Geoparquet in person meetup at Data + AI

Over the past few years, GeoParquet has seen significant adoption by major software products and data providers, signaling a robust growth trajectory. Concurrently, there's a burgeoning interest in integrating geo support more natively into Parquet, as well as other related formats and protocols within this ecosystem. While this is undoubtedly positive, it presents a unique set of challenges. Without a swift and unified approach to incorporating geo functionality, there's a real risk of fragmentation in geospatial support across different initiatives.

To address this, we are convening an in-person meetup aimed at fostering a collaborative dialogue among all stakeholders. Our goal is to forge a coherent vision for making GEO a primary data type across various projects. This gathering will serve as a platform to present and align on this vision, ensuring a unified approach moving forward.

The target technologies we are looking at are:

ARROW - geoarrow
PARQUET - geoparquet
ICEBERG
DELTA LAKE
HUDI
AVRO
ORCx

When: Thursday June 13th at 8AM - 9:30AM / 5PM - 6:30PM CET

Where: Planet office, 645 Harrison Street, 4th
Floor, San Francisco, CA 94107.
Remote: Possible to joining remotely Geoparquet in person meetup at Data + AI
Thursday, June 13 · 8:00 – 9:30am https://planet.zoom.us/j/98864285995
Register: To register just add your name to the list of attendees here.

To get in at Planet Labs you can try to go in and talk to the security guard and give them your name, I tried to get everyone on the list. They'll send you down to the elevator. The 4th floor is key carded, but I'll try to send it up.

Just fill out the iPad form and say your are here to see Chris Holmes on it, and I'll come get you.

The easiest will likely be to just text / call me (Chris Holmes) at 718-290-5730 when you're out front. I may come down and wait for some people.

AGENDA:

1. Overall intro to what all these formats/protocols are
2. Intro to existing initiatives and state of geo on each of them,
    a. arrow - geoarrow
    b. Parquet - geoparquet and native geo type
    c. Iceberg - current proposal
    d. Delta lake - I don't think there is currently geo
    e. Hudi - I don't think there is currently any geo
    f. Avro,orc - no primary support
3. What could be a set of common recommendations to try to add geo to each of these based on geoparquet experience.

Attendees:

Chris Holmes - Planet
Javier de la Torre - CARTO
Milos Colic - CARTO
Jia Yu - Wherobots
Ben Pruden - Wherobots
Matthew Topol - Voltron Data
Ian Cook - Voltron Data
Kristin Cowalcijk (remote) - Wherobots
Feng Zhang（remote) - Wherobots
Joris Van den Bossche (remote) - Voltron Data
Jake Wasserman (remote) - Meta
Sergey Sukov - Action Engine
Dewey Dunnington (remote) - Voltron Data
John Powell (remote) - Addresscloud
Matt Travis (remote) - Addresscloud
Mark
Shoaib Burq (remote) - geobase.app

NOTES

Javi: arrow -> encoding, parquet -> data type, iceberg -> operators and usage api

Jia: Iceberg adds another layer of table metadata, that lets you define table name, database name, etc. And allows you to perform db operations - insertion, deletion, upsert, etc in a cloud environment. Enables 'lake house' / 'data lake'. In havasu product we added geospatial support into Havasu. Added 2 things - 1) how to store data in iceberg. For storing data in iceberg, and

for that we used GeoParquet. For raster we did our own format. And then on top of that added table operations, including basic filter pushdown, and also implemented a writer. Iceberg needs to interface with a format.

Javi - in iceberg do you define st_ functions?

Jia - no, it's just how you store stuff.

Milo - it's a protocol, not a format. Historically the issue is that parquet has lots of files, and if there's multiple threads you can have partially executed jobs. So added transaction logs for concurrent writes, snapshots, etc. Like having a database, to do those operations, but not having a database - no always on service.

Jia/Matt/Milos - If you don't care about spatial filter pushdown then iceberg does not need any expression, the engine covers it. Iceberg defines partition expressions - it provides a list of transforms, so you can do partitions not just on exact value of column, but also a transform of a column. In the geo iceberg proposal it defines that bit, so you don't have to do full scans.

Jia - To enable filter pushdown you only need one expression - st_intersects. All others can leverage that. 'Are there rows in this file that might match' (Matt).

Javi - How did proposal start with apple?

- Jia - a few months back we (Jia and Javi) talked to Tabular, they said happy to help, we're review proposal, but can't put much time in. But Apple was very interested, contacted engineers looked at design doc and havasu implementation. Interested for apple maps, they have huge amounts of data, they're using Sedona and iceberg, and really want to have it in iceberg. They have a number of iceberg PMC members. The proposal is not identical to Havasu - community wants some change to make it into iceberg.
    - Don't want properties in the file metadata, they want things in table properties. Unless those properties become standard in Parquet - they don't want it in their metadata.
        - Matt - iceberg as a spec, it's not valid to consume any individual file in the table without going through iceberg. So they want it at the iceberg table level. That's why it'd be ideal to get the geometry types in the parquet spec, and then iceberg could add a geometry type that matches / is similar to the.
        - Milos - important to understand that at the file level many files are 'not valid', since things got deleted, and so if you tried to read the iceberg parquet file it could be very bad.
            - Joris - it may not really matter, as iceberg could put the table in their data properties. They don't really need that native type. It is useful for the file to encode the geometry and use the statistics.
            - Jia - yes, if you don't care about other properties it does not need native geometry type.
            - Milos - you descend into implementation by convention, and having a native type could avoid some of those potential problems.

        - Matt - many iceberg implementations do different things. Some have their own parquet reader. Like Joris said it may not matter at the iceberg level. But benefit of having it in the underlying file. In Snowflake you can just throw all your parquet files and have it import it. So proper translation between parquet geometry type and what cloud vendors would result. Some people may just grab files and say ‘make me a table from these files’.
    - Jia - they also do not like the geoparquet 1.1 native encoding, since it doesn’t have a fixed schema, and doesn’t support geometry (?)
        - Joris - can you explain more about the requirement of a fixed schema?
        - Jia - what I understand from iceberg PMC is that the current encoding in 1.1, if you use different point, line, polygon - it can differ.
        - Joris - to clarify, you only use those more specific types if you know you only have points. If you know it’s not the case you don’t use it.
        - Milos - but it’s polymorphism, the child types aren’t converted to one another. One is a collection of tuples, one is a collection of collection. So that gets complicated. What they want is a single fixed encoding.
        - Joris - if you want those types then you just need those. There’s clearly a big benefit for points.
        - Matt - for a given column you want to define a type.
            - Milos - a predicate can generate a mixture of types, the output can be point / multipoint, depending on the system. So you could end up with all types of geometry.
            - Matt - how do you solve this in geo?
                - Postgis / sedona just allow different rows to have different geometry.
        - Joris - it’s true 1.1 spec only included specific types. But nothing prevents us (and have been discussing) to have an additional type that represents a geometry or geometry collection that supports all those types. If you know your operations might return various types you can use the generic one.

- Joris on Arrow / GeoArrow
    - At core arrow is specification for how to represent tabular data in memory. If you consider an array in memory that’s a buffer and you point at the start of the buffer then every library understands it. Arrow is nothing more than that, just for more complex data.
    - There’s lots of functionality in the core arrow libraries, which sometimes makes it a bit more complicated.

- So many languages have arrow implementations. Since it's the same format in memory you get no deserialization / 0 copy transfer between languages.
- It's actually a collection of buffers. All arrow types have a validity bitmap, that's just 'where are your nulls'.
    - So arrow defines data types?
    - Yes - arrow spec, format, and then all the implementations.
- GeoArrow is an extension type. Which is some underlying existing type in the arrow spec, and you add a little bit of metadata with a specific key, and then you can define in your own library how you want to handle that extension type. That extension type can pass through any system that understands arrow, even if it doesn't understand the extension.
- So GeoArrow is implemented as a series of extension types. It defines that set, which name to use, and which metadata.
- There are two big groups
    - - serialized formats (wkb or wkt), but you can still annotate them with an extension type. So if you transfer arrow data, even if it's using wkb, you can attach that information, like about the coordinate referenced system. Just uses binary type in arrow.
        - You're using WKB with CRS? Not EKWB? Only supports proj?
        - Lots of discussion in GeoParquet. In GeoArrow CRS key should have projjson. But not sure if we merged that PR. For GeoArrow we want to be more flexible - you might read data from somewhere that has a CRS description that's not in projjson. So we still want to pass along that information.
    - Native formats, where we store raw coordinates, with a nested list of lists, depending on type. So those are geometry type specific. And have been discussing one for geometry collection / geometry union.
- Javi - if arrow has all the type info do we need geoparquet?
    - Matt - yes, they map to one another.
- Dewey - 2 things going on in geoarrow
    - Get data from one place to another. Keep everything geo, so it's not dropped. That's where CRS at the type level comes in. That's why parquet having a geometry type is great, to be sure the crs wouldn't get dropped. Data would pass right through.
    - Also provides better options for dealing with points, because binary for points is just unimportant.
        - Milos - big advantage of EWKB. So keeping it as mandatory part of geometry is very important.
        - Dewey - valid use case on satellite data, where each is a different UTM zone.
        - Jia - what we do is crs in each, but bounding box is always wgs 84. Helps with performance.
- Jia - GeoArrow is not in Arrow spec, right?

        - Matt. Yes - there are a few extension types that are canonical in the arrow spec. Great example is UUID. It's a canonical extension type - fixed binary, 16. Not a type in the built in arrow sense, but everyone agrees. So we have some things that are canonical in the community in the spec, and then others are 'do whatever you want', but all will pass through.
        - Jia - ongoing extension to allow extension type, which would help the 'big geo to parquet' problem.

- Javi - what's the overall idea / ideal.
    - Matt - ideally you have geometry types in iceberg spec, with defined geometry types in geoparquet, defined in geoarrow. ANd you have ADBC driver that can query Snowflake, use GeoArrow types to return from Snowflake's geometry type, that stream of arrow record batches can go where ever it needs to. And then it could write to iceberg / geoparquet. Everyone agrees how to represent it at these levels. Then any system that wants to read from any of those then data transfer is in geoarrow.
        - What does this require at each?
            - We have geoarrow - we don't need to have a canonical extension type. Only benefit of 'canonical' extension type is to give incentive / reason for others. Not required.
            - CH - is it a goal to be a canonical extension type?
                - Joris / Matt - no, because the canonical extension type is new, so we'd have to change the name. But it's listed on the main 'extension' page.
                - Dewey - one weakness is the 'geoarrow' community is that the governance is 'weak'. It's just kyle/dewey/Joris. So it may be good to have it on the canonical list to give some assurance that the PMC is going to maintain. (general agreement from others that would be better).
                - Javi - we could focus on well known binary. And have iceberg work with that. Lots of agreement.
                - Joris - terminology is not always clear. But most practical usage of GeoArrow is with the well known binary type. But we use geoarrow to refer to the native type. But if you ask GDAL to export data into arrow format it will give you a geoarrow well known binary type. But the fact it preserves the information is the value of it, even though it's using WKB.
    - Milos - ideal state is you have an object that is full self contained, so a column represents itself, and naming convention drives that behavior. If you have a parcel with a house, and you have geometries for parcel and building, then it's difficult - what does WKB mean. Schema level conventions are good, since they're quicker. That's important, with arrow you keep appending buffers, so there's not a big tax.

        - Matt - because arrow represents complex types you could have an extension type as a ‘struct has these fields’.
        - Milos - perhaps a ‘cloud native representation type’ - always start with bounding box, follow by CRS, follow by lines.
        - Matt - for type, the representation could be ‘struct of these types with these names and these children’. Then parquet has structs, iceberg has structs. Then expected layout of new type is complex definition is particular fields with particular types. This is ‘geometry blah’, and we’ve defined its a struct of these fields.
- Javi - seems like we should have two tracks - one on WKB, one on more native types.
-

-

REQUIREMENTS:

-CHRIS: Enable data producers to distribute without having to create disgtributions for parquet and iceberg.

-Support for all metadata we created on geoparquet where it matters.

BARRIERS:

-Apple wants something geoarrow cant do

-It is hard to get a new data type in Parquet

-

Other:

-Possible double track with WKB and geoarrow?