstores(conn)

df <- dbGetQuery(conn, "select * from int10k where uid > 'anxnfkjj2329' limit 5000;")

head(df)

#correlation
> cor(dfuid,dfscore)
[1] 0.05107418

#build the simple linear regression
> model<-lm(uid~score,data=df)
> model

Call:
lm(formula = uid ~ score, data = df)

Coefficients:
(Intercept) score
2.115e+07 1.025e-03

```
#get the names of all of the attributes
> attributes(model)
$names
[1] "coefficients" "residuals" "effects" "rank"
[5] "fitted.values" "assign" "qr" "df.residual"
[9] "xlevels" "call" "terms" "model"


$class
[1] "lm"
```

Spatial Data Management

Jaguar supports spatial data in various forms, including vector geometry objects and raster coordinates. Vector geometry objects include square, rectangle, circle, ellipse, cube, box, sphere, etc. Raster objects include multipoint, linestring, multilinestring, polygon, and multipolygon. Spatial data often occurs in autonomous driving and traffic management systems.

Spatial Data Types

In addition to existing data types in Jaguar, new spatial data types are also supported. In spatial data management, the reference system and direction of an object or surface are important

factors.
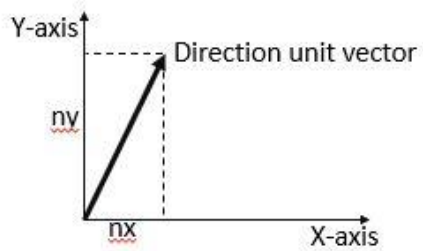
$0 <= nx <= 1$
$0 <= ny <= 1$
In 2D: sqrt($nx^2 + ny^2$) = 1
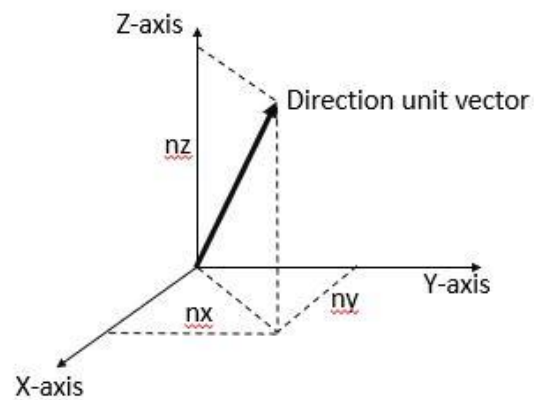
$0 <= nx <= 1$
$0 <= ny <= 1$
In 3D: sqrt($nx^2 + ny^2 + nz^2$) = 1

point(x y)
二维点

point3d(x y z)
三维点

Line(x1 y1, x2 y2)
二维线段

Line3D(x1 y1 z1, x2 y2 z2)
三维线段
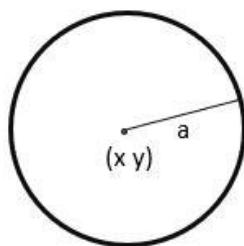
Circle(x y a)
二维圆

Circle3d(x y z a)
三维圆
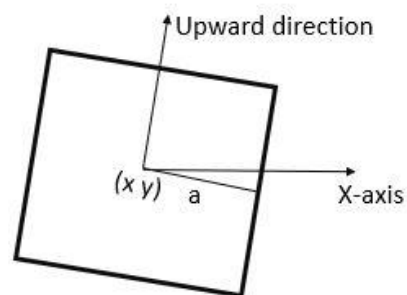
Square(x y a nx)
二维正方形

Square3d(x y z a nx)
三维正方形
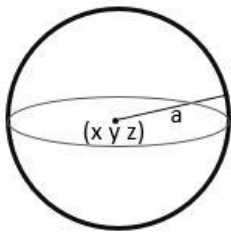
sphere(x y z a)
球体
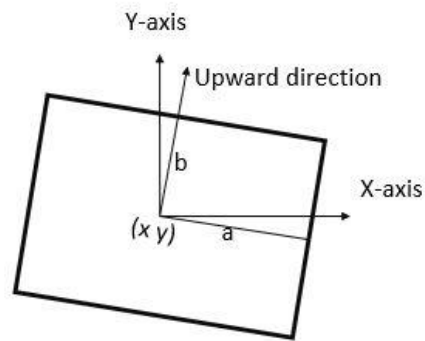
rectangle(x y a b nx)
矩形
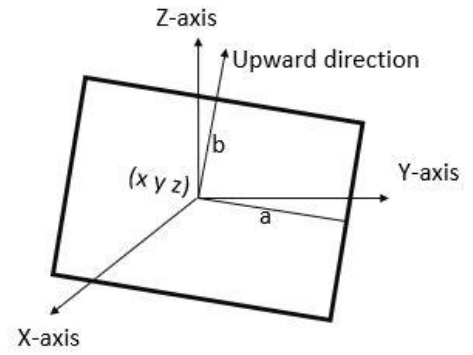
rectangle3d(x y z a b nx ny)
三维矩形面
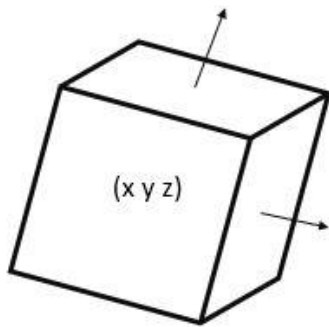
cube(x y z a nx ny)
a = half edge
Nx ny: direction
立方
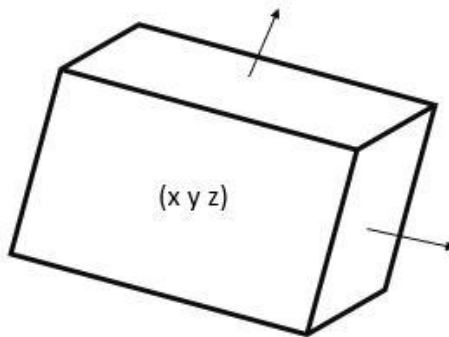
box(x y z a b c nx ny)
a = half width
b = half depth
c = half height
Nx ny: direction
长方体
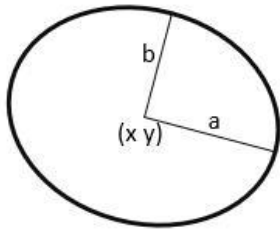
ellipse(x y a b nx)
a = half width
b = half height
Nx: direction
椭圆
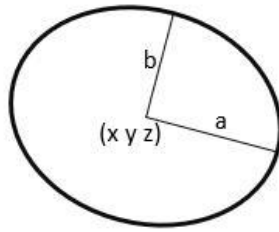
ellipse3d(x y z a b nx ny)
a = half width
b = half height
Nx ny: direction
三维椭圆面
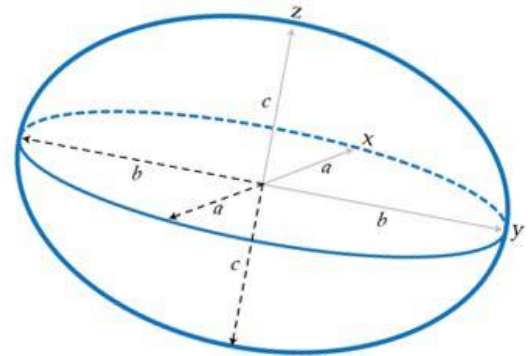
ellipsoid(x y z a b c nx ny)
a= half width
b = half depth
c = half height
Nx ny: direction of own Z-axis
立体椭圆体
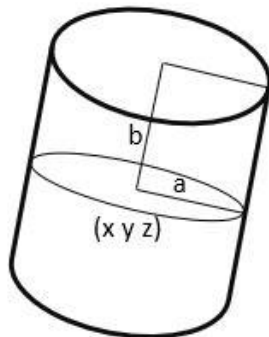
cynlinder(x y z a b nx ny)
a = radius
b = half height
Nx ny: direction
圆柱
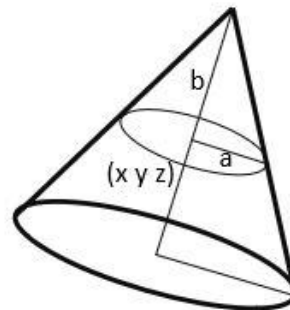
cone(x y z a b nx ny)
a = radius at half height
b = half height
Nx ny: direction
圆锥
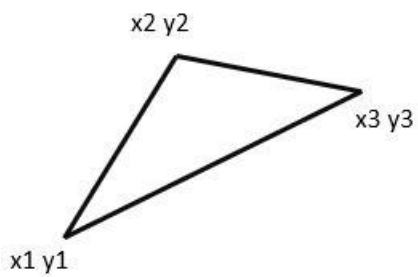
triangle(x1 y1 x2 y2 x3 y3)
平面三角形
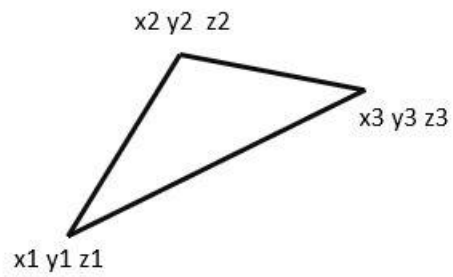
triangle3D(x1 y1 z1 x2 y2 z2 x3 y3 z3)
三维空间三角面
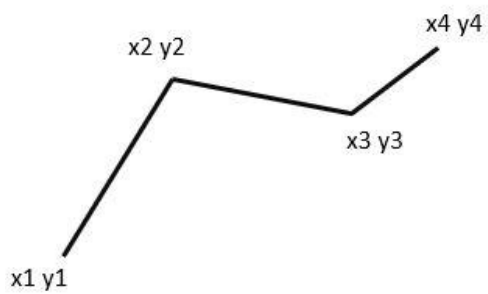
linestring(x1 y1, x2 y2, x3 y3, x4 y4)
线串
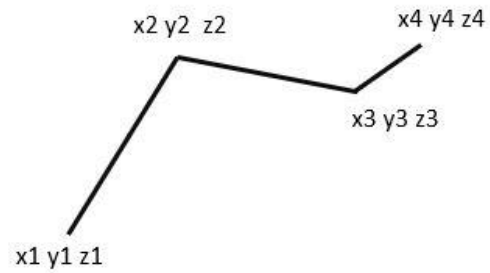
linestring3D(x1 y1 z1, x2 y2 z2, x3 y3 z3, x4 y4 z4)
三维空间线串
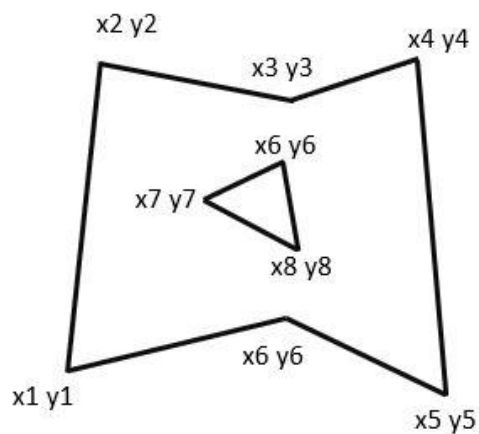
polygon((x1 y1, x2 y2, x3 y3,x4 y4,x5 y5,x6 y6, x1 y1), (x6 y6,x7 y7,x8 y8,x6 y6))
平面多边形 – 可含有一个或多个洞
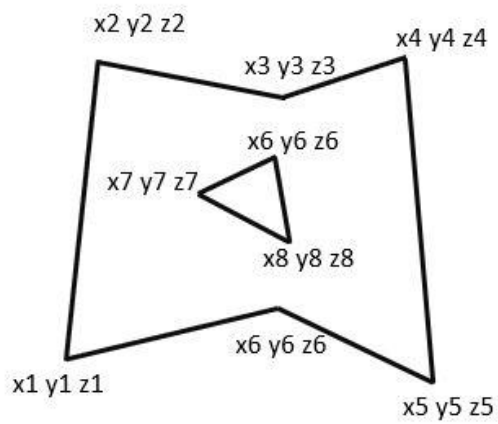
polygon3D((x1 y1 z1, x2 y2 z2, x3 y3 z3,x4 y4 z4,x5 y5 z5,x6 y6 z6, x1 y1 z1), (x6 y6 z6,x7 y7 z7,x8 y8 z8,x6 y6 z6))
三维多边形 – 可含有一个或多个洞

multipoint(x y)
二维多点

multipoint3d(x y z)
三维多点

x2 y2
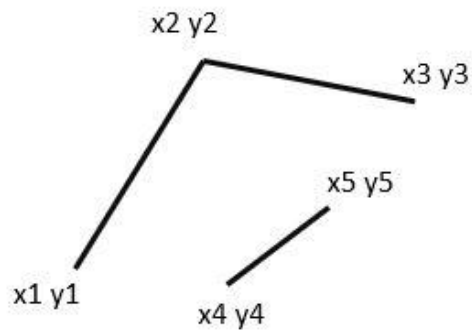
x3 y3

x5 y5

x1 y1

x4 y4

multilinestring((x1 y1, x2 y2, x3 y3), (x4 y4, x5 y5))
多线串

multilinestring3d((x1 y1 z1, x2 y2 z2, x3 y3 z3), (x4 y4 z4, x5 y5 z5))
三维多线串
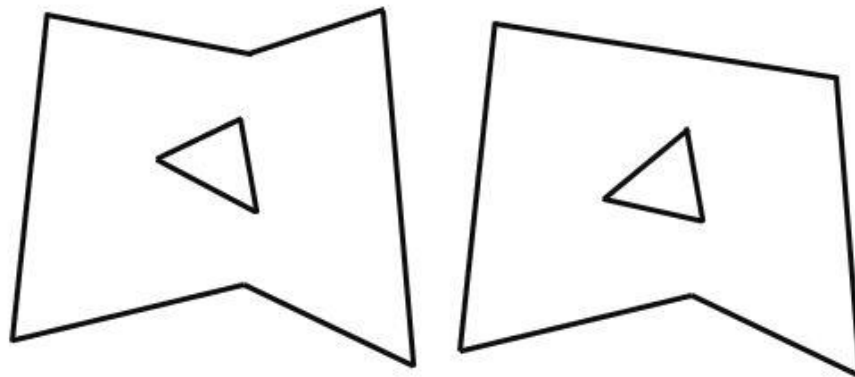
multipolygon((polygon1), (polygon2), …)
多-多边形 – 可含有多个多边形

multipolygon3d((polygon1), (polygon2), …)
三维多-多边形 – 可含有多个三维多边形

Spatial Data Storage

Creating store Containing Spatial Data

In creating store containing spatial data types, the type of a column can have a spatial reference identifier (SRID). If no SRID is provided, the default value is zero, meaning it is a simple geometric coordinate system. In addition to the SRID of the column, the number of metrics associated with location point or a shape can be specified with the "metrics:" keyword.

The following examples show how to create stores with spatial columns.

    create store if not exists geom (key: a int, value: pt point(srid:4326), b int);

    create store if not exists geom2 (key: a int, value: pt point(srid:wgs84), b int);

    create store if not exists geom3 (key: a int, value: pt point, b int);

    create store dot (key: a int, pt1 point, b int, pt2 point, value: c int, d int, pt3 point3d);

    create store cb (key: a int, q1 cube, b int, q2 cube, value: c int, q3 cube);

    create store es (key: a int, c ellipsoid, value: d int, e ellipse);

    create store linestr (key: lsw linestring(srid:wgs84), a int, value: lss linestring);

    create store pol (key: a int, value: po2 polygon, po3 polygon3d, tm datetime, ls linestring);

    create store mline (key: a int, value: m multilinestring, m3 multilinestring3d);

    create store mpg (key: a int, value: p multipolygon, p3 multipolygon3d);

    create store street (key: a int, value: pt linestring(srid:wgs84,metrics:10), b int);

    create store base (key: a int, value: pt point(srid:wgs84,metrics:20), char(32));

The number of metrics is unlimited, as long as the storage space allows. Each metric has a length of 8 bytes, with default value of zero. The metrics are identified by mN, such as:

    select col:m1, col:m3 from mytab where a=100 and col:x=200 and col:y=300;

Inserting Spatial Data

Spatial data can be inserted into a store as any other data types. GeoJson formatted data is also accepted by Jaguar. GeoJson format begins with "json" type identifier. The following examples show how to insert spatial data into stores.

insert into geom values (1, point(23.2222 52.39393), 123);

insert into geom2 values (1, point(22 33), 123, point(99 221));

insert into geom2 values (10, json({"type":"Point", "coordinates": [2,3]}), 123);

insert into geom3 (b, pt2, pt1, a) values (2, point(25 33), point(23 451), 153);

insert into d6 (pt2, a, b, pt1) values (json({"type":"Point", "coordinates": [124,351]}), 209, 13, point(92 19));

insert into sph1 (s1, b, a, s2) values (sphere(2 3 4 123), 921, 234, sphere(99 22 33 20000));

insert into rect1 (c, a, r1) values (22, 31, rectangle(29 13 48 19));

insert into cyn values (1, cylinder(1 2 3 45 88 0.3), 1239);

insert into cn values (1, cone(1 2 3 45 88), 1239, cone(33 22 44 44 99 0.4 0.3));

insert into eps values (1, ellipse(1 2 45 88), 1239, ellipse(22 44 44 99));

insert into tri values (triangle3d(11 33 88 99 23 43 9 8 2), 123);

insert into linestr values (2, linestring(11.13 2,2.9 33 , 33 44, 5.5 6.6, 55 66, 77 88), 210, linestring(3.3 4.4, 5.5 6.6, 8.9 9));

insert into pol values (1, polygon((0 0, 20 0, 88 99, 0 0)));

insert into mp values (125, multipoint(1 2 , 3 4, 2 1), json({"type":"MultiPoint", "coordinates": [[1,2,3],[3,4,5]] });

insert into mline values(1, multilinestring((0 0,2 0,8 9,0 0),(1 2,2 3,1 2)),multilinestring3d((1 1 1,2 2 2,3 3 3),(2 2 2,3 3 1)));

It should be noted that a polygon can have an outer "ring" and several internal "rings" or holes inside the outer ring. The start and end points of a ring must be the same to form a closed shape. A multipolygon just contain multiple polygons where each polygon can have one or more rings. An example of polygon is:

polygon((0 0,2 0,8 9, 23 32, 0 0), (1 2,2 3, 4 5, 1 2), (12 32, 33 44, 50 60, 12 32))

An example of 3D multipolygon is:

multipolygon3d(((1 1 1,2 2 2,3 3 3, 1 1 1), (2 2 2,3 3 1, 3 5 6, 2 2 2)), ((1 2, 20 30, 30 40, 1 2), (0 0, 1 2, 2 3, 0 0)));

Metric data should be placed following the coordinates and geometric data of a shape. For example,

Insert into t values (square(0 0 100 0.2 m1 m2 m3 …));

Insert into ms values (linestring(0 0 m1 m2 m3, 3 4 m1 m2 m3, 4 6 m1 m2 m3));

Insert into ms values (triangle(0 0, 3 4, 4 6, m1 m2 m3));

You can prepare a csv-like file containing spatial data and load the file into a store. An example of such file is (input.csv):

   1,"john doe", point(1 2), 421, linestring(11 22, 33 44, 55 66)

   2,"john doe", point(2 2), 321, linestring(101 202, 303 494, 550 676), 876

   3,"sam", point(3 2), 351, linestring(151 282, 33 454, 505 666)

   4,"dave", point(4 2), 39, linestring(171 252, 33 424, 575 696)

Then you can execute the following command to load the data into a store:

```
$JAGUAR_HOME/bin/jagimportcsv  -d DB -t TAB -f input.csv
```

Spatial Data Query

Coordinate

The x, y, z coordinates of a spatial column can be used for data query. For example:

  select * from points where a > 100 and pt:x between 0.3 and 122 and pt:y between 300 and 400;

Within
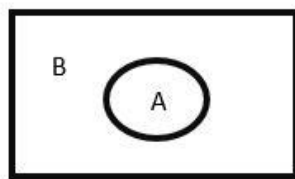
Within(A, B)
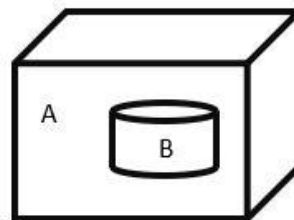
Contain(A, B)

The "within" function checks if a shape is strictly within another shape. For example:

```
select * from poly where nm like 'east%' and within(po, square(0 0 10000) );

select * from poly3d where within(po, cube(0 0 0 10000) );

select * from sq where within(s, polygon( (0 0, 1 1, 2 2, 0 0)) );

select * from sq where within(s, rectangle(0 0 20 30) );
```

NearBy

Checks if a shape is near to a location by a distance.

```
select * from sq1 where nearby(s1, point( 0 0 ), 2000 );
```

The following store lists the shapes that can be calculated for the nearby relation.

NearBy()	Line
	line3d
Between two shapes	triangle
	triangle3d
	cube
	sphere
	circle3d
	box
	ellipsoid
	cone
	rectangle3d
	circle
	square
	rectangle
	ellipse
	point
	point3d

Intersect

Intersect: 相交　　　Disjoint: 没有任何相交

```
select * from sq1 where intersect(s1, square(0 0 200));
select * from rect where intersect(r, rectangle(0 0 200 100));
select * from lstr where intersect(s, rectangle(0 0 200 100));
select * from lstr where intersect(s, circle(0 0 200));
select * from lstr where intersect(s, ellipse(0 0 200 100));
```

CoveredBy

CoveredBy function is similar to "Within" except that some boundary can overlap between two shapes.

Cover

Cover is the opposite of CoveredBy function.

Contain

The "contain" function checks if a shape strictly contains another shape. It is the opposite of the "Within" function between two shapes.

Disjoint

```
select * from sq1 where disjoint(s1, rectangle(0 0 200 300));
```

Disjoint is the opposite of "Intersect" function.

The distance function computes the distance between two shapes.

If the SRID of the shapes are WGS84, then the distance computed is in meters between two shapes described with (longitude, latitude). The X coordinate is the longitude, and Y the latitude. The following store lists the shapes that can be calculated for the distance() function.

Distance()	Line
	line3d
	triangle
Between two shapes	triangle3d
	cube
	sphere
	circle3d
	box
	ellipsoid
	cone
	rectangle3d
	circle
	square
	rectangle
	ellipse
	point
	point3d

Shapes for Location Relation

The following stores list the shapes that can be used to query the location relation among them. Most shapes can be used to query within, contain, cover, coveredby, intersect, disjoint functions.

Point	point
	line
	Triangle
●	square
	Rectangle
	Circle
	ellipse
	linestring
	polygon

Point3D	point3d
	line3d
	cube
●	box
	sphere
	ellipsoid
	cone
	linestring3d

MultiPoint	point
	line
	Triangle
	square
	Rectangle
	Circle
	ellipse
	linestring
	polygon

MultiPoint3D	point3d
	line3d
	cube
	box
	sphere
	ellipsoid
	cone

Line	triangle
	square
	rectangle
	ellipse
	circle
	linestring
	polygon

Line3D	
	Cube
	Box
	Sphere
	Ellipsoid
	cone
	triangle3D
	square3D
	rectangle3D

Triangle	
	triangle
	square
	rectangle
	ellipse
	circle
	linestring
	polygon
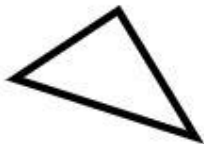
Triangle3D	
	Cube
	Box
	Sphere
	Ellipsoid
	cone

Circle
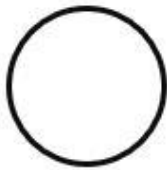
circle
square
rectangle
ellipse
triangle
linestring
polygon

Circle3D
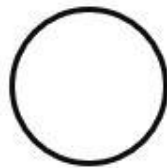
cube
box
sphere
ellipsoid
cone

Sphere
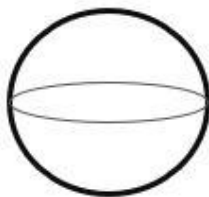
cube
box
sphere
ellipsoid
cone
linestring
polygon

Square	triangle
	square
	rectangle
	ellipse
	circle
	linestring
	polygon

Square3D	Cube
	Box
	Sphere
	Ellipsoid
	cone
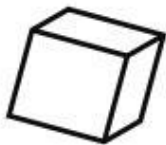
Cube	Cube
	Box
	Sphere
	Ellipsoid
	cone

Rectangle	triangle
	square
	rectangle
	ellipse
	circle
	linestring
	polygon
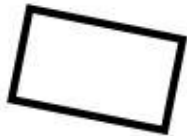
Rectangle3D	Cube
	Box
	Sphere
	Ellipsoid
	cone
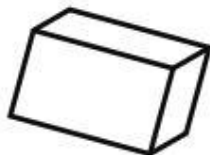
Box	Cube
	Box
	Sphere
	Ellipsoid
	cone

Cylinder

Cube	
Box	
Sphere	
Ellipsoid	
Cone	

Cone
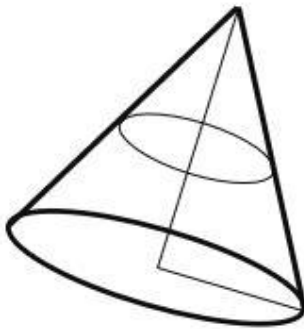
Cube	
Box	
Sphere	
Ellipsoid	
cone	

Ellipse

triangle	
square	
rectangle	
ellipse	
circle	
linestring	
polygon	

Ellipsoid
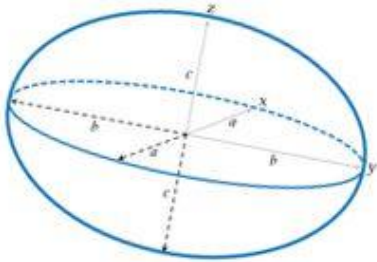
Cube	
Box	
Sphere	
Ellipsoid	
cone	

LineString

triangle	
square	
rectangle	
ellipse	
circle	
linestring	
polygon	

MultiLineString	triangle
	square
	rectangle
	ellipse
	circle
	linestring
	polygon

LineString3D	triangle3D
	square3D
	rectangle3D
	cube
	box
	sphere
	ellipsoid
	cone
	linestring3D

MultiLineString3D	triangle3D
	square3D
	rectangle3D
	cube
	box
	sphere
	ellipsoid
	cone
	linestring3D

Polygon	triangle
	square
	rectangle
	ellipse
	circle
	linestring
	polygon
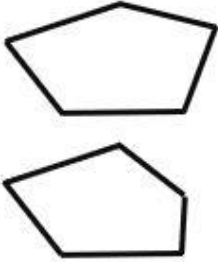
Polygon3D	
	triangle3D
	square3D
	rectangle3D
	cube
	box
	sphere
	ellipsoid
	cone
	linestring3D
	polygon3d
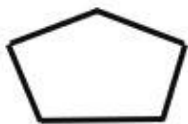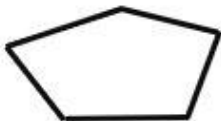
MultiPolygon	
	triangle
	square
	rectangle
	ellipse
	circle
	linestring
	Polygon (within)

MultiPolygon3D	
	cube
	box
	sphere
	ellipsoid
	Cone
	Linestring3d (intersect) Cylinder (intersect)

Area

The area of 2D shapes and surface area of 3D can be computed with the area() function, which is valid for the following geometric or geographic shapes:

- Circle
- Circle3D
- Square
- Square3D
- Rectangle
- Rectangle3D
- Triangle
- Triangle3D
- Sphere
- Cube
- Cylinder
- Cone
- Ellipse
- Ellipsoid
- Polygon
- MultiPolygon

For example:

```
jaguar:mydb> select area(sq3) from geotab where key1 < 100;

jaguar:mydb> select col1, area(circle(1 1 10)) as b from geotab;
```

GeoJson

GeoJson is a format to represent shapes using JSON data string. If a GeoJson string is desired from a query result, then the "all" function can be used. The all() function returns all the data of a complex geometric shape in GeoJson format. For example,

```
select all(s) from lstr where intersect(s, circle(0 0 200));
select all(s) from lstr where a=100;
```

The complex shapes include linestring, multilinestring, polygon, multipolygon and their 3D counterparts. In the client programming API, there are methods to get the all data points of a complex shape: getAll(), getAllByName(), getAllByIndex(). Please refer to the file JagaurAPI.h in $JAGUAR_HOME/include/ directory for detailed information.

Dimension

dimension(col) -- get dimension as integer of a shape column. It returns 2 for 2D shapes, and 3 for 3D shapes, and 0 for non-geometric columns. For example:

```
jaguar:test> select dimension(mp) as dim from mpolygon;
jaguar:test> dim:[2]
```

GeoType

geotype(col)   -- get type as string of a shape column. For example:

```
jaguar:test> select geotype(mp) as tp from mpolygon;
jaguar:test> tp:[MultiPolygon]
```

PointN

pointn(col,n)   -- get n-th point (1-based) of a shape column. It returns in format: [x y] for 2D shapes and [x y z] for 3D shapes. For example:

```
jaguar:test> select pointn(pt, 3) as p3 from mpoints;

jaguar:test> p3:[10.2 32.7]
```

Extent

extent(col) -- get bounding box of a shape column.

For 2D shapes, it returns [xmin ymin xmax ymax]

For 3D shapes, it returns [xmin ymin zmin xmax ymax zmax]

```
jaguar:test> select extent(pd) as bbx from mpoints;

jaguar:test> bbx:[0 0 33.2 49.8]
```

StartPoint

startpoint(col)   -- get the start point of a line string column.

For 2D shapes, it returns [x y]. For 3D shapes, it returns [x y z].

```
jaguar:test> select startpoint(pd) as st from mpoints;

jaguar:test> st:[203.2 178.5]
```

EndPoint

endpoint(col)   -- get the end point of a line string column.

For 2D shapes, it returns [x y]. For 3D shapes, it returns [x y z].

```
jaguar:test> select endpoint(pd) as end from mpoints;

jaguar:test> end:[903.4 778.6]
```

IsClosed

isclosed(col) -- check if raster points of a line string column is closed. (0 or 1)

```
jaguar:test> select isclosed(linecol) as ic from mpoints;

jaguar:test> ic:[0]
```

Number of Points

numpoints(col) -- get total number of points of a line string or polygon

```
jaguar:test> select * from mpoints where numpoints(pcol) < 100;
```

Number of Rings

numrings(col) -- get total number of rings of a polygon or multipolygon. A polygon contains an outer-ring, and also may contain 0 or more inner-rings (holes).

```
jaguar:test> select a, numrings(p) r from mpoints;

jaguar:test> a:[123] r:[3]
```

Number of Lines

numlines(col) -- get total number of linestrings of a multilinestring, polygon or multipolygon.

```
jaguar:test> select a, numlines(p) r from mlstrs;

jaguar:test> a:[123] r:[3]
```

SRID

srid(col)   -- get SRID of a shape column.

Summary

summary(col) -- get a text summary of a shape column. For example:

```
jaguar:test> select summary(p) sum from mpoints;
  sum=[geotype:Polygon srid:0 dimension:2 numpoints:9 numrings:2 isclosed:1]
```

Minimum and Maximum Points

xmin(col)   -- get the minimum x-coordinate of a shape with raster data

ymin(col)   -- get the minimum y-coordinate of a shape with raster data

zmin(col)   -- get the minimum z-coordinate of a shape with raster data

xmax(col)   -- get the maximum x-coordinate of a shape with raster data

ymax(col)   -- get the maximum y-coordinate of a shape with raster data

zmax(col)   -- get the maximum z-coordinate of a shape with raster data

For example:

```
jaguar:test> select xmin(p) xm from mpoints;

jaguar:test> xm=[30.4 332.3 939.9]
```


ConvexHull

Convexhull(geom) -- get the convex hull as polygon of a shape with raster data

For example:

```
jaguar:test> select convexhull(mline) from multlines;

jaguar:test> select numpoints(convexhull(lstr)) from linestr;
```


Centroid

centroid(geom) -- get the centroid coordinates of a vector or raster shape

For example:

```
jaguar:test> select centroid(mline) from multilines;
```


Volume

`volume(geom)` -- get the volume of a 3D shape

Closestpoint

`closestpoint(point(x y), geom)` -- get the closest point on geom from point(x y)

Angle

`angle(line(x y), geom)` -- get the angle in degrees between two lines

Buffer

`buffer(geom, 'STRATEGY')` -- get polygon buffer of a shape.

`The STRATEGY is:`

Parameter	Value	Option
distance	Symmetric or asymmetric	RADIUS: length
join	Round or miter	Number of points
end	Round or flat	
point	Circle or square	Number of points

`'distance=symmetric/asymmetric:RADIUS,join=round/miter:N,end=round/flat,point=circle/square:N'`

Length

`length(geom)` -- get length of line, line3d, linestring, linestring3d, multilinestring, multilinestring3d.

Perimeter

`perimeter(geom)` -- get perimeter length of a closed shape (vector or raster)

Equal

`equal(geom1,geom2)` -- check if shape geom1 is exactly the same as shape geom2

IsSimple

`issimple(geom)` -- check if shape geom has no self-intersecting or tangent points

IsValid

`isvalid(geom )` -- check if multipoint, linestring, polygon, multilinestring, multipolygon is valid

IsRing

`isring(geom )` -- check if linestring is a ring

IsPolygonCCW

`ispolygonccw(geom )` -- check if the outer ring is counter-clock-wise, inner rings clock-wise

IsPolygonCW

`ispolygoncw(geom  )` -- check if the outer ring is clock-wise, inner rings counter-clock-wise

OuterRing

`outerring(polygon)` -- the outer ring as linestring of a polygon

OuterRings

`outerrings(mpolygon)` -- the outer rings as multilinestring of a multipolygon

InnerRings

`innerrings(polygon)` -- the inner rings as multilinestring of a polygon or multipolygon

RingN

`ringn(polygon,n)` -- the n-th ring as linestring of a polygon. n is 1-based

InnerRingN

`innerringn(polygon,n)` -- the n-th inner ring as linestring of a polygon. n is 1-based

PolygonN

`polygonn(multipgon,n)` -- the n-th polygon of a multipolygon. n is 1-based

Unique

`unique(geom)` -- geom with consecutive duplicate points removed

Union

`union(geom1,geom2)` -- union of two geoms. Polygon outer ring should be counter-clockwise.

Collect

`collect(geom1,geom2)` -- collection of two geometric shapes.

ToPolygon

`topolygon(geom)` -- converting square, rectangle, circle, ellipse, triangle to polygon

Text

`text(geom)` -- text string of a geometry shape

Difference

`difference(geom1,geom2)` -- geom1 minus the common part of geom1 and geom2

SymDifference

`symdifference(geom1,geom2)` -- geom1+geom2 minus the common part of geom1 and geom2 (symmetric difference)

IsConvex

`isconvex(pgon)` -- check if the outer ring of a polygon is convex

Interpolate

`interpolate(lstr,frac)` -- the point on linestring lstr where line length from beginning to the point is equal to frac, which is between 0 and 1.

For example: `select interpolate(lstr, 0.5) from mylines;`

LineSubstring

`linesubstring(lstr,startfrac,endfrac)` – the substring of linestring lstr where the substring starts at startfrac and ends at endfrac. 0.0 <= starfrac <= endfrac <= 1.0.

LocatePoint

`locatepoint(lstr,point)` – fraction of length where a point on linestring is closest to a given point.

For example: `select locatepoint(lstr, point(100 200)) from mylines;`

AddPoint

`addpoint(lstr,point)` – add a point at the end of a linestring.

`addpoint(lstr,point,position)` – add a point at the position of a linestring. Position is 1-based.

For example:

```
select addpoint(lstr3, point3d(100 200 300)) from mylines3d;
```

SetPoint

`setpoint(lstr,point,position)` – set or replace a point at the position of a linestring. Position is 1-based.

For example:

```
select setpoint(lstr3, point3d(100 200 300), 3) from mylines3d;
```

RemovePoint

`removepoint(lstr,position)` – remove a point at the position of a linestring. Position is 1-based.

For example:

```
select numpoints(removepoint(lstr3, 3)) from mylines3d;
```

Reverse

`reverse(geom)` – reverse the order of points in line, linestring, polygon, and multipolygon.

For example:

```
select reverse(lstr) from mylines3d;
```

Scale

`scale(geom,facor)` – scale the coordinates of geom by a factor.

`scale(geom,xfactor,yfactor)` – scale the x-y coordinates of 2D geom by two factors.

`scale(geom,xfactor,yfactor,zfactor)` – scale a 3D geom by three factors.

For example: `select scale(ls, 0.5, 0.8) from mylines;`

ScaleAt

`scaleat(geom,point,fac)` – scale the coordinates of geom by a factor relative to a point.

`scaleat(geom,point,xfac,yfac)` – scale 2D geom by two factors relative to a point.

`scaleat(geom,point,xfac,yfac,zfac)` – scale a 3D geom by three factors from a point.

For example:

```
select scaleat(ls, point(100 100), 0.5, 0.8) from mylines;
```

ScaleSize

`scalesize(geom,fac)` – scale the size of vector geom by a factor relative to self-center.

`scalesize(geom,xfac,yfac)` – scale 2D vector geom by two factors relative to self-center

`scalesize(geom,xfac,yfac,zfac)` – scale a 3D vector geom by three factors.

For example:

```
select scalesize(s2, 0.5, 0.8) from mysquare;
```

Translate

`translate(geom,dx,dy)` – translate the position of 2D geom by dx in X and dy in Y axis.

157

`translate(geom,dx,dy,dz)` – translate the position of 3D geom by dx, dy, and dz.

For example:

```
select translate(s2, 0.5, 0.8) from mysquare;
```

TransScale

`transscale(geom,dx,dy,fac)` – translate 2D geom by dx and dy and then scale it by fac.

`transscale(geom,dx,dy,xfac,yfac)` – translate geom and then scale it by xfac and yfac.

`transscale(geom,dx,dy,dz,xfac,yfac,zfac)` – translate 3D geom and scale.

For example:

```
select transscale(s2, 100, 200, 2.0, 3.0) from mysquare;
```

Rotate

`rotate(geom,N)` – rotate 2D geom by N degrees counter-clock-wise around point(0 0).

`rotate(geom,N, 'degree')` – rotate 2D geom by N degrees counter-clock-wise.

`rotate(geom,N, 'radian')` – rotate 2D geom by N radians counter-clock-wise.

For example:

```
select rotate(s, 90, 'degree') from mysquare;
```

RotateSelf

`rotateself(geom,N)` – rotate 2D geom by N degrees counter-clock-wise around self-center.

`rotateself(geom,N, 'degree')` – rotate 2D geom by N degrees counter-clock-wise.

`rotateself(geom,N, 'radian')` – rotate 2D geom by N radians counter-clock-wise.

For example:

```
select rotateself(s, 90, 'degree') from mysquare;
```

RotateAt

`rotateat(geom,N, 'degree',x,y)` – rotate 2D geom by N degrees around point(x y)

`rotateat(geom,N, 'radian',x,y)` – rotate 2D geom by N radians around point(x y)

For example:

```
select rotateat(s, 90, 'degree', 200, 300) from mysquare;
```

Affine

`affine(geom,a,b,d,e,dx,dy)` – apply affine transformation of 2D geom.

`affine(geom,a,b,c,d,e,f,g,h,i,dx,dy)` – apply affine transformation of 3D geom.

In affine transformation:

2D:

```
newx = a*x + b*y + dx
newy = d*x + e*y + dy
```

3D:

```
newx = a*x + b*y + c*z + dx
newy = d*x + e*y + f*z + dy
newz = g*x + h*y + i*z + dz
```

For example:

```
select affine(ls, 10, 20, 30, 40, 100, 200 ) from mylines;
```

Voronoi Polygons

voronoipolygons(mpoint)   -- find Voronoi polygons from multipoints

voronoipolygons(mpoint,tolerance) -- find Voronoi polygons from multipoints with tolerance

voronoipolygons(mpoint,tolerance,bbox) -- find Voronoi polygons from multipoints with tolerance and bounding box. Default bounding box is 30% larger than the bounding box of all the points.

For example:

```
    select voronoipolygons(p, 10, bbox(-100, -100, 200, 200) ) as vor
from mypoints;

    select numpolygons(voronoipolygons(p, 10, bbox(-100, -100, 200,
200) )) as np from mypoints;
```

Voronoi Lines

voronoilines(mpoint)   -- find Voronoi linestrings from multipoints

voronoilines(mpoint,tolerance) -- find Voronoi linestrings from multipoints with tolerance

voronoilines(mpoint,tolerance,bbox) -- find Voronoi linestrings from multipoints with tolerance and bounding box. Default bounding box is 30% larger than the bounding box of all the points.

For example:

```
    select voronoilines(p, 10, bbox(-100, -100, 200, 200) ) as vor
from mypoints;
```

Delaunay Triangles

delaunaytriangles(mpoint) -- find Delaunay triangles from multipoints

delaunaytriangles(mpoint,tolerance) -- find Delaunay triangles from multipoints with a tolerance

For example:

```
    select delaunaytriangles(p) as dt from mypoints;
```

GeoJson

geojson(geom)   -- GeoJSON string of geom

geojson(geom,N) -- GeoJSON string of geom, receiving maximum of N points (default 3000).

geojson(geom,N,n) -- GeoJSON string of geom, receiving maximum of N points, n sample points on 2D vector shapes.

For example:

```
select geojson(p) as js from mypolygons;

select geojson(p,1000) as js from mypolygons;
```


ToMultipoint

tomultipoint(geom)   -- converting geom to multipoint

tomultipoint(geom,N) -- converting geom to multipoint. N is number of points sent to client

For example:

```
select tomultipoint(p) as mt from mypolygons;
```


WKT (Well Known Text)

wkt(geom)   -- display geom as WKT (Well Known Text) string

For example:

```
select wkt(p) from mypolygons;
```


MinimumBoundingCircle

minimumboundingcircle(geom)   -- minimum bounding circle of 2D geom

For example:

```
select minimumboundingcircle(p) from mypoints;
```

MinimumBoundingSphere

minimumboundingsphere(geom)   -- minimum bounding sphere of 3D geom

For example:

```
select minimumboundingsphere(p) from mypoints;
```

IsOnLeft

isonleft(geom1,geom2)   -- detects if geom1 is on the left of geom2, for linear shapes only.

For example:

```
select isonleft(point, linestr) from mylines;
```

IsOnRight

isonright(geom1,geom2)   -- detects if geom1 is on the right of geom2, for linear shapes

For example:

```
select isonright(lstr, linestr) from mylines;
```

LeftRatio

leftratio(geom1,geom2)   -- ratio of geom1 on the left of geom2, for linear shapes

For example:

```
select leftratio(point, linestr) from mylines;
```

RightRatio

rightratio(geom1,geom2) -- ratio of geom1 on the right of geom2, for linear shapes

For example:

```
select rightratio(point, linestr) from mylines;
```

KNN (K Nearest Neighbor)

knn(geom, point, K) -- K nearest neighbors on geom to point

knn(geom, point, K,min,max) -- K nearest neighbors on geom to point between distance min and max

For example:

```
select knn(linestr, point(30 40)) from mylines;
select knn(linestr3d, point3d(30 40 50), 10, 100) from mylines3d;
```

MetricN

metricn(geom) -- all metrics of vector shape geom (m1#m2#m3#...)

metricn(geom,N) -- metrics of N-th point of raster shapes. For vector shapes, the N-th metric.

metricn(geom,N,m) -- metric of N-th point, m-th metric for raster shapes.

For example:

```
select metricn( mysquare ) from squares;
select metricn( mysquare, 3 ) from squares;
select metricn( lstr, 3, 1 ) from linestrings;
```

Spatial Index

Indexes can be built and maintained for spatial data like other types of data. For example,

```
create index linestr3d1_idx1 on linestr3d1(b, ls2);
```

Data on the index can be selected using where clauses as regular store.

Time Series Data Management

JaguarDB Time Series

Normally time series is a series of data points indexed in time order. Time series data plays a crucial role in a wide range of AI applications due to its sequential and temporal nature. Time series data consists of observations collected over successive time intervals, such as daily stock prices, hourly sensor readings, monthly temperature records, and more. Leveraging time series data can provide valuable insights and enable various AI-driven applications. In JaguarDB, the time series has a different meaning: it is both a sequence of data points and a series of tick stores holding aggregated data values at specified time spans. For example, a time series store in JaguarDB can have a base store storing data points in time order, and tick stores such as 5-minute, 15-minute, hourly, daily, weekly, monthly stores to store aggregated data within these time spans.

Tick table		
Tick table3		
Tick table2		
Tick table1		
Base table	Time series data points	
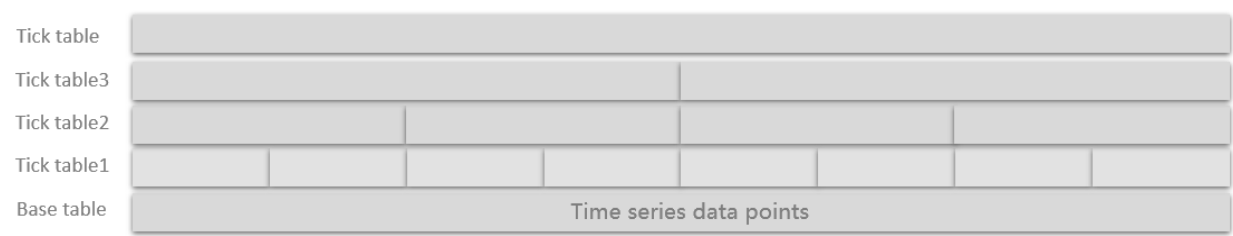
When a time series store is created, the tick stores are created automatically. The tick stores are used to quickly query aggregated data without doing lengthy computations. As a result, queries for aggregated values in different time periods is extremely fast.

Creating Time Series stores

The following formats describe commands to create a time series store:

```
 create store timeseries(TICK:RETENTION, TICK:RETENTION, …|BASERENTION)
BASEstore (key: KEYCOL1, KEYCOL2, …, value: col rollup VTYPE, …);
```

Where:

TICK:RETENTION specifies a tick type and retention period of the tick store;

BASERENTION represents the retention period of the base store;

BASEstore is the name of the base store;

KEYCOL1, KEYCOL2, … are the key columns in the base store;

Rollup specifies the columns whose values will be rolled up to the tick stores;

VTYPE is the type of the column to be rolled up.

The TICK keyword starts with a number and a period type. For example, 15s means a tick store of 15 seconds; 30m means a tick store of 30 minutes.

The letter 's' indicates TICKs in seconds.

The letter 'm' indicates TICKs in minutes.

The letter 'h' indicates TICKs in hours.

The letter 'd' indicates TICKs in days.

The letter 'w' indicates TICKs in weeks.

The letter 'M' indicates TICKs in months.

The letter 'M' indicates TICKs in months.

The letter 'q' indicates TICKs in quarters.

The letter 'y' indicates TICKs in years.

The letter 'D' indicates TICKs in decades.

Valid TICKs in seconds scale include: 1s, 2s, 3s, 5s, 6s, 10s, 12s, 15s, 20s, 30s.

Valid TICKs in minutes include: 1m, 2m, 3m, 5m, 6m, 10m, 12m, 15m, 20m, 30m.

Valid TICKs in hours include: 1h, 2h, 3h, 4h, 6h, 8h, 12h.

Valid TICKs in days include: 1d, 2d, 3d, 4d, 5d, 6d, 7d, 10d, 15d.

Valid TICKs in weeks include: 1w, 2w, 3w, 4w.

Valid TICKs in months include: 1M, 2M, 3M, 4M, 6M.

Valid TICKs in quarters include: 1q, 2q.

Valid TICKs in years can be any number of years.

Valid TICKs in decades can be any number of decades.

Multiple TICKs are allowed in the same TICK group. For example, you can have 5m and 15m stores, and 1d and 10d tick stores.

The format for the RETENTION is the same as the TICK format, except that it can have any number of retention periods. The RETENTION specifies how long the data points in the base store should be kept. Examples of RETENTION are 15d, 1M, 3M, 1y, etc. If no RETENTION is provided, the data points in the tick store are not deleted. If the retention period is passed, old data will be deleted from the tick stores.

The BASERETNYTION specifies how long the data points in the base store should persist. Data points that are older than the retention period are deleted frequently. If no BASERETENTION is provided, the data points in the base store will not be deleted.

A rollup column in a base store indicates that its value will be rolled to the tick stores. In the tick stores the last stored value of the rollup column is saved from the base store. In addition, aggregated values of 'sum', 'min', 'max', 'avg', 'var' of the column are computed and stored in the tick stores.

The type 'sum' indicates that the rollup column in the base store is aggregated into the tick store by taking the cumulative value of the column.

The type 'min' indicates that the rollup column in the base store is aggregated to the tick store by taking the minimum value of the column.

The type 'max' indicates that the rollup column in the base store is aggregated to the tick store by taking the maximum value of the column.

The type 'avg' indicates that the rollup column in the base store is aggregated to the tick store by taking the average value of the column.

The type 'var' indicates that the rollup column in the base store is aggregated to the tick store by taking the variance of the column. The variance is the squared value of the standard deviation.

A tick store has the following columns, which are called the heap columns:

col::sum -- represents the total sum of the column 'col';

col::min -- represents the minimum value of the column 'col';

col::max -- represents the maximum value of the column 'col';

col::avg -- represents the average of the column 'col';

col::var -- represents variance value of the column 'col';

The standard deviation of a column can be obtained by taking the square root of the variance. All these statistical values are computed automatically during data ingestion for quick data analysis.

The heap columns hold statistical values of a rollup column. A rollup column is the column on which the aggregating operations are carried out.

VTYPE is the type of the column to be rolled up. The type can only of numerical types: tinyint, smallint, int, bigint, float, and double. In the tick stores, the integer types become bigint type, and the float or double types become double type. String, date, time, location or other columns cannot be rolled up to the tick stores.

The base store can include columns of any type, whose values are not rolled up to the tick stores if they are not marked with the keyword 'rollup'. Only the columns with the rollup property are rolled up to the tick stores. Without any retention specifications, the create command looks like this:

```
create store timeseries(TICK, TICK, …)
BASEstore (key: KEYCOL1, KEYCOL2, …, value: col rollup VTYPE, …);
```

It is possible to provide or omit the retention for tick stores and base store independently. That is, one store (base or tick able) may have a retention while others may not have a retention.

BASEstore is the name for the base store. The tick stores will have a name that include the name of the base store and the TICK type. For example, if the timeseries has 15m and 1d, then the store name for 15m TICK will be BASEstore@15m; the store name for the 1d TICK

will be BASEstore@1d. The tick stores have the key columns and the rollup columns from the base store. For example, if the base store has a name "traffic", then the tick stores will have names "traffic@15m" and "traffic@1d" which all can be queried directly for time series data analysis.

Examples of Time Series stores

Food Delivery Time Series

The following command will create a base store named "delivery", and two ticks:

```
create store timeseries(1M:1y,1y)
delivery (key: ts timestamp, courier char(32), customer char(32),
          value: meals rollup bigint, addr char(128) );
```

The base store "delivery" has no retention, so its records are persisted forever unless explicitly deleted by the user. One TICK is of monthly (M), having a retention period of one year (y). Another TICK is of yearly (y) with no retention (kept forever). The column 'meals' will be rolled up to ticks '1M' and '1y', but the column 'addr' is not rolled up to the ticks. Key columns are always rolled up to the tick stores.

Traffic Monitoring Time Series

The following command will create a base store named "traffic", and five ticks:

```
create store timeseries(15m:3h,1h:48h,1d:3M,1q|3y)
traffic (key: ts timestamp, line char(32),
         value: volume rollup bigint,
                driver char(32)
);
```

The base store "traffic" has a retention of 3 years, so its records are persisted for 3 years and any older records will be deleted. The column 'volume' is used to collect the traffic volume

since the last observation time. The value of this column will be rolled up to all the tick stores. One tick '15m' is to aggregate data for every 15 minutes, persisted for a total of 3 hours. Tick '1h' is to aggregate passengers by hourly, with retention of 48 hours. Tick '1d' stores aggregated data from the base store in a daily schedule, persisting for 3 months. Tick '1q' stores quarterly (3 months) aggregated data, without any retention (meaning persisting forever).

In the traffic base store, timestamp 'ts' is the leading key, line (which can be a bus line, train line, or any transportation vehicle line number) is the second key. The key columns will all be rolled up to the tick stores for fast data searching. The column 'volume' is also rolled up to the tick stores as data of interest for time series data analysis. The driver's name represented by the column 'driver' will not be present in any of the ticks. Except the key columns, columns that are not marked 'rollup' will not be present in any of the ticks. Importantly, only numeric columns (such as int, bigint, float, double) can be rolled up to the ticks. Non-numeric values cannot be rolled up to the tick stores, because it does not make sense to aggregate non-numeric fields.

IoT Sensor Time Series

JaguarDB can manage data generated by IoT (Internet of Things) sensors. To monitor individual sensors and their captured data, the following time series store structure can be used:

```
create store timeseries(5m:1d,1h:48h,1d:3M,1M:20y|5y)
sensorstat (key: sensorID char(16), ts timestamp,
            value: temperature rollup float,
                   pressure rollup float,
                   windspeed rollup float,
                   rpm  rollup float,
                   fuel rollup float,
                   model char(16),
                   type  char(16)
);
```

The base store "sensorstat" has a retention of 5 years, so its records are persisted for 5 years and older records will be deleted. The column 'temperature' is used to measure the temperature at the location of the sensor. The value of this column will be rolled up to the tick stores. The column 'pressure' measures the pressure at the location of the sensor. The value of this column will also be rolled up to the tick stores. The column 'windspeed' measures the wind speed at the location of the sensor. The value of this column will also be rolled up to the tick stores. The column 'rpm' is used to measure the revolutions per minute (RPM) of an engine at the location of the sensor. The value of this column will be rolled up to the tick stores. The column 'fuel' measures fuel consumption at the location of the sensor. The value of this column will be rolled up to the tick stores. The columns 'model' and 'type' record the model and type of the sensor or the device that the sensor is equipped for. The last two columns, however, are not rolled up to the ticks.

The tick stores will have keys and rollup columns in the base store, and the aggregated heap columns. For example, the tick store 'sensorstat@1d' has the following structure:

```
store test.sensorstat@1d|3M
(
  key:
          sensorid char(16),
          ts datetimesec,
  value:
          temperature float(36.6),
          temperature::sum double(40.10),
          temperature::min double(40.10),
          temperature::max double(40.10),
          temperature::avg double(40.10),
          temperature::var double(40.10),
          pressure float(36.6),
          pressure::sum double(40.10),
          pressure::min double(40.10),
          pressure::max double(40.10),
```

```
        pressure::avg double(40.10),

        pressure::var double(40.10),

        windspeed float(36.6),

        windspeed::sum double(40.10),

        windspeed::min double(40.10),

        windspeed::max double(40.10),

        windspeed::avg double(40.10),

        windspeed::var double(40.10),

        rpm float(36.6),

        rpm::sum double(40.10),

        rpm::min double(40.10),

        rpm::max double(40.10),

        rpm::avg double(40.10),

        rpm::var double(40.10),

        fuel float(36.6),

        fuel::sum double(40.10),

        fuel::min double(40.10),

        fuel::max double(40.10),

        fuel::avg double(40.10),

        fuel::var double(40.10),

        counter bigint DEFAULT '1',

        spare_ char(386),

);
```

In this time series, since the sensorID is the first key, looking up various data associated with a sensor is very fast. The ticks contain aggregate values in different time windows so that data analysis in various time windows can be completed very quickly without conducting full scan of stores nor doing complex computations.

Base store and Ticks

In JaguarDB, a base store stores the detailed time series data. A tick (or a tick store) stores aggregated data from the base store according to the predefined tick length. When a time series store is created, the associated tick stores are automatically created. When time series data is written to the base store, the aggregated data is also automatically written to all the tick stores. User can then directly query the tick stores for data in different time windows.

For example, in the above `sensorstat` time series store, when this store is created, the following tick stores are also created:

```
sensorstat@5m
sensorstat@1h
sensorstat@1d
sensorstat@1M
```

They can be show in the jag client program:

```
jaguar:mydb>  desc sensorstat;
store timeseries(5m:1d,1h:48h,1d:3M,1M:20y|5y) test.sensorstat
(
```

```
  key:
    sensorid char(16),
    ts timestamp,
  value:
    temperature rollup float(36.6),
    pressure rollup float(36.6),
    windspeed rollup float(36.6),
    rpm rollup float(36.6),
    fuel rollup float(36.6),
    model char(16),
    type char(16),
    spare_ char(76),
);

jaguar:mydb>  desc sensorstat@5m;
store test.sensorstat@5m|1d
(
  key:
    sensorid char(16),
    ts datetimesec,
  value:
    temperature float(36.6),
    temperature::sum double(40.10),
    temperature::min double(40.10),
    temperature::max double(40.10),
    temperature::avg double(40.10),
    temperature::var double(40.10),
    pressure float(36.6),
    pressure::sum double(40.10),
    pressure::min double(40.10),
    pressure::max double(40.10),
    pressure::avg double(40.10),
```

```
        pressure::var double(40.10),

        windspeed float(36.6),

        windspeed::sum double(40.10),

        windspeed::min double(40.10),

        windspeed::max double(40.10),

        windspeed::avg double(40.10),

        windspeed::var double(40.10),

        rpm float(36.6),

        rpm::sum double(40.10),

        rpm::min double(40.10),

        rpm::max double(40.10),

        rpm::avg double(40.10),

        rpm::var double(40.10),

        fuel float(36.6),

        fuel::sum double(40.10),

        fuel::min double(40.10),

        fuel::max double(40.10),

        fuel::avg double(40.10),

        fuel::var double(40.10),

        counter bigint DEFAULT '1',

        spare_ char(386),

);


store test.sensorstat@1h|48h

(

  key:

    sensorid char(16),

    ts datetimesec,

  value:

    temperature float(36.6),

    temperature::sum double(40.10),

    temperature::min double(40.10),
```

```
        temperature::max double(40.10),

        temperature::avg double(40.10),

        temperature::var double(40.10),

        pressure float(36.6),

        pressure::sum double(40.10),

        pressure::min double(40.10),

        pressure::max double(40.10),

        pressure::avg double(40.10),

        pressure::var double(40.10),

        windspeed float(36.6),

        windspeed::sum double(40.10),

        windspeed::min double(40.10),

        windspeed::max double(40.10),

        windspeed::avg double(40.10),

        windspeed::var double(40.10),

        rpm float(36.6),

        rpm::sum double(40.10),

        rpm::min double(40.10),

        rpm::max double(40.10),

        rpm::avg double(40.10),

        rpm::var double(40.10),

        fuel float(36.6),

        fuel::sum double(40.10),

        fuel::min double(40.10),

        fuel::max double(40.10),

        fuel::avg double(40.10),

        fuel::var double(40.10),

        counter bigint DEFAULT '1',

        spare_ char(386),

    );
```

```
Jaguar:mydb> desc sensorstat@1M;

store test.sensorstat@1M|20y
(
  key:
    sensorid char(16),
    ts datetimesec,
  value:
    temperature float(36.6),
    temperature::sum double(40.10),
    temperature::min double(40.10),
    temperature::max double(40.10),
    temperature::avg double(40.10),
    temperature::var double(40.10),
    pressure float(36.6),
    pressure::sum double(40.10),
    pressure::min double(40.10),
    pressure::max double(40.10),
    pressure::avg double(40.10),
    pressure::var double(40.10),
    windspeed float(36.6),
    windspeed::sum double(40.10),
    windspeed::min double(40.10),
    windspeed::max double(40.10),
    windspeed::avg double(40.10),
    windspeed::var double(40.10),
    rpm float(36.6),
    rpm::sum double(40.10),
    rpm::min double(40.10),
    rpm::max double(40.10),
    rpm::avg double(40.10),
    rpm::var double(40.10),
    fuel float(36.6),
```

```
    fuel::sum double(40.10),

    fuel::min double(40.10),

    fuel::max double(40.10),

    fuel::avg double(40.10),

    fuel::var double(40.10),

    counter bigint DEFAULT '1',

    spare_ char(386),

);
```

The keyword "20y" in "store test.sensorstat@1M|20y" means the retention period for tick sensorstat@1M is 20 years. The string "test" means the store is created in the "test" database. It should be noticed that the tick stores contain only the key columns and the rollup columns.

In addition, there is an extra column 'counter' added automatically in the tick stores. It tracks the number of records in a period of time. In a tick store, the datetime or timestamp key column will not have the detailed date time value. Instead. The data time column will have values at the start of the tick. For example, in an hour tick store, the date time key column will not have minutes and seconds. It will have time values that are rounded to hours, e.g. "2022-10-12 13:00:00", "2022-08-15 16:00:00".

Inserting Data into Time Series stores

A user can insert data into the base stores just like any other stores. Data in tick stores will be automatically prepared and inserted by JaguarDB. The following example shows how to insert data into the base store:

```
insert into sensorstat (sensorid, temperature, pressure, windspeed, rpm, fuel, model,
type ) values ( 'drone1-sid1', '20.0', '35.5', '30.2', '1300', '1.3', 'AA212', 'DH' );

insert into sensorstat (sensorid, temperature, pressure, windspeed, rpm, fuel, model,
type ) values ( 'drone1-sid1', '20.5', '35.8', '30.7', '1320', '1.5', 'AA212', 'DH' );

insert into sensorstat (sensorid, temperature, pressure, windspeed, rpm, fuel, model,
type ) values ( 'drone1-sid2', '21.0', '35.7', '30.8', '1304', '1.2', 'AA213', 'DH' );

insert into sensorstat (sensorid, temperature, pressure, windspeed, rpm, fuel, model,
type ) values ( 'drone2-sid1', '22.0', '36.4', '30.3', '1404', '2.2', 'AB213', 'DF' );
```

Data from different sensors which may be attached to different devices can be stored in the base store 'sensorstat'. Here the key column 'ts' is omitted and a default client's local time will be inserted automatically.

When the base store is populated, the four tick stores are automatically populated:

```
sensorstat@5m  -- will have data aggregated every 5 minutes

sensorstat@1h  -- will have data aggregated every hour

sensorstat@1d  -- will have data aggregated every day

sensorstat@1M  -- will have data aggregated every month
```

Reading Data From Time Series stores

Reading From Base store and Tick stores

A user can read data from the base stores as well as the tick stores just like any other stores. The following example shows how to read data from the base store:

```
>  select sensorid, ts, temperature, pressure, rpm from sensorstat where
sensorid='drone1-sid1';

sensorid=[drone1-sid1] ts=[2021-03-25 03:31:49.801081] temperature=[20.0]
pressure=[35.5] rpm=[1300.0]


> select sensorid, ts, temperature, pressure, rpm from sensorstat where
sensorid='drone1-sid2';

sensorid=[drone1-sid2] ts=[2021-03-25 03:42:41.462460] temperature=[21.0]
pressure=[35.7] rpm=[1304.0]
```

The following example shows how to read aggregated data from the tick stores:

```
> select sensorid, ts, temperature::avg, pressure::avg, rpm::max from sensorstat@1d
where sensorid='drone1-sid1';

sensorid=[drone1-sid1] ts=[2021-03-25 01:00:00] temperature::avg=[20.0]
pressure::avg=[35.5] rpm::max=[1300.0]
```

```
> select sensorid, ts, temperature::avg, pressure::avg, rpm::max from sensorstat@1d
where sensorid='drone1-sid2';

sensorid=[drone1-sid2] ts=[2021-03-25 01:00:00] temperature::avg=[21.0]
pressure::avg=[35.7] rpm::max=[1304.0]


> select sensorid, ts, temperature::avg, pressure::avg, rpm::min, rpm::max from
sensorstat@1d where sensorid='*';

sensorid=[*] ts=[2021-03-25 01:00:00] temperature::avg=[20.875] pressure::avg=[35.85]
rpm::min=[1300.0] rpm::max=[1404.0]
```

Note that the condition sensorid='*' can be used to select data for all the possible values of the sensorid column. Only on key columns can a user apply the "*" condition. This predicate is described below in detail.

Grouping Data In Windows

The window(length, column) function takes a datetime (including time of different granularities) column and breaks the column into time windows. The length argument represents the length of time for the windows, and column is a store column name. The window column must have a time type. The following examples demonstrate how to make queries based on the time windows.

```
select pickup_datetime, window(5m, pickup_datetime)
from rides
where date(pickup_datetime)='2021-02-11';
```

The above query breaks the `pickup_datetime` into a series of 5 minutes intervals and gets the start time of the intervals in the day of '2021-02-11'. The window() function can appear anywhere in the query statement but only one is required.

The get aggregated values of numeric columns in a store, the `group by` clause can be used to get the aggregated values in each of the time windows:

```
select pickup_datetime, window(5m, pickup_datetime), avg(total_amount) as
avg_total_amount
from rides
where date(pickup_datetime)='2021-02-11'
group by pickup_datetime;
```

The window function creates time windows of 5 minutes based on the column 'pickup_datetime'. The average of value of total_amount is taken in the 5 minute windows by the `group by` method.

All Key Values in Tick store

JaguarDB precomputes values for the '*' condition of every key column in the tick store. Also, all combinations of the '*' value of each key column are calculated. For example: If there are three key columns (named A, B, C for instance), then the following combinations are prepared:

Key A	Key B	Key C	Value Rollup Columns
*	B	C	V1, V2, V3, ...
A	*	C	V1, V2, V3, ...
A	B	*	V1, V2, V3, ...
A	*	*	V1, V2, V3, ...
*	B	*	V1, V2, V3, ...
*	*	C	V1, V2, V3, ...
*	*	*	V1, V2, V3, ...

Selection of data can choose all combinations of the key values. For example, the following select patterns can be applied:

```
select * from tickstore where A='*' and B='value-of-B';

select * from tickstore where A='*' and B='value-of-B' and C='value-of-C';

select * from tickstore where A='value-of-A' and B='*' and C='value-of-C';

select * from tickstore where A='*' and B='*' and C='value-of-C';

select * from tickstore where A='*' and B='*';

select * from tickstore where A='*' and B='*' and C='*';
```

A condition with value of '*' outputs only a single record, instead of multiple records of all possible values. The following query:

```
select * from tickstore where A='*' and B='*';
```

181

may outputs records of all possible values of key C. However, the following query:

```
select * from tickstore where A='*' and B='*' and C='*';
```

will outputs a single record, if such data exists in the store. The column values are aggregated in runtime under the key entry of '*' for all possible values of the key column. Queries for aggregation values are fast because only a single record is read to retrieve the aggregated data without scanning the stores to get the result.

Indexes of Time Series stores

Automatically managing the aggregation tick stores in JaguarDB is not the end of the story. JaguarDB also enables a user to create indexes on the timeseries stores for flexible queries, then JaguarDB will automatically create index records and apply them on the tick stores. The following paragraphs demonstrate how indexes are created and used. For example, suppose we have inserted data records into the base store 'delivery':

```
insert into delivery ( courier, customer, meals, addr ) values ( 'QDEX', 'JohnDoe',
'3', '110 A Street, CA 90222' );

insert into delivery ( courier, customer, meals, addr ) values ( 'QDEX', 'JaneDoe',
'5', '110 B Street, CA 90001' );

insert into delivery ( courier, customer, meals, addr ) values ( 'QSEND', 'MaryAnn',
'3', '100 C Street, CA 92220' );

insert into delivery ( courier, customer, meals, addr ) values ( 'QSEND', 'PaulD',
'12', '550 Ivy Road, CA 90221' );
```

```
> select * from delivery;

ts=[2021-03-14 20:51:06.457043] courier=[QDEX] customer=[JohnDoe] meals=[3] addr=[110
A Street, CA 90222]

ts=[2021-03-14 20:51:41.282601] courier=[QDEX] customer=[JaneDoe] meals=[5] addr=[110
B Street, CA 90001]

ts=[2021-03-14 20:52:11.605846] courier=[QSEND] customer=[MaryAnn] meals=[3] addr=[100
C Street, CA 92220]

ts=[2021-03-14 20:52:36.826472] courier=[QSEND] customer=[PaulD] meals=[12] addr=[550
Ivy Road, CA 90221]
```

Then if we read the data from the tick stores, we see column statistics are shown:

```
> select * from delivery@1M;

ts=[2021-03-01 00:00:00] courier=[*] customer=[*] meals=[12] meals::sum=[23] meals::min=[3]
meals::max=[12] meals::avg=[6] meals::var=[44] counter=[4]

ts=[2021-03-01 00:00:00] courier=[*] customer=[JaneDoe] meals=[5] meals::sum=[5] meals::min=[5]
meals::max=[5] meals::avg=[5] meals::var=[0] counter=[1]

ts=[2021-03-01 00:00:00] courier=[*] customer=[JohnDoe] meals=[3] meals::sum=[3] meals::min=[3]
meals::max=[3] meals::avg=[3] meals::var=[0] counter=[1]

ts=[2021-03-01 00:00:00] courier=[*] customer=[MaryAnn] meals=[3] meals::sum=[3] meals::min=[3]
meals::max=[3] meals::avg=[3] meals::var=[0] counter=[1]

ts=[2021-03-01 00:00:00] courier=[*] customer=[PaulD] meals=[12] meals::sum=[12] meals::min=[12]
meals::max=[12] meals::avg=[12] meals::var=[0] counter=[1]

ts=[2021-03-01 00:00:00] courier=[QDEX] customer=[*] meals=[5] meals::sum=[8] meals::min=[3]
meals::max=[5] meals::avg=[4] meals::var=[0] counter=[2]

ts=[2021-03-01 00:00:00] courier=[QDEX] customer=[JaneDoe] meals=[5] meals::sum=[5]
meals::min=[5] meals::max=[5] meals::avg=[5] meals::var=[0] counter=[1]

ts=[2021-03-01 00:00:00] courier=[QDEX] customer=[JohnDoe] meals=[3] meals::sum=[3]
meals::min=[3] meals::max=[3] meals::avg=[3] meals::var=[0] counter=[1]

ts=[2021-03-01 00:00:00] courier=[QSEND] customer=[*] meals=[12] meals::sum=[15] meals::min=[3]
meals::max=[12] meals::avg=[8] meals::var=[0] counter=[2]

ts=[2021-03-01 00:00:00] courier=[QSEND] customer=[MaryAnn] meals=[3] meals::sum=[3]
meals::min=[3] meals::max=[3] meals::avg=[3] meals::var=[0] counter=[1]

ts=[2021-03-01 00:00:00] courier=[QSEND] customer=[PaulD] meals=[12] meals::sum=[12]
meals::min=[12] meals::max=[12] meals::avg=[12] meals::var=[0] counter=[1]


> select * from delivery@1y;

ts=[2021-01-01 00:00:00] courier=[*] customer=[*] meals=[12] meals::sum=[23] meals::min=[3]
meals::max=[12] meals::avg=[6] meals::var=[44] counter=[4]

ts=[2021-01-01 00:00:00] courier=[*] customer=[JaneDoe] meals=[5] meals::sum=[5] meals::min=[5]
meals::max=[5] meals::avg=[5] meals::var=[0] counter=[1]

ts=[2021-01-01 00:00:00] courier=[*] customer=[JohnDoe] meals=[3] meals::sum=[3] meals::min=[3]
meals::max=[3] meals::avg=[3] meals::var=[0] counter=[1]

ts=[2021-01-01 00:00:00] courier=[*] customer=[MaryAnn] meals=[3] meals::sum=[3] meals::min=[3]
meals::max=[3] meals::avg=[3] meals::var=[0] counter=[1]

ts=[2021-01-01 00:00:00] courier=[*] customer=[PaulD] meals=[12] meals::sum=[12] meals::min=[12]
meals::max=[12] meals::avg=[12] meals::var=[0] counter=[1]

ts=[2021-01-01 00:00:00] courier=[QDEX] customer=[*] meals=[5] meals::sum=[8] meals::min=[3]
meals::max=[5] meals::avg=[4] meals::var=[0] counter=[2]

ts=[2021-01-01 00:00:00] courier=[QDEX] customer=[JaneDoe] meals=[5] meals::sum=[5]
meals::min=[5] meals::max=[5] meals::avg=[5] meals::var=[0] counter=[1]

ts=[2021-01-01 00:00:00] courier=[QDEX] customer=[JohnDoe] meals=[3] meals::sum=[3]
meals::min=[3] meals::max=[3] meals::avg=[3] meals::var=[0] counter=[1]

ts=[2021-01-01 00:00:00] courier=[QSEND] customer=[*] meals=[12] meals::sum=[15] meals::min=[3]
meals::max=[12] meals::avg=[8] meals::var=[0] counter=[2]
```

```
ts=[2021-01-01 00:00:00] courier=[QSEND] customer=[MaryAnn] meals=[3] meals::sum=[3]
meals::min=[3] meals::max=[3] meals::avg=[3] meals::var=[0] counter=[1]

ts=[2021-01-01 00:00:00] courier=[QSEND] customer=[PaulD] meals=[12] meals::sum=[12]
meals::min=[12] meals::max=[12] meals::avg=[12] meals::var=[0] counter=[1]
```

The leading key column is the timestamp in the base and tick stores. If we want to look up records by courier names, we can create an index using the courier column as the leading column in the index:

```
create index delivery_index_courier on delivery(courier, customer, meals);
```

After the create command, the index for the base store can be directly queried:

```
> select * from delivery_index_courier;

courier=[QDEX] customer=[JaneDoe] meals=[5] ts=[2021-03-25 22:42:22.164523]

courier=[QDEX] customer=[JohnDoe] meals=[3] ts=[2021-03-25 22:42:22.163801]

courier=[QSEND] customer=[MaryAnn] meals=[3] ts=[2021-03-25 22:42:22.165190]

courier=[QSEND] customer=[PaulD] meals=[12] ts=[2021-03-25 22:42:22.165899]
```

The above "create index" command creates only an index on the base store, not indexes on the tick stores. Recall that a timeseries store is a cluster of stores that include the base store and a series of tick stores. In this case, only one index is created on the basestore. However, if a user wishes to create a cluster of indexes on the timeseries stores, then the following command can used to create a cluster of indexes:

```
create index delivery_index_courier ticks on delivery(courier, customer, meals);
```

An index named "delivery_index_courier@1M" is created based on the tick store "delivery@1M". Another index named "delivery_index_courier@1y" is created based on the tick store "delivery@1y". The indexes for the tick stores, however, do not contain the heap columns such as meals::min, meals::max. and meals::avg columns, because the index column "meals" is a key in the index. If the meals column is to be treated as a heap column, then the index can be created with the following command:

```
create index deliv_index_cour ticks on delivery(key: courier, customer, value: meals);
```

The above command creates a cluster of indexes that include all the statistical values of the field "meals".

If only certain heap columns were to be tracked by indexes, indexes on tick stores can be created with selected heap columns:

```
create index delivery_index2_courier on delivery@1M(courier, customer, meals::min,
meals::max, meals::sum );

> select * from delivery_index2_courier;

courier=[*] customer=[*] meals::min=[3] meals::max=[12] meals::sum=[23] ts=[2021-03-01 00:00:00]

courier=[*] customer=[JaneDoe] meals::min=[5] meals::max=[5] meals::sum=[5] ts=[2021-03-01
00:00:00]

courier=[*] customer=[JohnDoe] meals::min=[3] meals::max=[3] meals::sum=[3] ts=[2021-03-01
00:00:00]

courier=[*] customer=[MaryAnn] meals::min=[3] meals::max=[3] meals::sum=[3] ts=[2021-03-01
00:00:00]

courier=[*] customer=[PaulD] meals::min=[12] meals::max=[12] meals::sum=[12] ts=[2021-03-01
00:00:00]

courier=[QDEX] customer=[*] meals::min=[3] meals::max=[5] meals::sum=[8] ts=[2021-03-01 00:00:00]

courier=[QDEX] customer=[JaneDoe] meals::min=[5] meals::max=[5] meals::sum=[5] ts=[2021-03-01
00:00:00]

courier=[QDEX] customer=[JohnDoe] meals::min=[3] meals::max=[3] meals::sum=[3] ts=[2021-03-01
00:00:00]

courier=[QSEND] customer=[*] meals::min=[3] meals::max=[12] meals::sum=[15] ts=[2021-03-01
00:00:00]

courier=[QSEND] customer=[MaryAnn] meals::min=[3] meals::max=[3] meals::sum=[3] ts=[2021-03-01
00:00:00]

courier=[QSEND] customer=[PaulD] meals::min=[12] meals::max=[12] meals::sum=[12] ts=[2021-03-01
00:00:00]
```

The query below selects all deliveries from all couriers to the customer 'PaulD':

```
> select * from delivery_index2_courier where courier='*' and
customer='PaulD';

courier=[*] customer=[PaulD] meals::min=[12] meals::max=[12] meals::sum=[12] ts=[2021-03-01
00:00:00]
```

Like any other indexes in JaguarDB, query by the leading column in the index is very fast because it is ordered first in the storage structure of the index.

Delete Data From Time Series

Normally users store time series data and do not expect to delete the data. However, in case a user wishes to delete records in time series stores, the user can execute the delete command. Records in the base stores are removed but records in the tick stores are not deleted. The user however can execute the delete command on a tick store specifically.

For example:

```
delete from delivery where courier='QSEND';

select * from delivery;

ts=[2021-03-25 22:42:22.163801] courier=[QDEX] customer=[JohnDoe] meals=[3] addr=[110
A Street, CA 90222]

ts=[2021-03-25 22:42:22.164523] courier=[QDEX] customer=[JaneDoe] meals=[5] addr=[110
B Street, CA 90001]
```

We can see that records of courier 'QSEND' are deleted in the base store 'delivery', but the tick stores still have them:

```
> select * from delivery@1M;

ts=[2021-03-01 00:00:00] courier=[*] customer=[*] meals=[12] meals::sum=[23]
meals::min=[3] meals::max=[12] meals::avg=[6] meals::var=[44] counter=[4]

ts=[2021-03-01 00:00:00] courier=[*] customer=[JaneDoe] meals=[5] meals::sum=[5]
meals::min=[5] meals::max=[5] meals::avg=[5] meals::var=[0] counter=[1]

ts=[2021-03-01 00:00:00] courier=[*] customer=[JohnDoe] meals=[3] meals::sum=[3]
meals::min=[3] meals::max=[3] meals::avg=[3] meals::var=[0] counter=[1]

ts=[2021-03-01 00:00:00] courier=[*] customer=[MaryAnn] meals=[3] meals::sum=[3]
meals::min=[3] meals::max=[3] meals::avg=[3] meals::var=[0] counter=[1]

ts=[2021-03-01 00:00:00] courier=[*] customer=[PaulD] meals=[12] meals::sum=[12]
meals::min=[12] meals::max=[12] meals::avg=[12] meals::var=[0] counter=[1]

ts=[2021-03-01 00:00:00] courier=[QDEX] customer=[*] meals=[5] meals::sum=[8]
meals::min=[3] meals::max=[5] meals::avg=[4] meals::var=[0] counter=[2]

ts=[2021-03-01 00:00:00] courier=[QDEX] customer=[JaneDoe] meals=[5] meals::sum=[5]
meals::min=[5] meals::max=[5] meals::avg=[5] meals::var=[0] counter=[1]

ts=[2021-03-01 00:00:00] courier=[QDEX] customer=[JohnDoe] meals=[3] meals::sum=[3]
meals::min=[3] meals::max=[3] meals::avg=[3] meals::var=[0] counter=[1]
```

```
ts=[2021-03-01 00:00:00] courier=[QSEND] customer=[*] meals=[12] meals::sum=[15]
meals::min=[3] meals::max=[12] meals::avg=[8] meals::var=[0] counter=[2]

ts=[2021-03-01 00:00:00] courier=[QSEND] customer=[MaryAnn] meals=[3] meals::sum=[3]
meals::min=[3] meals::max=[3] meals::avg=[3] meals::var=[0] counter=[1]

ts=[2021-03-01 00:00:00] courier=[QSEND] customer=[PaulD] meals=[12] meals::sum=[12]
meals::min=[12] meals::max=[12] meals::avg=[12] meals::var=[0] counter=[1]


> select * from delivery@1y;

ts=[2021-01-01 00:00:00] courier=[*] customer=[*] meals=[12] meals::sum=[23]
meals::min=[3] meals::max=[12] meals::avg=[6] meals::var=[44] counter=[4]

ts=[2021-01-01 00:00:00] courier=[*] customer=[JaneDoe] meals=[5] meals::sum=[5]
meals::min=[5] meals::max=[5] meals::avg=[5] meals::var=[0] counter=[1]

ts=[2021-01-01 00:00:00] courier=[*] customer=[JohnDoe] meals=[3] meals::sum=[3]
meals::min=[3] meals::max=[3] meals::avg=[3] meals::var=[0] counter=[1]

ts=[2021-01-01 00:00:00] courier=[*] customer=[MaryAnn] meals=[3] meals::sum=[3]
meals::min=[3] meals::max=[3] meals::avg=[3] meals::var=[0] counter=[1]

ts=[2021-01-01 00:00:00] courier=[*] customer=[PaulD] meals=[12] meals::sum=[12]
meals::min=[12] meals::max=[12] meals::avg=[12] meals::var=[0] counter=[1]

ts=[2021-01-01 00:00:00] courier=[QDEX] customer=[*] meals=[5] meals::sum=[8]
meals::min=[3] meals::max=[5] meals::avg=[4] meals::var=[0] counter=[2]

ts=[2021-01-01 00:00:00] courier=[QDEX] customer=[JaneDoe] meals=[5] meals::sum=[5]
meals::min=[5] meals::max=[5] meals::avg=[5] meals::var=[0] counter=[1]

ts=[2021-01-01 00:00:00] courier=[QDEX] customer=[JohnDoe] meals=[3] meals::sum=[3]
meals::min=[3] meals::max=[3] meals::avg=[3] meals::var=[0] counter=[1]

ts=[2021-01-01 00:00:00] courier=[QSEND] customer=[*] meals=[12] meals::sum=[15]
meals::min=[3] meals::max=[12] meals::avg=[8] meals::var=[0] counter=[2]

ts=[2021-01-01 00:00:00] courier=[QSEND] customer=[MaryAnn] meals=[3] meals::sum=[3]
meals::min=[3] meals::max=[3] meals::avg=[3] meals::var=[0] counter=[1]

ts=[2021-01-01 00:00:00] courier=[QSEND] customer=[PaulD] meals=[12] meals::sum=[12]
meals::min=[12] meals::max=[12] meals::avg=[12] meals::var=[0] counter=[1]
```

How about the records in the index? Are they deleted from the indexes?

```
> select * from delivery_index_courier;

courier=[QDEX] customer=[JaneDoe] meals=[5] ts=[2021-03-25 22:42:22.164523]

courier=[QDEX] customer=[JohnDoe] meals=[3] ts=[2021-03-25 22:42:22.163801]
```

We can see the records in the index for the base store are deleted.

If we delete the records in the tick stores:

```
delete from delivery@1M where courier='QSEND';
delete from delivery@1y where courier='QSEND';
```

then the records in the tick stores and the records in the indexes for the tick stores are deleted:

```
> select * from delivery@1M;
ts=[2021-03-01 00:00:00] courier=[*] customer=[*] meals=[23] counter=[4]
ts=[2021-03-01 00:00:00] courier=[*] customer=[JaneDoe] meals=[5] counter=[1]
ts=[2021-03-01 00:00:00] courier=[*] customer=[JohnDoe] meals=[3] counter=[1]
ts=[2021-03-01 00:00:00] courier=[*] customer=[MaryAnn] meals=[3] counter=[1]
ts=[2021-03-01 00:00:00] courier=[*] customer=[PaulD] meals=[12] counter=[1]
ts=[2021-03-01 00:00:00] courier=[QDEX] customer=[*] meals=[8] counter=[2]
ts=[2021-03-01 00:00:00] courier=[QDEX] customer=[JaneDoe] meals=[5] counter=[1]
ts=[2021-03-01 00:00:00] courier=[QDEX] customer=[JohnDoe] meals=[3] counter=[1]


> select * from delivery@1y;
ts=[2021-01-01 00:00:00] courier=[*] customer=[*] meals=[23] counter=[4]
ts=[2021-01-01 00:00:00] courier=[*] customer=[JaneDoe] meals=[5] counter=[1]
ts=[2021-01-01 00:00:00] courier=[*] customer=[JohnDoe] meals=[3] counter=[1]
ts=[2021-01-01 00:00:00] courier=[*] customer=[MaryAnn] meals=[3] counter=[1]
ts=[2021-01-01 00:00:00] courier=[*] customer=[PaulD] meals=[12] counter=[1]
ts=[2021-01-01 00:00:00] courier=[QDEX] customer=[*] meals=[8] counter=[2]
ts=[2021-01-01 00:00:00] courier=[QDEX] customer=[JaneDoe] meals=[5] counter=[1]
ts=[2021-01-01 00:00:00] courier=[QDEX] customer=[JohnDoe] meals=[3] counter=[1]
```

In general, if the data in a store is deleted, then data in the associated indexes will be deleted.

Truncate Time Series

A base store can be truncated (deleting all records but keeping the store structure) and the indexes associated with the base store will be automatically truncated.

```
truncate store delivery;

select count(*) from delivery;

test.delivery has 0 rows

select count(*) from delivery@1M;

test.delivery@1M has 11 rows

select count(*) from delivery@1y;

test.delivery@1y has 11 rows


select count(*) from delivery_index_courier;

test.delivery_index_courier has 0 rows


select count(*) from delivery_index_courier@1M;

test.delivery_index_courier@1M has 11 rows

select count(*) from delivery_index_courier@1y;

test.delivery_index_courier@1y has 11 rows
```

If a base store is truncated, its tick stores are left untouched. A tick store can be manually truncated and its associated index will be truncated accordingly:

```
truncate store delivery@1M;

select count(*) from test.delivery@1M;

test.delivery@1M has 0 rows

select count(*) from delivery_index_courier@1M;

test.delivery_index_courier@1M has 0 rows


truncate store delivery@1y;

select count(*) from test.delivery@1y;

test.delivery@1y has 0 rows

select count(*) from delivery_index_courier@1y;

test.delivery_index_courier@1y has 0 rows
```

Drop Time Series

If a base store is dropped, all its data are permanently deleted. Also, all its tick stores are dropped, and all associated indexes including the indexes for the base store, and the indexes for the tick stores are all dropped. Be cautious when you want to drop stores.

Space and Time Data Management

The following example illustrates how a user can manage time series data and location-based data in one JaguarDB 'rides' store. The rides store is created by the following command:

```
CREATE store timeseries(5m,30m,1d,1M) rides (
    key:
    pickup_datetime  datetimesec,
    dropoff_datetime datetimesec,
    driver_name      char(16),
    rate_type        char(8),
    payment_type     char(1),

    value:
    passenger_count  rollup int,
    trip_distance    rollup float(8.2),
    pickup_location  point(srid:wgs84),
    dropoff_location point(srid:wgs84),
    fare_amount      rollup float(8.2),
    tip_amount       rollup float(6.2),
    tolls_amount     float(6.2),
    total_amount     rollup float(8.2),
);
```

Here the 'rides' is the base store, and there are four tick stores created for ticks of five minutes, thirty minutes, one day, and one month. Each rollup column will generate five heap columns in the tick stores. Passenger pickup location and drop off location are represented by points having longitude and latitude coordinates in degrees. Data can be inserted by the following example:

```
insert into rides values ( '2021-02-11 09:22:12', '2021-02-11 09:50:42', 'DriverAHM',
'REG', '1', '2', '48.6', point(122.036 37.7), point(122.385 37.622), '56.5', '10.5',
'5.0', '72.0' );

insert into rides values ( '2021-02-11 09:32:12', '2021-02-11 09:58:42', 'DriverJHS',
'HYP', '1', '3', '49.2', point(122.035 37.369), point(122.381 37.621), '73.5', '12.5',
'5.8', '91.8' );



insert into rides values ( '2021-02-12 09:32:12', '2021-02-12 13:50:42', 'DriverAHM',
'REG', '1', '2', '66.8', point(121.8864 37.3382 ), point(122.382 37.622), '96.1',
'20.5', '8.0', '124.6' );
```

With the data we have, we can answer the following questions:

```
(1) How many rides took place on each day?

    select pickup_datetime as day, counter as rides from rides@1d where
    driver_name='*' and rate_type='*' and payment_type='*';

    Answer:
    day=[2021-02-11 00:00:00] rides=[2]
    day=[2021-02-12 00:00:00] rides=[1]

(2) How many rides took place on the day of '2021-02-12'?

    select pickup_datetime as day, counter as rides from rides@1d where
    driver_name='*' and rate_type='*' and payment_type='*' and
    pickup_datetime='2021-02-11 00:00:00';

    Answer:
    day=[2021-02-11 00:00:00] rides=[2]

(3) What is the average fare amount?

    select avg( fare_amount::avg) avg_fare_mount from rides@1M where
    driver_name='*' and rate_type='*' and payment_type='*';

    Answer:
    avg_fare_mount=[75.366667]

(4) What is the average fare amount in February of year 2021?

    select pickup_datetime as month, fare_amount::avg avg_fare_mount from rides@1M
    where driver_name='*' and rate_type='*' and payment_type='*' and
    pickup_datetime='2021-02-01 00:00:00';
```

Answer:
month=[2021-02-01 00:00:00] avg_fare_mount=[75.3666666667]

(5) What is the average fare amount for each driver?

select driver_name, avg(fare_amount::avg) avg_fare_mount from rides@1M where
driver_name != '*' and rate_type='*' and payment_type='*' group by driver_name;

Answer:
driver_name=[DriverAHM] avg_fare_mount=[76.3]
driver_name=[DriverJHS] avg_fare_mount=[73.5]

(6) How many rides took place for each rate type?

select rate_type, sum(counter) rides from rides@1M where rate_type != '*' and
driver_name='*' and payment_type='*' group by rate_type;

Answer:
rate_type=[HYP] rides=[1.0]
rate_type=[REG] rides=[2.0]

(7) What are the monthly average trip distance for all drivers?

select pickup_datetime as month, trip_distance::avg from rides@1M where
rate_type='*' and payment_type='*' and driver_name='*';

Answer:

month=[2021-02-01 00:00:00] trip_distance::avg=[54.8666666667]

(8) What are the monthly average trip distance and maximum average distance for
each driver?

select driver_name driver, pickup_datetime as month, avg(trip_distance::avg)
avg_distance, max(trip_distance::avg) max_avg_distance from rides@1M where
rate_type='*' and payment_type='*' and driver_name != '*' group by driver_name;

Answer:

driver=[DriverAHM] month=[2021-02-01 00:00:00] avg_distance=[57.7] max_avg_distance=[57.7]

driver=[DriverJHS] month=[2021-02-01 00:00:00] avg_distance=[49.2] max_avg_distance=[49.2]

(9) How many rides took place every 5 minutes for the day of '2021-02-11' ?

select pickup_datetime time, counter rides from rides@5m where driver_name='*'
and rate_type='*' and payment_type='*' and pickup_datetime >= '2021-02-11
00:00::00' and pickup_datetime < '2021-02-12 00:00:00' ;

Answer:

time=[2021-02-11 09:20:00] rides=[1]

```
    time=[2021-02-11 09:30:00] rides=[1]
```

(10)   How many rides on the day of '2021-02-11' originated from within 10
     kilometers of Sunnyvale, California in 30 minute buckets?

```
select pickup_datetime as day from rides where distance(pickup_location,
point( 122.035 37.369 ), 'center' ) < 18000;
```

(11)   What is the average total amount by 5 minutes for the day of 2021-02-11?

```
select pickup_datetime start5min, window(5m, pickup_datetime),
avg(total_amount) avg_total_amount
from rides
where date(pickup_datetime)='2021-02-11'
group by pickup_datetime;
```

The window function creates time windows of 5 minutes based on the column
'pickup_datetime'. The average is taken in the 5 minute windows by grouping the windows.

Spring Boot Framework

Spring Boot provides a platform for Java developers to develop a stand-alone and production-grade spring AI application. The following examples use Maven 3.9.3 (apache-maven-3.9.3) for Java project building, and Java 19.0.2 for compilation. The following pom.xml file is ready for Maven to build the project:

```
File  pom.xml:
```

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">

    <modelVersion>4.0.0</modelVersion>


    <groupId>com.jaguardb</groupId>

    <artifactId>myproject</artifactId>

    <version>0.0.1-SNAPSHOT</version>
```

```xml
<!-- Inherit defaults from Spring Boot -->
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>3.1.2</version>
</parent>


<!-- Add typical dependencies for a web application -->
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>


    <dependency>
        <groupId>com.jaguardb</groupId>
        <artifactId>jaguar-jdbc</artifactId>
        <version>2.1</version>
    </dependency>
</dependencies>
    <!-- Package as an execustore jar -->
    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
                <configuration>
                </configuration>
            </plugin>

            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.11.0</version>
                <configuration>
                    <source>1.9</source>
                    <target>1.9</target>
                </configuration>
```

```
            </plugin>

        </plugins>
    </build>
    <!-- Add Spring repoistories -->
    <!-- (you don't need this if you are using a .RELEASE version) -->
    <repositories>
        <repository>
            <id>spring-snapshots</id>
            <url>https://repo.spring.io/snapshot</url>
            <snapshots><enabled>true</enabled></snapshots>
        </repository>
        <repository>
            <id>spring-milestones</id>
            <url>https://repo.spring.io/milestone</url>
        </repository>
    </repositories>
    <pluginRepositories>
        <pluginRepository>
            <id>spring-snapshots</id>
            <url>https://repo.spring.io/snapshot</url>
        </pluginRepository>
        <pluginRepository>
            <id>spring-milestones</id>
            <url>https://repo.spring.io/milestone</url>
        </pluginRepository>
    </pluginRepositories>
</project>
```

To invoke JaguarDB JDBC classed, the Jagua JDBC class jar file needs to be installed for
Maven to find the classes with the following command:

```
JAR=$HOME/jaguar/lib/jaguar-jdbc-2.1.jar
mvn install:install-file -Dfile=$JAR -DgroupId=com.jaguardb \
    -DartifactId=jaguar-jdbc -Dversion=2.1 -Dpackaging=jar
```

In the same directory where the file pom.xml is located, you can create a directory src/main/java, and create a Java file for your project:

`src/main/java/JaguarDBExample.java` file:

```java
import java.io.*;
import java.sql.DatabaseMetaData;
import java.sql.PreparedStatement;
import javax.sql.DataSource;
import java.sql.SQLException;
import java.sql.Connection;
import java.sql.Statement;
import java.sql.ResultSet;
import java.sql.ResultSetMetaData;
import java.sql.DriverManager;
import java.util.Random;

import com.jaguar.jdbc.JaguarDriver;
import com.jaguar.jdbc.JaguarDataSource;
import com.jaguar.jdbc.JaguarPreparedStatement;
import com.jaguar.jdbc.JaguarResultSetMetaData;

import org.springframework.boot.*;
import org.springframework.boot.autoconfigure.*;
import org.springframework.stereotype.*;
import org.springframework.web.bind.annotation.*;

@RestController
@EnableAutoConfiguration
public class JaguarDBExample {

    @RequestMapping("/")
    String home() {
        return "Hello World from JaguarDB SprintBoot!";
    }
```

```java
    @RequestMapping("/create")

    String create() {

        try {

            Statement statement = connection_.createStatement();

            statement.executeUpdate("create store boot123 ( key: zid zuid, value: addr
char(32));" );

        } catch (SQLException e ) {

            return "create store exception";

        }


        return "Created store boot123 key: zid zuid, value: addr char(32)";

    }


    @RequestMapping("/insert")

    String insert() {

        StringBuffer sb = new StringBuffer();


        try {

            JaguarStatement jst;

            Statement statement = connection_.createStatement();

            statement.executeUpdate("insert into boot123 (addr ) values ( 'v1000')");

            jst = (JaguarStatement)statement;

            sb.append( jst.getLastZuid() + "<br>");


            statement.executeUpdate("insert into boot123 (addr ) values ( 'v2000')");

            jst = (JaguarStatement)statement;

            sb.append( jst.getLastZuid() + "<br>");


            statement.executeUpdate("insert into boot123 (addr ) values ( 'v3000')");

            jst = (JaguarStatement)statement;

            sb.append( jst.getLastZuid() + "<br>");


        } catch ( SQLException e ) {

            return "insert into store exception";

        }
```

```java
        sb.append("<br>Inserted records into store boot123");
        return sb.toString();
    }


    @RequestMapping("/select")
    String select() {
        ResultSet rs;

        try {
            Statement statement = connection_.createStatement();
            rs = statement.executeQuery("select * from boot123");
        } catch ( SQLException e ) {
            return "select exception";
        }


        String key, val;
        StringBuffer sb = new StringBuffer();

        try {
            while(rs.next()) {
                key = rs.getString("uid");
                val = rs.getString("addr");

                sb.append( key );
                sb.append( ":" );
                sb.append( val );
                sb.append( "<br>" );
            }
        } catch ( SQLException e ) {
            return "select next() exception";
        }


        String hdr = "select result:<br><br>";
        String ret = hdr + sb.toString();
        return ret;
```

```
    }


    @RequestMapping("/drop")

    String drop() {

        try {

            Statement statement = connection_.createStatement();

            statement.executeUpdate("drop store if exists boot123");

        } catch (  SQLException e ) {

            return "drop exception";

        }


        return "store boot123 is dropped";

    }


    public static void main(String[] args) throws Exception {

        System.loadLibrary("JaguarClient");


        ds_ = new JaguarDataSource( "127.0.0.1", 8888, "test");

        connection_ = ds_.getConnection("admin_api_key");


        SpringApplication.run(JaguarDBExample.class, args);

    }


    static private DataSource ds_;

    static private Connection connection_;

}
```

To build and run the Spring Boot application, you can execute the following command:

```
export LD_LIBRARY_PATH=$HOME/jaguar/lib

./mvnw spring-boot:run
```

The command mvnw is created in the current directory by the following command:

```
mvn wrapper:wrapper
```

While the Sprint Boot application is up and running, you can open the following URL in your browser:

http://192.168.1.58:8080/

http://192.168.1.58:8080/create

http://192.168.1.58:8080/insert

http://192.168.1.58:8080/select

http://192.168.1.58:8080/drop

The IP address 192.168.1.58 represents the host where the Spring Boot application is running. The example provides a basic demonstration of how an application can interact with JaguarDB. However, developers have the potential to create much more sophisticated applications by leveraging the capabilities of JaguarDB and combining them with their expertise in Java programming.

Summary

Jaguar is a massive linearly scalable vector database that can be used as a high-performant vector data store, search engine, indexing engine. Jaguar has strong support for vector data, time-series and location-based data. It allows fast similarity search, anomaly detection, scalar and vector indexing. Jaguar integrates vector data, time-series data, location data, documents into one for the full control of AI systems.

Reference

Vector Search

```python
import jaguarpy, sys, json
from sentence_transformers import SentenceTransformer



### store text data into jaguardb
def storeText(jag, model, text):
    sentences = [ text ]
    embeddings = model.encode(sentences, normalize_embeddings=False)
    comma_separated_str = ",".join( [str(x) for x in embeddings[0] ])

    istr = "insert into  textvec values ( '" + comma_separated_str + "', '" + text + "' )"
    jag.execute( istr )
    return jag.getLastZuid()



### search similar text data from jaguadb
def searchSimilarTexts(jag, model, queryText, K):
    sentences = [ queryText ]
    embeddings = model.encode(sentences, normalize_embeddings=False)
    comma_separated_str = ",".join( [str(x) for x in embeddings[0] ])

    qstr = "select similarity(v, '" + comma_separated_str
    qstr += "', 'topk=" + str(K) + ",type=cosine_fraction_short')"
    qstr += " from  textvec"

    jag.query( qstr )

    jsonstr = ''
    while jag.fetch():
        jsonstr = jag.json()

    return jsonstr



def getTextByVID(jag, vid):
    qstr =" select zid from test. textvec. textvec_idx where v='" + vid + "'"
    zid = ''
    jag.query( qstr )
```

201

```python
    while jag.fetch():
        zid = jag.getValue("zid")


    qstr = "select text from  textvec where zid='" + zid + "'"
    jag.query( qstr )
    txt = ''
    while jag.fetch():
        txt = jag.getValue("text")


    return txt



def retrieveTopK( jag, model, query_text, K ):
    print("Query: " + query_text )
    json_str = searchSimilarTexts( jag, model, query_text, K )
    json_obj = json.loads(json_str)


    i = 0;
    print("\n")
    print("Retrieved similar texts: ")
    for rec in json_obj:
        dat = rec[str(i)]
        print("\n")
        print("Rank: " + str(i+1))
        vid = dat["id"]
        print("Vector ID: " + vid )
        print("Distance: " + dat["distance"] )
        txt = getTextByVID( jag, vid )
        print("Text: " + txt )
        i += 1


    print("\n\n")



'''
A number of texts are first inserted into vector dabatase. Then later a user enters
a query text, the program will find top K (K=5) texts that best match the query text.
'''
```

```python
def main():

    ### connect to JaguarDB
    jag = jaguarpy.Jaguar()

    host = "127.0.0.1"
    port = sys.argv[1]
    user = "user-api-key"
    vectordb = "test"

    rc = jag.connect( host, port, user, vectordb )
    print ("Connected to JaguarDB server" )

    ### create store for vector data. Notice that 1024 is the dimension for BAAI/bge-large-en model
    jag.execute("drop store if exists  textvec")
    jag.execute("create store  textvec ( key: zid zuid, value: v vector(1024, 'cosine_fraction_short'), text char(2048) )")

    jag.execute("drop index if exists  textvec_idx on  textvec")
    jag.execute("create index  textvec_idx on  textvec( v, zid )")

    ### use the BAAI/bge-large-en model
    model = SentenceTransformer('BAAI/bge-large-en')

    ### store texts into vectordb
    text = "Human impact on the environment (or anthropogenic environmental impact) refers to changes to biophysical environments and to ecosystems, biodiversity, and natural resources caused directly or indirectly by humans."
    zuid1 = storeText( jag, model, text )

    text = "a group of people involved in persistent interpersonal relationships, or a large social grouping sharing the same geographical or social territory, typically subject to the same political authority and dominant cultural expectations. Human societies are characterized by patterns of relationships (social relations) between individuals who share a distinctive culture and institutions; a given society may be described as the total of such relationships among its constituent members."
    zuid2 = storeText( jag, model, text )

    text = "In 1768, Astley, a skilled equestrian, began performing exhibitions of trick horse riding in an open field called Ha'Penny Hatch on the south side of the Thames River, England. In 1770, he hired acrobats, tightrope walkers, jugglers and a clown to fill in the pauses between the equestrian demonstrations and thus chanced on the format which was later named a circus.
```

Performances developed significantly over the next fifty years, with large-scale theatrical battle reenactments becoming a significant feature. "

    zuid3 = storeText(jag, model, text)

    text = "Astley had a genius for trick riding. He saw that trick riders received the most attention from the crowds in Islington. He had an idea for opening a riding school in London in which he could also conduct shows of acrobatic riding skill. In 1768, Astley performed in an open field in what is now the Waterloo area of London, behind the present site of St John's Church. Astley added a clown to his shows to amuse the spectators between equestrian sequences, moving to fenced premises just south of Westminster Bridge, where he opened his riding school from 1769 onwards and expanded the content of his shows. He taught riding in the mornings and performed his feats of horsemanship in the afternoons."

    zuid4 = storeText(jag, model, text)

    text = "After the Amphitheatre was rebuilt again after the third fire, it was said to be very grand. The external walls were 148 feet long which was larger than anything else at the time in London.The interior of the Amphitheatre was designed with a proscenium stage surrounded by boxes and galleries for spectators. The general structure of the interior was octagonal. The pit used for the entertainers and riders became a standardised 43 feet in diameter, with the circular enclosure surrounded by a painted four foot barrier. Astley's original circus was 62 ft (~19 m) in diameter, and later he settled it at 42 ft (~13 m), which has been an international standard for circuses since."

    zuid5 = storeText(jag, model, text)

    text = "According to the Big Bang theory, the energy and matter initially present have become less dense as the universe expanded. After an initial accelerated expansion called the inflationary epoch at around 10-32 seconds, and the separation of the four known fundamental forces, the universe gradually cooled and continued to expand, allowing the first subatomic particles and simple atoms to form. Dark matter gradually gathered, forming a foam-like structure of filaments and voids under the influence of gravity. Giant clouds of hydrogen and helium were gradually drawn to the places where dark matter was most dense, forming the first galaxies, stars, and everything else seen today."

    zuid6 = storeText(jag, model, text)

    text = "By comparison, general relativity did not appear to be as useful, beyond making minor corrections to predictions of Newtonian gravitation theory. It seemed to offer little potential for experimental test, as most of its assertions were on an astronomical scale. Its mathematics seemed difficult and fully understandable only by a small number of people. Around 1960, general relativity became central to physics and astronomy. New mathematical techniques to apply to general relativity streamlined calculations and made its concepts more easily visualized. As astronomical phenomena were discovered, such as quasars (1963), the 3-kelvin microwave background radiation (1965), pulsars (1967), and the first black hole candidates (1981), the theory explained their attributes, and measurement of them further confirmed the theory."

    zuid7 = storeText(jag, model, text)

    text = "In astronomy, the magnitude of a gravitational redshift is often expressed as the velocity that would create an equivalent shift through the relativistic Doppler effect. In such units, the 2 ppm sunlight redshift corresponds to a 633 m/s receding velocity, roughly of the same magnitude as convective motions in the sun, thus complicating the measurement.[9] The GPS satellite gravitational blueshift velocity equivalent is less than 0.2 m/s, which is negligible compared to the actual Doppler shift resulting from its orbital velocity."

    zuid8 = storeText(jag, model, text)

    text = "Turn on the sprinkler system. In order to locate the break or leak in the sprinkler system, you need to run water through it. Turn on the sprinkler system to activate the flow of

water. Allow the water to run for about 2 minutes before you check the lines. Do this in the daytime, when you'll have an easier time spotting the leak. If your sprinkler system is separated into zones, activate the zones one at a time so you can identify the break or leak more easily."

```
zuid9 = storeText( jag, model, text )
```

text = "Check for water bubbling up from the soil. If you see a pool of water or water coming from the soil,then there's a leak in the sprinkler line buried underneath. Mark the general location of the leak or break so you can identify it when the water is turned off. Place an item like a shovel or a rock on the ground near the leak. Turn off the sprinkler system after you've found the leak. If you've found the signs of a leak and located the region where the line is leaking or broken, turn off the water so you can repair the line. Use the shut-off valve in the control box to stop the flow of water through the system."

```
zuid10 = storeText( jag, model, text )
```

text = "In fact, Antarctica is such a good spot for meteorite hunters that crews of scientists visit every year, searching for these otherworldly rocks, driving around the surface until they spot a lone dark rock on an otherwise unbroken expanse of white. However, you don't always have to travel to the other side of the world to find a meteorite. Sometimes meteorites will come to you. Keep an eye open for local reports of brilliant fireballs lighting your region's sky. Debris from such displays scatters across the ground and \

        sometimes hits structures or vehicles. Watch for information about fireballs in your area on the websites of the American Meteor Society or the International Meteor Organization."

```
zuid11 = storeText( jag, model, text )
```

text = "Most tornadoes are found in the Great Plains of the central United States – an ideal environment for the formation of severe thunderstorms. In this area, known as Tornado Alley, storms are caused when dry cold air moving south from Canada meets warm moist air traveling north from the Gulf of Mexico. Tornadoes can form at any time of year, but most occur in the spring and summer months along with thunderstorms. May and June are usually the peak months for tornadoes. The Great Plains are conducive to the type of thunderstorms (supercells) that spawn tornadoes. It is in this region that cool, dry air in the upper levels of the atmosphere caps warm, humid surface air. This situation leads to a very unsstore atmosphere and the development of severe thunderstorms."

```
zuid12 = storeText( jag, model, text )
```

Make a query and get similar texts from database

query_text = "More recently, that focus has shifted eastward by 400 to 500 miles. In the past decade or so tornadoes have become prevalent in eastern Missouri and Arkansas, western Tennessee and Kentucky, and northern Mississippi and Alabama—a new region of concentrated storms. Tornado activity in early 2023 epitomized the trend."

```
K = 3;
```

```
retrieveTopK( jag, model, query_text, K )
```

Make another query and get similar texts from database

query_text = "Think of designing a landscape for the bare lot surrounding your new home as an adventure in creativity. Perhaps your property needs only a few small, easily doable projects to make it more attractive. Either way, it's important to consider how each change will relate to the big picture. Stand back from time to time to see the entire landscape and how each part fits into it."

```
K = 3;
```

```
retrieveTopK( jag, model, query_text, K )
```

```
    ### select a vector from one column

    jag.query("select vector(v, 'type=cosine_fraction_short') from  textvec where zid='"+ zuid1
+"'" )

    while jag.fetch():

        print("json ", jag.json() )


    jag.close()

    jag = None


if __name__ == "__main__":

  main()
```

Location Data

The following statements are examples of JaguarDB geospatial data management.

```
drop store if exists geom1;

create store if not exists geom1 ( key: a int, value: pt point(srid:4326), b int );

desc geom1 detail;

insert into geom1 values ( 1, point(22 33), 123 );

insert into geom1 (a, pt, b ) values ( 2, point(22 33), 123 );

insert into geom1 (b, pt, a ) values ( 222, point(22 33), 12 );

select * from geom1;

drop index if exists geom1_idx1 on geom1;

create index geom1_idx1 on geom1(b,pt);

select * from geom1_idx1;


drop store if exists geom2;

create store if not exists geom2 ( key: a int, value: pt1 point, b int, uid uuid, pt2
point(srid:wgs84) );

desc geom2;

insert into geom2 values ( 1, point(22 33), 123, point(99 221) );

insert into geom2 values ( 10, json({"type":"Point", "coordinates": [2,3]}), 123,
json({"type":"Point", "coordinates":[5,9]}) );
```

```
insert into geom2 (a, pt1, pt2, b ) values ( 2, point(22 33), point(23 421), 123 );

insert into geom2 (b, pt2, pt1, a ) values ( 222, point(22 33), point(90 21), 17 );

select * from geom2;

drop index if exists geom2_idx1  on geom2;

create index geom2_idx1 on geom2(b,pt2);

select * from geom2_idx1;


drop store if exists geom3;

create store if not exists geom3 ( key: pt1 point, value: b int, uid uuid, a int, pt2
point );

desc geom3;

insert into geom3 values ( point(22 33), 123, 2, point(99 221) );

insert into geom3 (b, pt2, pt1, a ) values ( 2, point(25 33), point(23 451), 153 );

select * from geom3;

drop index if exists geom3_idx1  on geom3;

create index geom3_idx1 on geom3(b,pt2,uid);

select * from geom3_idx1;


drop store if exists d5;

create store if not exists d5 ( key: a int, pt1 point3d, b int, pt2 point3d, value: c
int, pt3 point3d, d int, pt4 point3d(srid:wgs84) );

desc d5;

insert into d5 values( 1, point3d(22 33 4), 23, point3d(99 22 1), 244, point3d(8 2 3),
234,  point3d(8 2 3) );

insert into d5 values( 2, point3d(32 83 0), 23, point3d(94 82 1), 214, point3d(9 7 2),
234,  point3d(1 2 3) );

select * from d5;

drop index if exists d5_idx  on d5;

create index d5_idx on d5(pt3, pt4, d, c );

select * from d5_idx;


drop store if exists d6;

create store if not exists d6 ( key: a int, pt1 point, b int, pt2 point, value: c int,
pt3 point, d int, pt4 point3d );

desc d6;

insert into d6 values( 1, point(22 33 ), 23, point(99 1), 244, point(8  3), 234,
point3d(8 2 3) );
```

```
insert into d6 values( 2, point(32 83 ), 23, point(94 82 ), 214, point(9  2), 234,
point3d(1 2 3) );

insert into d6 ( pt2, a, b, pt1 ) values ( json({"type":"Point", "coordinates":
[123,321]}), 208, 12, point(91 17) );

insert into d6 ( pt2, a, b, pt1 ) values ( json({"type":"Point", "coordinates":
[124,351]}), 209, 13, point(92 19) );

select * from d6;

drop index if exists d6_idx  on d6;

create index d6_idx on d6(pt4, pt3, d, c );

select * from d6_idx;




drop store if exists cir1;

create store if not exists cir1 ( key: a int, c1 circle(4326), b int, c2 circle,
value: c int, c3 circle, d int, c4 circle );

desc cir1 detail;

insert into cir1 values ( 100, circle( 22 33 100), 123, circle(99 22 191), 2133,
circle(99 22 12), 123, circle(88 33 2211) );

insert into cir1 values ( 101, circle( 92 33 140), 523, circle(99 42 191), 2133,
circle(99 42 12), 823, circle(38 43 811) );

drop index if exists cir1_idx1 on cir1;

create index cir1_idx1 on cir1(c3, d, c2);

select * from cir1_idx1;




drop store if exists sph1;

create store if not exists sph1 ( key: a int, s1 sphere, b int, s2 sphere, value: c
int, s3 sphere );

desc sph1 detail;

insert into sph1 values ( 100, sphere( 2 3 4 123), 321, sphere(99 22 33 20000), 321,
sphere(99 223 12020 29292) );

insert into sph1 values ( 102, sphere( 2 3 4 123), 321, sphere(99 22 33 20000), 321,
sphere(99 223 12020 29292) );

insert into sph1 (s1, b, a, s2 ) values ( sphere( 2 3 4 123), 921, 234, sphere(99 22
33 20000) );

insert into sph1 (s1, b, a, s2 ) values ( sphere( 2 32 5 123), 951, 534, sphere(99 22
33 20000) );

drop index if exists sph1_idx1 on sph1;

create index sph1_idx1 on sph1(c, s2, a, s1);

select * from sph1_idx1;
```

```
drop store if exists sq1;

create store if not exists sq1 ( key: a int, s1 square, b int, s2 square, value: c
int, s3 square );

desc sq1 detail;

insert into sq1 values ( 100, square( 22 453 22222), 100, square(9 3 123), 299,
square(82 332 1212) );

drop index  if exists sq1_idx1 on sq1;

create index sq1_idx1 on sq1(s3);



drop store if exists cb1;

create store if not exists cb1 ( key: a int, q1 cube, b int, q2 cube, value: c int, q3
cube );

desc cb1 detail;

insert into cb1 values ( 111, cube( 2 3 4 1233), 1234, cube(233 22 55 9393), 3212,
cube(92 92 82 2345) );

select * from cir1;

select * from sph1;

select * from sq1;

select * from cb1;

drop index  if exists cb1_idx1 on cb1;

create index cb1_idx1 on cb1(q3,q2, b);

select * from cb1_idx1;




drop store if exists rect1;

create store if not exists rect1 ( key: a int, r1 rectangle, value: c int );

desc rect1 detail;

insert into rect1 values ( 1, rectangle(22 33 88 99), 233 );

insert into rect1 ( c, a, r1 ) values ( 22, 31, rectangle(29 13 48 19) );

drop index  if exists rect1_idx1 on rect1;

create index rect1_idx1 on rect1(c, r1);



drop store if exists bx1;

create store if not exists bx1 ( key: a int, b1 box, value: c int, b2 box );

desc bx1 detail;
```

```
insert into bx1 values ( 1, box(22 33 44 88 99 123), 233, box(9 9 9 22 22 33) );
insert into bx1 ( c, a, b1 ) values ( 22, 31, box(29 13 48 19 21 12) );
select * from bx1;
select distance( point3d(0 0 0), b2, 'max') as maxdist from bx1;
select distance( point3d(0 0 0), b2, 'min') as mindist from bx1;
drop index  if exists bx1_idx1 on bx1;
create index bx1_idx1 on bx1(b2, c );


drop store if exists cyn1;
create store if not exists cyn1  ( key: a int, c1 cylinder, value: c int );
desc cyn1 detail;
insert into cyn1 values ( 1, cylinder(1 2 3 45 88 0.3), 1239 );
insert into cyn1 ( c, c1, a ) values ( 13, cylinder(1 2 3 45 88), 139 );
select * from cyn1;
drop index  if exists cyn1_idx1 on cyn1;
create index cyn1_idx1 on cyn1(c);
select * from cyn1_idx1;



drop store if exists cn1;
create store if not exists cn1  ( key: a int, c1 cone, value: c int, c2 cone );
desc cn1 detail;
insert into cn1 values ( 1, cone(1 2 3 45 88), 1239, cone(33 22 44 44 99 0.4 0.3) );
insert into cn1 ( c, c1, a  ) values ( 13, cone(1 2 3 45 88), 139 );
select * from cn1;
drop index  if exists cn1_idx1 on cn1;
drop index  if exists cn1_idx2 on cn1;
create index cn1_idx1 on cn1(c2);
create index cn1_idx2 on cn1(c,c2);
select * from cn1_idx1;
select * from cn1_idx2;


drop store if exists el1;
create store if not exists el1  ( key: a int, c1 ellipse, value: c int, c2 ellipse );
```

```
desc el1 detail;

insert into el1 values ( 1, ellipse(1 2 45 88), 1239, ellipse(22 44 44 99) );

insert into el1 ( c, c1, a  ) values ( 13, ellipse(2 3 45 88), 139 );

select * from el1;

drop index  if exists el1_idx1 on el1;

drop index  if exists el1_idx2 on el1;

create index el1_idx1 on el1(c2,c);

create index el1_idx2 on el1(c,c2);

select * from el1_idx1;

select * from el1_idx2;


drop store if exists es1;

create store if not exists es1  ( key: a int, c1 ellipsoid, value: c int, c2
ellipse );

desc es1 detail;

insert into es1 values ( 1, ellipsoid(1 2 3 45 88 99), 1239, ellipse(22 44 44 99) );

insert into es1 ( c, c1, a  ) values ( 13, ellipsoid(2 3 4 45 88 99), 139 );


select * from rect1;

select * from bx1;

select * from cyn1;

select * from cn1;

select * from el1;

select * from es1;


drop index  if exists es1_idx1 on es1;

drop index  if exists es1_idx2 on es1;

drop index  if exists es1_idx3 on es1;

create index es1_idx1 on es1(c2);

create index es1_idx2 on es1(c,c2);

create index es1_idx3 on es1(c1,a,c2);

select * from es1_idx1;

select * from es1_idx2;

select * from es1_idx3;
```

```
drop store if exists line1;

create store if not exists line1  ( key: a int, c1 line, value: c int, c2 line );

desc line1 detail;

insert into line1 values ( 1, line(1 2, 45 8.3), 1239, line(44 99, 291 9.1 ) );

insert into line1 values ( 3, line(1 20, 4 3), 139, line(4 9, 91 9 ) );

select * from line1;

drop index  if exists line1_idx1 on line1;

create index line1_idx1 on line1(c2, c, c1 );

select * from line1_idx1;




drop store if exists line3d2;

create store if not exists line3d2  ( key: a int, c1 line3d, value: c int, c2
line3d );

desc line3d2 detail;

insert into line3d2 values ( 1, line3d(1 2 45 8.3 22 3.3), 1239, line3d(44 99 291 9.1
33 44 ) );

select * from line3d2;

drop index  if exists line3d2_idx1 on line3d2;

create index line3d2_idx1 on line3d2(c2, c, c1 );

select * from line3d2_idx1;


drop store if exists tri1;

create store if not exists tri1 ( key: t1 triangle, value: a int );

insert into tri1 values ( triangle( 11 33 88 99 21 32), 123 );

insert into tri1 values ( triangle( 31 33 18 99 33 44), 223 );

drop index  if exists tri1_idx1 on tri1;

create index tri1_idx1 on tri1( a );

select * from tri1_idx1;


drop store if exists tri31;

create store if not exists tri31 ( key: t1 triangle3d, value: a int );

insert into tr31 values ( triangle3d( 11 33 88 99 23 43 9 8 2), 123 );

insert into tr31 values ( triangle3d( 31 33 18, 99 12 34, 9 9 1), 223 );
```

```
drop index  if exists tri31_idx1  on tri1;

create index tri31_idx1 on tri1( a );

select * from tri31_idx1;




### queries

select * from cir1 where within(point(10 22), c1 );

select * from cir1 where coveredby(point(10 22), c1 );

select * from cir1 where contain(c1, point(10 22) );

select * from cir1 where cover(c1, point(10 22) );



select * from cir1 where within( c1, rectangle(1 2 23 34 0.1) );

###                                              x y  a  b nx

select * from cir1 where disjoint( c1, rectangle(1 2 23 34 0.1) );

select * from cir1 where nearby( c1, rectangle(1 2 23 34 0.1), 200 );

select distance( c1, point(22 33), 'center' ) as dist from cir1;

select distance( c1, point(22 33), 'max' ) as dist from cir1;

select distance( c1, point(22 33), 'min' ) as dist from cir1;

select distance( point(22 33), c1, 'center' ) as dist from cir1;

select distance( point(22 33), c1, 'max' ) as dist from cir1;

select distance( point(22 33), c1, 'min' ) as dist from cir1;



select * from cb1 where within( point3d(100 200 300), q1 );

###                                      x   y    z

select * from cb1 where cover( q1, sphere(11 234 234  100) );

###                                        x   y   z    r



select * from cb1 where nearby( q2, sphere(31 434 235  100), 3000 );



select * from cb1 where nearby( q2, ellipsoid(31 434 235  100 200 200), 3000 );



select * from cb1 where nearby( q3, ellipsoid(31 434 235  100 200 300 0.1 0.2),
3000 );

###                                       x   y   z    a   b   c   nx ny
```

213

linestring

```
drop store if exists linestr1;

create store linestr1 ( key: a int, value: ls1 linestring(wgs84), b int, ls2
linestring );

desc linestr1 detail;

insert into linestr1 values ( 1, linestring( 11 2,2 33 , 33 44, 55 66, 55 66, 77
88 ), 200, linestring( 33 44, 55 66, 8 9 ) );

insert into linestr1 values ( 2, linestring( 11.13 2,2.9 33 , 33 44, 5.5 6.6, 55 66,
77 88 ), 210, linestring( 3.3 4.4, 5.5 6.6, 8.9 9 ) );

insert into linestr1 values ( 2,json('{"type":"LineString","coordinates":
[[2,3],[3,4]]}'),121,json({"type":"LineString","coordinates": [[2,3],[3,4]]}) );

select * from linestr1;

select * from linestr1 where within( ls1, square( 10 10 78.1 ) );


create index linestr1_idx1 on linestr1( b, ls2 );

desc linestr1_idx1 detail;

select * from linestr1_idx1;



drop store if exists linestr21;

create store linestr21 ( key: ls1 linestring(wgs84), value: a int );

desc linestr21 detail;

insert into linestr21 values ( linestring( 11 2,2 33 , 33 44, 55 66, 55 66, 77 88 ),
200 );

select * from linestr21;

select * from linestr21 where within( ls1, square( 10 10 78.1 ) );

create index linestr21_idx1 on linestr21( a );

desc linestr21_idx1 detail;

select * from linestr21_idx1;



drop store if exists linestr2;

create store linestr2 ( key: ls1 linestring(wgs84), a int, value: ls2 linestring );

desc linestr2 detail;

insert into linestr2 values ( linestring( 1 2,2 33 , 33 44, 55 66, 55 66, 77 88 ),
200, linestring( 33 44, 55 66, 8 9 ) );
```

```
insert into linestr2 values ( linestring( 1.13 2,2.9 33 , 33 44, 5.5 6.6, 55 66, 77
88 ), 210, linestring( 3.3 4.4, 5.5 6.6, 8.9 9 ) );

select * from linestr2;

select * from linestr2 where within( ls1, square( 10 10 78.1 ) );

create index linestr2_idx1 on linestr2( ls2 );

desc linestr2_idx1 detail;

select * from linestr2_idx1;




drop store if exists linestr3;

create store linestr3 ( key: ls1 linestring(wgs84), a int, value: ls2 linestring, b
int );

desc linestr3 detail;

insert into linestr3 values ( linestring( 211 2,2 33 , 33 44, 55 66, 55 66, 77 88 ),
200, linestring( 33 44, 55 66, 8 9 ), 804 );

insert into linestr3 values ( linestring( 211.13 2,2.9 33 , 33 44, 5.5 6.6, 55 66, 77
88 ), 210, linestring( 3.3 4.4, 5.5 6.6, 8.9 9 ), 805 );

select * from linestr3;

select * from linestr3 where within( ls2, square( 10 10 78.1 ) );

select geojson(ls1) from linestr3 where intersect( ls1, square( 10 10 78.1 ) );

select geojson(ls1), geojson(ls2) from linestr3 where intersect( ls1, square( 10 10
78.1 ) ) and intersect( ls2, square( 10 10 1000 ) );

create index linestr3_idx1 on linestr3( b, ls2 );

desc linestr3_idx1 detail;

select * from linestr3_idx1;




drop store if exists linestr3d1;

create store linestr3d1 ( key: ls1 linestring3d(wgs84), a int, value: ls2 linestring,
b int );

desc linestr3d1 detail;

insert into linestr3d1 values ( linestring3d( 1 2 2,1 2 33 , 8 33 44, 8 55 66  ),
200, linestring( 33 44, 55 66  ), 804 );

insert into linestr3d1 values ( linestring3d( 1.1 2 2, 2 2.9 3 , 3 3 4, 2 5 6  ),
210, linestring( 3.3 4, 5 6  ), 805 );

insert into linestr3d1 values ( linestring3d( 0 -10 0, 0 10 0, 2 2.9 3 , 3 3 4, 2 5
6 ), 310, linestring( 3.3 4, 5 6 ), 805 );
```

```
insert into linestr3d1 values ( linestring3d( 0 -20 0, 0 20 0, 2 2.7 3.8  ), 315,
linestring( 3.3 4.2, 5.1 6.7 ), 808 );

select * from linestr3d1;

select ls2:x, ls2:y from linestr3d1;

select geo:id, geo:col, geo:i, ls2:x, ls2:y from linestr3d1 where ls2:x > 0;

select * from linestr3d1 where within( ls1, cube( 10 10 10 78.1 ) );

select * from linestr3d1 where intersect( ls1, cube( 10 10 10 78.1 ) );

select * from linestr3d1 where intersect( ls1, linestring3d( 0 0 -10, 0 0 10, 10 10
78.1) );

select geojson(ls1) from linestr3d1 where intersect( ls1, linestring3d( 0 0 -10, 0 0
10, 10 10 78.1) );

select geojson(ls2) from linestr3d1 where within( ls2, square( 0 0 1000000 ) );

create index linestr3d1_idx1 on linestr3d1( b, ls2 );

desc linestr3d1_idx1 detail;

select * from linestr3d1_idx1;


drop store if exists lstr;

create store lstr ( key: a int, value: ls linestring );

insert into lstr values ( 1, linestring(0 0, 20 0) );

insert into lstr (ls, a) values ( linestring(0 0, 20 0), 121 );

insert into lstr (a, ls) values ( 124, linestring(1 1, 20 0) );

select * from lstr;

select geojson(ls) from lstr where intersect(ls, linestring(10 -10, 10 10) );

select geojson(ls) from lstr where within(ls, square(0 0 10000) );


drop store if exists pol1;

create store pol1 ( key: a int, value: pol polygon );

insert into pol1 values ( 1, polygon( (0 0, 20 0, 88 99, 0 0) ) );

insert into pol1 values ( 21, polygon( (0 0, 80 0, 80 80, 0 80, 0 0) ) );

insert into pol1 values ( 2,  json({"type":"Polygon", "coordinates": [[[0,0], [2,0],
[8,9], [0, 0]], [[1, 2], [2, 3],[1, 2]]]}) );

insert into pol1 values ( 3,  json({"type":"Polygon", "coordinates": [[[0,0], [2,0],
[8,9], [0, 0]], [[1, 2], [2, 3],[1, 2]]]}) );

select * from pol1;

select * from pol1 where intersect( pol, line(0 10 80 10 ) );

select geojson(pol) from pol1 where intersect(pol, linestring(10 -10, 10 10) );
```

```
select geojson(pol) from pol1 where within(pol, square(0 0 10000) );

select geojson(pol) from pol1 where intersect(pol, square(0 0 10000) );


drop store if exists pol2;

create store pol2 ( key: a int, value: po2 polygon, po3 polygon3d, tm timestamp
default current_timestamp, ls linestring );

insert into pol2 values( 1, polygon((0 0,2 0,8 9,0 0),(1 2,2 3,1 2)),polygon3d((1 1
1,2 2 2,3 3 3,1 1 1),(2 2 2,3 3 1,2 2 2)),linestring(30 40,40 50,5 6));

insert into pol2 values( 2, json({"type":"Polygon", "coordinates": [[[0,0], [2,0],
[8,9], [0, 0]], [[1, 2], [2, 3],[1, 2]]]}),polygon3d((4 1 2,2 2 2,3 3 3,1 9 1, 4 1
2),(2 2 2,3 3 1, 8 2 9, 2 2 2)),linestring(30 40,40 50,5 6));

select * from pol2;

select geojson(po3) from pol2 where within(po3, cube(0 0 0 100000) );

select geojson(po2) from pol2 where within(po2, square( 0 0 100000) );

select geojson(po2) from pol2 where intersect(po2, square( 0 0 100000) );



drop store if exists mp;

create store mp ( key: a int, value: m1 multipoint, m2 multipoint3d );

desc mp detail;

insert into mp ( m1, a, m2 ) values ( multipoint( 1 2 , 3 4, 2 1 ), 100,
multipoint3d( 1 2 3, 3 4 5, 2 2 1) );

insert into mp values ( 123, multipoint( 1 2 , 3 4, 2 1 ), multipoint3d( 1 2 3, 3 4
5, 2 2 1) );

insert into mp values ( 125, multipoint( 1 2 , 3 4, 2 1 ), json({"type":"MultiPoint",
"coordinates": [ [1,2,3],[3,4,5] ] } );

select * from mp;



drop store if exists mline;

create store mline ( key: a int, value: l1 multilinestring, l2 multilinestring3d );

desc mline detail;

insert into mline values( 1, multilinestring((0 0,2 0,8 9,0 0),(1 2,2 3,1
2)),multilinestring3d((1 1 1,2 2 2,3 3 3),(2 2 2,3 3 1)));

insert into mline values( 1024, multilinestring( (1 1, 2 3, 4 5 , 4 9 )),
multilinestring3d(( 0 0 0, 1 9 9, 11 12 13, 33 32 34 )) );


insert into mline values( 3, json({"type":"MultiLineString","coordinates":
[ [ [0,0],[2,0],[8,9],[0,0]], [[1,2],[2,3],[1,2]]]}),multilinestring3d((1 1 1,2 2 2,3
3 3),(2 2 2,3 3 1)));
```

```
select * from mline;

select geojson(l1) from mline where intersect(l1, square( 0 0 100000) );

select geojson(l2) from mline where intersect(l2, cube( 0 0 0 100000) );




drop store if exists mpg;

create store mpg ( key: a int, value: p1 multipolygon, p2 multipolygon3d );

desc mpg detail;

insert into mpg values( 1, multipolygon(((0 0,2 0,8 9,0 0),(1 2,2 3, 7 8,1
2))),multipolygon3d(((1 1 1,2 2 2,3 3 3, 1 1 1),(2 2 2,3 3 1, 3 5 6, 2 2 2 ))));


insert into mpg values( 2,

        multipolygon( ((0 0,2 0,8 9,0 0),(1 2,2 3, 7 8,1 2)), ((0 0, 2 2, 3 3, 0
0)) ),

        multipolygon3d(((1 1 1,2 2 2,3 3 3, 1 1 1),(2 2 2,3 3 1, 3 5 6, 2 2 2 ))));


insert into mpg values( 3,

        multipolygon( ((0 0,2 0,8 9,0 0),(1 2,2 3, 7 8,1 2)), ((0.1 0.2, 2.2 2.2, 5 5,
0.1 0.2)) ),

        multipolygon3d(((1 1 1,2 2 2,3 3 3, 1 1 1),(2 2 2,3 3 1, 3 5 6, 2 2 2 ))));


insert into mpg values( 30,

        json( { "type":"MultiPolygon","coordinates": [ [[[4,0], [2,0], [8,9], [4, 0]],
[[1, 5], [2, 3],[1, 5]]], [[[4,4], [2,0], [8,9], [4, 4]], [[1, 2], [2, 3],[1,
2]]] ] } ),

        multipolygon3d(((1 1 1,2 2 2,3 3 3, 1 1 1),(2 2 2,3 3 1, 3 5 6, 2 2 2 ))));


insert into mpg values( 32,

        json( { "type":"MultiPolygon","coordinates": [ [[[4,0], [2,0], [8,9], [4, 0]],
[[1, 5], [2, 3],[1, 5]]], [[[4,4], [2,0], [8,9], [4, 4]], [[1, 2], [2, 3],[1,
2]]] ] } ) ) );




select * from mpg;

select geojson(p1) from mpg where intersect(p1, square( 0 0 100000) );

select geojson(p2) from mpg where intersect(p2, cube( 0 0 0 100000) );
```

```
drop store if exists rg2;

create store rg2 ( key: a int, value: dt datetime, d date, t time, r
range(datetime) );

insert into rg2 values ( 1, '2018-10-10 01:01:01', '2018-12-12', '12:11:11',
range( '2015-10-10 01:01:01', '2028-10-10 01:01:01' ) );

insert into rg2 values ( 2, '2014-10-10 01:01:01', '2015-12-12', '14:11:11',
range( '2010-10-10 01:01:01', '2028-12-31 01:01:01' ) );

select * from rg2;

select * from rg2 where within(d, range('2000-10-10', '2030-01-01') );

select * from rg2 where within(t, range('01:01:01', '13:13:11') );

select * from rg2 where within(dt, range('1980-01-1 01:01:01', '2019-08-09
13:13:11') );

select * from rg2 where intersect(r, range('1980-01-1 01:01:01', '2019-08-09
13:13:11') );

select * from rg2 where intersect(r, range('1980-01-1 01:01:01', '1999-08-09
13:13:11') );
```

```
drop store if exists pold;

create store pold ( key: a int , value:name char(64), pol polygon(wgs84));

insert into pold values(1, "California",json({"type":"Polygon","coordinates":[[[-
123.23325,42.006187],[-122.37885,42.01166],[-121.037,41.99523],[-
120.00186,41.99523],[-119.99638,40.26452],[-120.00186,38.999348],[-
118.71478,38.101128],[-117.4989,37.21934],[-116.540436,36.50186],[-
115.85034,35.970596],[-114.63446,35.00118],[-114.63446,34.87521],[-
114.47015,34.710903],[-114.33323,34.44801],[-114.136055,34.305607],[-
114.25655,34.174164],[-114.41538,34.108437],[-114.53587,33.933174],[-
114.497536,33.697666],[-114.52492,33.54979],[-114.72757,33.40739],[-
114.66184,33.034958],[-114.52492,33.02948],[-114.47015,32.843266],[-
114.52492,32.755634],[-114.72209,32.717297],[-116.04751,32.624187],[-
117.126465,32.536556],[-117.24696,32.668003],[-117.25243,32.876125],[-
117.32912,33.12259],[-117.47151,33.29785],[-117.7837,33.538837],[-
118.18352,33.76339],[-118.26019,33.703144],[-118.41355,33.74148],[-
118.39164,33.84007],[-118.5669,34.042713],[-118.802414,33.998898],[-
119.21866,34.14678],[-119.27891,34.26727],[-119.55823,34.415146],[-
119.87589,34.40967],[-120.13879,34.47539],[-120.47288,34.44801],[-
120.64814,34.579456],[-120.6098,34.85878],[-120.67005,34.902596],[-
120.63171,35.099766],[-120.8946,35.247643],[-120.905556,35.45029],[-
121.00414,35.461243],[-121.16845,35.636505],[-121.28346,35.674843],[-
121.332756,35.78438],[-121.71614,36.195152],[-121.89688,36.315643],[-
121.93522,36.638786],[-121.85854,36.6114],[-121.787346,36.803093],[-
121.92974,36.978355],[-122.105,36.956448],[-122.33504,37.11528],[-
122.41719,37.24125],[-122.400764,37.36174],[-122.51578,37.520573],[-
122.51578,37.783466],[-122.32956,37.783466],[-122.406235,38.15042],[-
122.488396,38.112083],[-122.50482,37.931343],[-122.701996,37.893005],[-
122.9375,38.029926],[-122.97584,38.265434],[-123.129196,38.451653],[-
123.33184,38.56667],[-123.44138,38.698112],[-123.73714,38.95553],[-
123.68784,39.032207],[-123.82477,39.366302],[-123.76452,39.552517],[-
```

```
123.85215,39.83184],[-124.109566,40.105686],[-124.3615,40.25904],[-
124.4108,40.43978],[-124.15886,40.877937],[-124.109566,41.025814],[-
124.15886,41.14083],[-124.06575,41.442062],[-124.1479,41.715908],[-
124.25745,41.78163],[-124.21363,42.00071],[-123.23325 ,42.006187]]]}));


drop store if exists lstrm;

create store lstrm ( key: a int, b int, value: ls linestring(srid:4326,metrics:10) );

insert into lstrm values ( 100, 200, linestring(0 80 100 200 300, 0.1 80.2 300 400 550
600 700, 0.2 80.5 1000 2000 23456, 0.8 80.9 10000 30000) );



select area(p1), area(p2) from  mpg;

select dimension(p1), dimension(p2) from mpg;

select geotype(p1) from mpg;

select pointn(ls,1) from lstr;

select pointn(ls,2) from lstr;


select extent(p1) from mpg;

select extent(p2) from mpg;

select startpoint(ls) from lstr;

select endpoint(ls) from lstr;

select isclosed(ls) from lstr;

select isclosed(p1) from mpg;

select numpoints(ls) from lstr;

select numpoints(l2) from mline;

select numsegments(l2) from mline;

select numrings(l2) from mline;

select numrings(p1) from mpg;

select numrings(p2) from mpg;

select numpolygons(p1) from mpg;

select numpolygons(p2) from mpg;

select srid(p1) from mpg;

select srid(p2) from mpg;

select summary(p1) from mpg;

select xmin(ls) from lstr;

select ymin(ls) from lstr;
```

```
select xmax(ls) from lstr;

select ymax(ls) from lstr;

select convexhull(p1) from mpg;

select convexhull(pol) from pol1;

select convexhull(ls) from lstr;

select centroid(p1) from mpg;

select volume(po3) from pol2;

select volume(q1), volume(q2) from cb1;

select closestpoint( point(1 1), ls) from lstr;

select closestpoint( point(1 1 ), pol) from pol1;

select angle(c1, c2) from line1;

select angle(line(0 0, 2 5), c2) from line1;

select angle(line3d(0 0 0, 3 4 5), l) from line3d;

select buffer(c2,'distance=symmetric:2,join=round:10,end=round,point=circle:20') from
line1;

select buffer(c2,'distance=asymmetric:2,join=miter:10,end=flat,point=square:20') from
line1;

select length(c2) from line1;

select perimeter(pol) from pol1;

select perimeter(s1) from sq1;

select equal(s1,s2) from sq1;

select issimple(c2) from line1;

select issimple(pol) from pol1;

select isvalid(pol) from pol1;

select isring(pol) from pol1;

select isring(c2) from line1;

select ispolygonccw(pol) from pol1;

select ispolygoncw(pol) from pol1;

select outerring(pol) from pol1;

select outerrings(p1) from mpg;

select innerrings(p1) from mpg;

select ringn(p1,1) from mpg;

select ringn(p2,2) from mpg;

select ringn(pol,1) from pol1;

select ringn(pol,2) from pol1;
```

```
select innerringn(pol,1) from pol1;

select polygonn(p1,1) from mpg;

select unique(c2) from line1;

select union(c1,c2) from line1;

select union(p1,p2) from mpg;

select union(pol,'polygon((0 0, 2 3, 2 4, 8 2, 3 9, 0 0))') from pol1;

select union(pol,polygon((0 0, 2 3, 2 4, 8 2, 3 9, 0 0))) from pol1;

select collect(pol,polygon((0 0, 2 3, 2 4, 8 2, 3 9, 0 0))) from pol1;

select collect(p1,polygon((0 0, 2 3, 2 4, 8 2, 3 9, 0 0))) from mpg;

select topolygon(c1,30) from cir1;

select topolygon(s1,30) from sq1;

select topolygon(s1) from sq1;

select topolygon(s1) s1pgon, topolygon(s2) s2pgon from sq1;

select text(s1) from sq1;

select difference(line(0 0, 2 2), point(2 2) ) df;

select difference(linestring(0 0, 2 2, 3 4), point(2 2) ) df;

select difference(linestring(0 0, 2 2, 3 4, 4 6), line(2 2, 3 4) ) df;

select difference(pol, polygon((0 0, 8 0, 800 800, 80 80, 0 0),( 3 4, 4 6, 4 2, 3
4 )) ) df from pol1;

select difference( 'polygon((0 0, 8 0, 8 8, 0 8, 0 0),( 3 4, 4 6, 4 2, 3 4 ))', pol )
df from pol1;


select symdifference(line(0 0, 2 2), point(2 2) ) df;

select symdifference(linestring(0 0, 2 2, 3 4), point(2 2) ) df;

select symdifference(linestring(0 0, 2 2, 3 4, 4 6), line(2 2, 3 4) ) df;

select symdifference(linestring(0 0, 2 2, 3 4, 4 6), linestring(2 2, 3 4, 8 9) ) df;

select symdifference(pol, polygon((0 0, 8 0, 800 800, 30 800, 0 0),( 3 4, 4 6, 4 2, 3
4 )) ) df from pol1;

select symdifference(pol, polygon((0 0, 8 0, 800 800, 80 80, 0 0),( 3 4, 4 6, 4 2, 3
4 )) ) df from pol1;

select symdifference( 'polygon((0 0, 8 0, 800 800, 80 80, 0 0),( 3 4, 4 6, 4 2, 3
4 ))', pol ) df from pol1;


select intersection('polygon((0 0, 8 0, 8 8, 0 8, 0 0),( 3 4, 4 6, 4 2, 3 4 ))',
'polygon((1 1, 9 1, 9 9, 1 9, 1 1 ))' ) dd;

select intersection('polygon((0 0, 8 0, 8 8, 0 8, 0 0),( 3 4, 4 6, 4 2, 3 4 ))', p1 )
dd from mpg;
```

```
select intersection(p1, 'polygon((0 0, 8 0, 8 8, 0 8, 0 0),( 3 4, 4 6, 4 2, 3 4 ))'  )
dd from mpg;

select intersection('polygon((0 0, 8 0, 8 8, 0 8, 0 0),( 3 4, 4 6, 4 2, 3 4 ))', p2 )
dd from mpg;



select union('polygon((0 0, 8 0, 8 8, 0 8, 0 0),( 3 4, 4 6, 4 2, 3 4 ))', 'polygon((1
1, 9 1, 9 9, 1 9, 1 1 ))' ) dd;

select union('polygon((0 0, 8 0, 8 8, 0 8, 0 0),( 3 4, 4 6, 4 2, 3 4 ))', p1 ) dd from
mpg;

select union(p1, 'polygon((0 0, 8 0, 8 8, 0 8, 0 0),( 3 4, 4 6, 4 2, 3 4 ))'  ) dd
from mpg;



select isconvex(pol) from pol1;

select interpolate(ls,0.5) from lstr;

select linesubstring(ls,0.2, 0.8 ) from lstr;

select locatepoint(ls, point( 3 9) ) from lstr;

select addpoint( ls, point(234 219) ) from lstr;

select addpoint( ls, point(234 219), 2 ) from lstr;

select setpoint( ls, point(234 219), 1 ) from lstr;

select removepoint( ls, 2 ) from lstr;

select reverse( ls ) from lstr;

select scale( ls, 3 ) from lstr;

select scale( ls, 10, 20 ) from lstr;

select scaleat( ls, point(10 20), 10 ) from lstr;

select scaleat( ls, point(10 20), 10, 20 ) from lstr;

select scalesize( ls, 10 ) from lstr;

select scalesize( ls, 10, 20 ) from lstr;

select scalesize( s1, 10, 20 ) from sq1;

select translate( ls, 10, 20 ) from lstr;

select transscale( ls, 200, 300, 10 ) from lstr;

select transscale( ls, 200, 300, 10, 20 ) from lstr;

select rotate( ls, 180 ) from lstr;

select rotate( ls, 1.0, 'radian' ) from lstr;

select rotateself( ls, 180 ) from lstr;

select rotateself( s1, 180 ) from sq1;

select rotateself( s1, 1.80, 'radian' ) from sq1;
```

```
select rotateat( s1, 1.80, 'radian', 100, 300 ) from sq1;

select rotateat( ls, 1.80, 'radian', 100, 300 ) from lstr;

select affine( ls, 1, 2,3, 4, 500, 600 )  from lstr;


select ls:x, ls:y, ls:m1, ls:m2, ls:m3, ls:m4 from lstrm where a < 10000;

select voronoipolygons(tomultipoint(ls) ) vp from lstrm;

select voronoipolygons(tomultipoint(ls,100) ) vp from lstrm;

select voronoipolygons(tomultipoint(ls),100,bbox(0 80 0.2 80.2) ) vp from lstrm;

select voronoilines(tomultipoint(ls) ) VL from lstrm;

select voronoilines(tomultipoint(ls),100) ) VL from lstrm;

select voronoilines(tomultipoint(ls),100,bbox(0 80 0.2 80.2) ) VL from lstrm;


select delaunaytriangles(tomultipoint(ls) ) dt from lstrm;

select delaunaytriangles(tomultipoint(ls,100) ) dt from lstrm;


select geojson(ls)  from lstrm;

select geojson(ls, 10000)  from lstrm;

select geojson(c1, 10000,300) from cir1;


select tomultipoint(ls) from lstrm;

select tomultipoint(c1, 300) from cir1;


select wkt(ls) from lstrm;

select minimumboundingcircle(ls) from lstrm;

select minimumboundingsphere(pt3) from d5 where a < 1000;


select isonleft(point(30 40), ls) from lstrm;

select leftratio(point(30 40), ls) from lstrm;

select isonright(point(30 40), ls) from lstrm;

select rightratio(point(30 40), ls) from lstrm;


select knn(ls, point(30 40), 10) from lstrm;

select knn(ls, point(30 40), 10, 10, 100) from lstrm;
```

```
select metricn( ls, 2 ) from lstrm;

select metricn( ls, 2, 3 ) from lstrm;
```

Timeseries Data

The following statements demonstrate timeseries data management.

```
create store timeseries(5m) tc1 (  key: k1 int, c1 char(2), ts timestamp, value: v1
rollup int, v2 int );

create store timeseries(5m) ts1 (  key: k1 int, ts timestamp, value: v1 rollup int, v2
int );

insert into ts1 ( k1, v1, v2 ) values ('5', '103', '247' );

insert into ts1 ( k1, v1, v2 ) values ('5', '303', '253' );

insert into ts1 ( k1, v1, v2 ) values ('5', '503', '553' );

insert into ts1 ( k1, v1, v2 ) values ('5', '903', '153' );

insert into ts1 ( k1, v1, v2 ) values ('5', '1903', '153' );

insert into ts1 ( k1, v1, v2 ) values ('6', '10', '29' );

insert into ts1 ( k1, v1, v2 ) values ('6', '100', '29' );

select * from ts1;

select * from ts1@5m;


create store timeseries(10s) ts1002 (  key: k1 int, ts timestamp, value: v1 rollup
int, v2 int );

insert into ts1002 ( k1, v1, v2 ) values ('5', '100', '200' );

select * from ts1002;

select * from ts1002@10s;


drop store ts2;

create store timeseries(5m) ts2 (  key: k1 int, ts timestamp, value: v1 rollup int, v2
int, v3 rollup int );

insert into ts2 values ('5', '2021-02-12 13:35:12', '100', '200', '111' );

select * from ts2;

insert into ts2 values ('5', '2021-02-12 13:35:13', '100', '200', '1123' );

insert into ts2 values ('6', '2021-02-12 13:36:14', '100', '200', '213' );

insert into ts2 values ('7', '2021-02-12 13:36:17', '100', '200', '233' );
```

225

```
insert into ts2 values ('6', '2021-02-12 13:37:17', '100', '200', '322' );

select * from ts2;

select * from ts2@5m;


create index ts2idx1 on ts2(v1, k1);

create index ts2idx2 ticks on ts2(v3, v1, k1);

create index ts2idx3 ticks on ts2(v3, k1, v1);

insert into ts2 (k1, v2 ) values ('10', '243' );

select * from ts2idx1;

select * from ts2idx2;

select * from ts2idx3;


create index ts2idx4 ticks on ts2(key: v3, k1, value: v1);

desc ts2idx4;

select * from ts2idx4;

select * from ts2idx4%5m;



drop store ts2002;

create store timeseries(5m|10m) ts2002 (  key: k1 int, ts timestamp, value: v1 rollup
int, v2 int );

insert into ts2002 values ('5', '2021-02-12 13:35:12', '100', '200' );

insert into ts2002 values ('5', '2021-02-12 13:35:13', '100', '200' );

insert into ts2002 values ('6', '2021-02-12 13:35:14', '100', '200' );

insert into ts2002 values ('6', '2021-02-12 13:36:14', '100', '200' );

insert into ts2002 values ('7', '2021-02-12 13:36:17', '100', '200' );

insert into ts2002 values ('6', '2021-02-12 13:36:17', '100', '200' );

insert into ts2002 values ('6', '2021-02-12 13:37:17', '100', '200' );

select * from ts2002;

select * from ts2002@5m;

create store timeseries(1h:0h) ts3 ( key: k1 int, ts timestamp, k2 int, k3 char(10),
k4 char(12), k5 char(23), value: b rollup int, c int, c2 int, c3 rollup int, d rollup
int, e int, f rollup int );

insert into ts3 values ('1', '2020-12-14 12:12:12', '100', 'k3k', 'k4k', 'k5k',
'200', '300', '400', '456', '222', '333', '321' );
```

```
insert into ts3 values ('2', '2020-12-14 12:12:12', '100', 'k3k', 'k4k', 'k5k',
'200', '300', '400', '456', '222', '333', '321' );

insert into ts3 values ('2', '2020-12-14 12:52:12', '100', 'k3k', 'k4k', 'k5k',
'200', '300', '400', '456', '222', '333', '321' );

insert into ts3 values ('3', '2020-12-14 12:52:12', '100', 'k3k', 'k4k', 'k5k',
'200', '300', '400', '456', '222', '333', '321' );


insert into ts3 values ('2', '2020-12-14 12:13:12', '101', 'k3k', 'k4k', 'k5k',
'202', '304', '400', '457', '223', '353', '421' );

insert into ts3 values ('2', '2020-12-14 12:14:12', '101', 'k3k', 'k4k', 'k5k',
'202', '304', '400', '457', '223', '353', '421' );

select * from ts3;

select * from ts3@1h;


drop store ts4;

create store timeseries(1h:0h, 3M:2y ) ts4 (  key: k1 int, ts timestamp, value: v1
rollup int, v2 int );

insert into ts4 ( k1, v1, v2 ) values ('1', '123', '321' );

insert into ts4 ( k1, v1, v2 ) values ('2', '123', '321' );

insert into ts4 ( k1, v1, v2 ) values ('3', '123', '321' );

insert into ts4 ( k1, v1, v2 ) values ('3', '123', '321' );

select * from ts4;

select * from ts4@1h;

alter store ts4 add tick(1d);


alter store ts4 add tick(1D:10D);

alter store ts4 drop tick(1D);


create index ts4_idx1 on ts4(v1, ts);

select * from ts4_idx1;


create index ts4_idx2 on ts4(ts, k1);

select * from ts4_idx2;


alter store ts4 add tick(1q);

select * from ts4@1q;
```

```
alter store ts4 retention 0;

alter store ts4 retention 12M;

alter store ts4@3M retention 3y;


desc ts4;

desc ts4@3M;


create store timeseries(1d) ts5 (  key: k1 int, ts timestamp, value: v1 rollup int, v2
int );

insert into ts5 values ('5', '2020-12-14 12:13:12', '100', '200' );

insert into ts5 values ('5', '2020-12-14 12:14:12', '100', '200' );

insert into ts5 values ('5', '2020-12-14 12:15:12', '100', '200' );

insert into ts5 values ('5', '2020-12-14 12:16:12', '100', '200' );

insert into ts5 values ('6', '2020-12-14 12:17:12', '100', '200' );

insert into ts5 values ('6', '2020-12-14 12:18:12', '100', '200' );

select * from ts5;

select * from ts5@1d;


create store timeseries(3d) ts5002 (  key: k1 int, ts timestamp, value: v1 rollup int,
v2 int );

insert into ts5002 values ('5', '2020-12-14 12:13:12', '100', '200' );

insert into ts5002 values ('5', '2020-12-14 12:14:12', '100', '200' );

insert into ts5002 values ('6', '2020-12-14 12:17:12', '100', '200' );

insert into ts5002 values ('6', '2020-12-14 12:18:12', '100', '200' );

insert into ts5002 values ('7', '2020-12-15 12:18:12', '100', '200' );

select * from ts5002;

select * from ts5002@3d;


create store timeseries(1w) ts6 (  key: k1 int, ts timestamp, value: v1 rollup int, v2
int );

insert into ts6 values ('5', '2020-12-14 12:13:12', '100', '200' );

insert into ts6 values ('5', '2020-12-14 12:14:12', '100', '200' );

insert into ts6 values ('5', '2020-12-14 12:15:12', '100', '200' );

insert into ts6 values ('5', '2020-12-14 12:16:12', '100', '200' );

insert into ts6 values ('5', '2021-02-04 12:16:12', '100', '200' );
```

```
select * from ts6;

select * from ts6@1w;



create store timeseries(1month) ts7 (  key: k1 int, ts timestamp, value: v1 rollup
int, v2 int );

insert into ts7 values ('5', '2020-12-14 12:13:12', '100', '200' );

insert into ts7 values ('5', '2021-02-04 12:16:12', '100', '200' );

select * from ts7;

select * from ts7@1M;



create store timeseries(1year) ts8 (  key: k1 int, ts timestamp, value: v1 rollup int,
v2 int );

insert into ts8 values ('5', '2020-12-14 12:13:12', '100', '200' );

insert into ts8 values ('5', '2021-02-04 12:16:12', '100', '200' );

select * from ts8;

select * from ts8@1y;



create store timeseries(1decade) ts9 (  key: k1 int, ts timestamp, value: v1 rollup
int, v2 int );

insert into ts9 values ('5', '2020-12-14 12:13:12', '100', '200' );

insert into ts9 values ('5', '2021-02-04 12:16:12', '100', '200' );

select * from ts9;

select * from ts9@1D;



create store timeseries(15s:60s|1h) ts10 ( key: ts timestamp, a int, value: b int
default '1000', c rollup int default '234' );

insert into ts10 ( a )values ( 100 );

insert into ts10 ( a )values ( 200 );

insert into ts10 ( a )values ( 300 );

insert into ts10 ( a )values ( 400 );

insert into ts10 ( a )values ( 600 );

insert into ts10 ( a )values ( 700 );

insert into ts10 values ( 600 );

select * from ts10;

select * from ts10@15s;

drop store tss1001;
```

```
create store tss1001 ( key: ts timestamp, a int, value: b int default '1000' );

insert into tss1001 ( a )values ( 100 );

select * from tss1001;


drop store tss1;

create store tss1 ( key: a int, value: b timestamp );

insert into tss1 values ( 100 );

select * from tss1;


drop store tss2;

create store tss2 ( key: a int, value: b timestampsec );

insert into tss2 values ( 100 );

insert into tss2 values ( 200 );

select * from tss2;


drop store tss3;

create store tss3 ( key: a int, value: b timestampnano );

insert into tss3 values ( 100 );

insert into tss3 values ( 200 );

select * from tss3;


drop store tss4;

create store tss4 ( key: a int, value: b timestampmill );

insert into tss4 values ( 100 );

insert into tss4 values ( 200 );

insert into tss4 values ( 300 );

select * from tss4;

drop store tspace1;

create store timeseries(5m) tspace1 (  key: k1 int, ts timestamp, loc point, k2 int
default '23', value: v1 rollup int, v2 int, v3 rollup int );

insert into tspace1 (k1, loc, v2 ) values ('10', point(2 3), '243' );

insert into tspace1 (k1, loc, v2, v1, v3 ) values ('10', point(2 3), '243', '1222',
'3456' );

insert into tspace1 (k1, loc, v2, v1, v3 ) values ('11', point(4 5), '643', '2222',
'4456' );
```

```
insert into tspace1 (k1, loc, v2, v1, v3 ) values ('12', point(4 5), '643', '2222',
'4456' );

select * from tspace1;

select * from tspace1@5m;

select * from tspace1@5m where nearby(loc, point(34 12), 100  ) and k1=12;


create index tspace1idx1 on tspace1(v1, k1);

create index tspace1idx2 on tspace1(v3, v1, k1);

create index tspace1idx3 on tspace1(v3, k1, v1);

select * from tspace1idx1;

select * from tspace1idx1@5m;

select * from tspace1idx2;

select * from tspace1idx3;


drop store tspace2;

create store timeseries(5m) tspace2 (  key: k1 int, ts timestamp, loc circle, k2 int
default '23', value: v1 rollup int, v2 int, v3 rollup int );

insert into tspace2 (k1, loc, v2, v1, v3 ) values ('10', circle(2 3 30), '243',
'1292', '3456' );

insert into tspace2 (k1, loc, v2, v1, v3 ) values ('11', circle(4 5 50), '643',
'2262', '4456' );

insert into tspace2 (k1, loc, v2, v1, v3 ) values ('12', circle(4 5 45), '645',
'2422', '4056' );

select * from tspace2;

select * from tspace2@5m;

select * from tspace2@5m where nearby(loc, point(34 12), 100  ) and k1=12;

select * from tspace2@5m where nearby(loc, point(34 30), 40  ) and k1=12;


create index tspace2idx1 on tspace2(v1, k1);

create index tspace2idx2 on tspace2(v3, v1, k1);

create index tspace2idx3 on tspace2(v3, k1, v1);

select * from tspace2idx1;

select * from tspace2idx1@5m;

select * from tspace2idx2;

select * from tspace2idx3;

select * from tspace2idx3 where v3=4456;
```

```
drop store sensorstat;

create store timeseries(5m:1d,1h:48h,1d:3M,1M:20y|5y)

sensorstat (key: sensorID char(16), ts timestamp,

            value: temperature rollup float,

                   pressure rollup float,

                   windspeed rollup float,

                   rpm  rollup float,

                   fuel rollup float,

                   model char(16),

                   type  char(16)

);

insert into sensorstat (sensorid, temperature, pressure, windspeed, rpm, fuel, model,
type ) values ( 'drone1-sid1', '20.0', '35.5', '30.2', '1300', '1.3', 'AA212', 'DH' );


insert into sensorstat (sensorid, temperature, pressure, windspeed, rpm, fuel, model,
type ) values ( 'drone1-sid1', '20.5', '35.8', '30.7', '1320', '1.5', 'AA212', 'DH' );

insert into sensorstat (sensorid, temperature, pressure, windspeed, rpm, fuel, model,
type ) values ( 'drone1-sid2', '21.0', '35.7', '30.8', '1304', '1.2', 'AA213', 'DH' );

insert into sensorstat (sensorid, temperature, pressure, windspeed, rpm, fuel, model,
type ) values ( 'drone2-sid1', '22.0', '36.4', '30.3', '1404', '2.2', 'AB213', 'DF' );


drop store delivery;

create store timeseries(1M:1y,1y)

    delivery (key: ts timestamp, courier char(32), customer char(32),

              value: meals rollup bigint, addr char(128) );


insert into delivery ( courier, customer, meals, addr ) values ( 'QDEX', 'JohnDoe',
'3', '110 A Street, CA 90222' );

insert into delivery ( courier, customer, meals, addr ) values ( 'QDEX', 'JaneDoe',
'5', '110 B Street, CA 90001' );

insert into delivery ( courier, customer, meals, addr ) values ( 'QSEND', 'MaryAnn',
'3', '100 C Street, CA 92220' );

insert into delivery ( courier, customer, meals, addr ) values ( 'QSEND', 'PaulD',
'12', '550 Ivy Road, CA 90221' );


select * from delivery;
```

```
select * from delivery@1M;

select * from delivery@1y;

create index delivery_index_courier on delivery(courier, customer, meals );

select * from delivery_index_courier;

select * from delivery_index_courier@1M;

select * from delivery_index_courier@1M where courier='*' and customer='JohnDoe';

select * from delivery_index_courier@1y;


create index delivery_index2_courier on delivery@1M(courier, customer, meals::min,
meals::max, meals::sum );

select * from delivery_index2_courier;

select * from delivery_index2_courier where courier='*' and customer='PaulD';
```