Support Spatial Join For Query

BackGround

Inorder to support in_polygon udf as a join condition for spatial index queries.

Currently, Carbon supports point and polygon query. With polygon query, carbon supports the following UDF's:
1. IN_POLYGON udf which takes a series of points (latitude and longitude) when plotted and connected forms a closed geometry object and the query will fetch all the points lying within the closed geometry object.
2. IN_POLYGON_LIST udf with multiple polygons that contains a series of points (i.e., longitude and latitude) as filter condition. Query will fetch all the points lying within the closed geometry object which is combined of those polygons with specified operation type.

With above two UDF's, the user has to provide the latitude and longitude points with the query, to fetch all the points lying within the closed geometry object. Currently, carbon does not support JOIN queries, where join condition is, IN_POLYGON filter on a table, where all the polygons exist.

Refer to "CarbonData Spatial Index Design Doc v2.docx" Attachment in the link: https://issues.apache.org/jira/projects/CARBONDATA/issues/CARBONDATA-4051?filter=allissues

Proposed Solution

Support a new UDF, which applies an IN_POLYGON filter on Table having Geo-spatial index for all the polygons present in the polygon table.

Polygon Query with JOIN:

User query can contain IN_POLYGON_JOIN UDF with Geo-spatial table name and polygon column from polygon table, as filter condition. Polygon column will have a series of points (i.e., longitude and latitude columns). First and last points in each row of the polygon table being the same. All the points of each row in the polygon column, when plotted and connected forms a closed geometry object. And query is used to fetch all the points lying within the closed geometry object.

Tables: T1, T2

T1 table:

x	latitude	longitude
x1	lat1	lon1
x2	lat2	lon2

T2 table:

Polygon	poitype	poly_Id
polygon(lon1 lat1, ..)	..	1
polygon(lonX latX, ..)	..	2

Query:

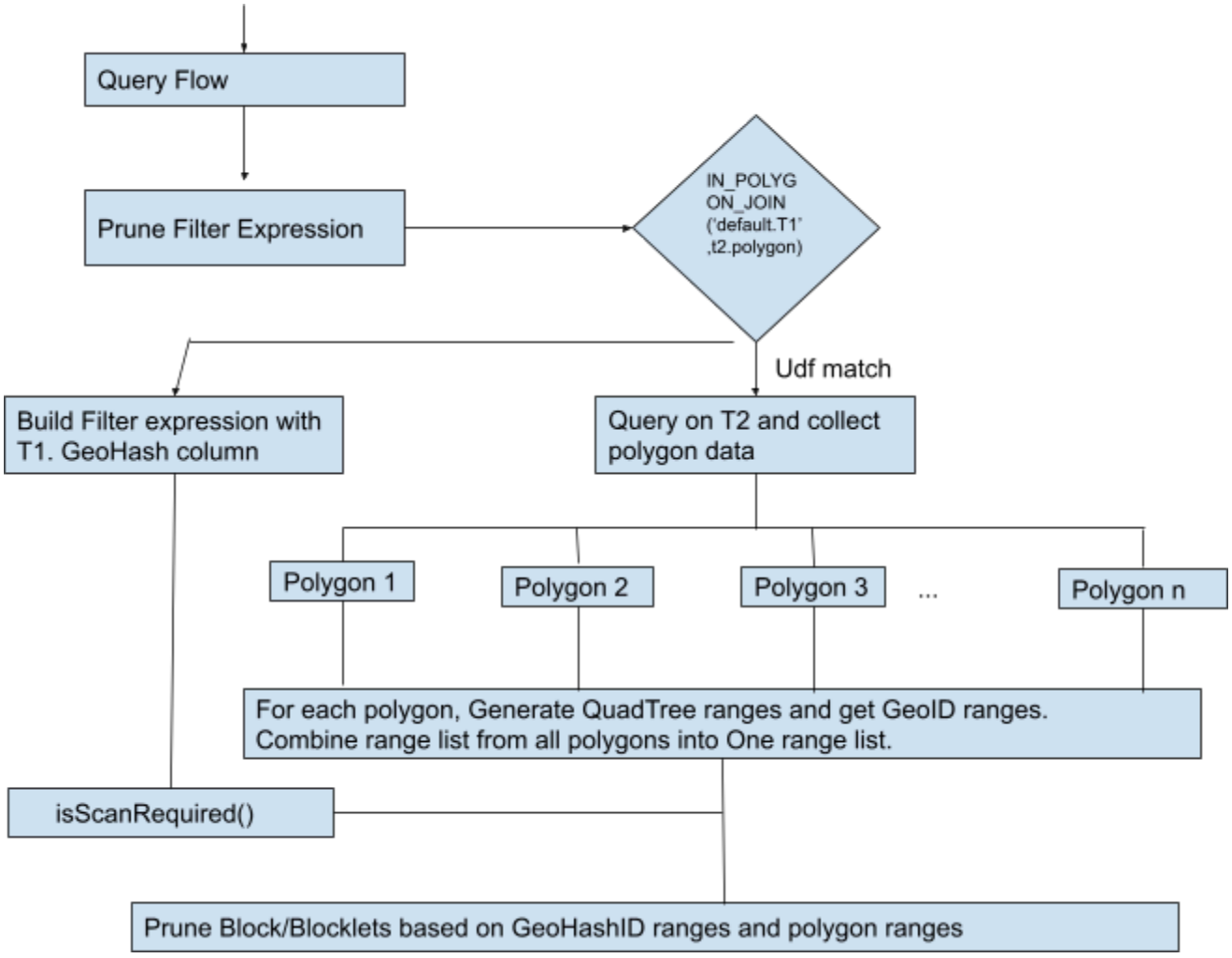
```
select sum(t1.x),t2.poly_Id
from table1 t1
inner join
(select polygon,poly_Id from table2_p where poitype='xxx') t2
on in_polygon_join('default.table1', 'default.table2_p.polygon')
group by t2.poly_Id
```

Interface Description:

IN_POLYGON_JOIN('dbName.tblName', polygon_column)

Input Parameters	**Type**	**Description**
dbName.tblName	String	Geo spatial table name
polygon_column	Table_Column<String>	Polygon column in table. Data present in polygon column: Input polygon list as a string in a defined format.For example, one polygon(a row) is *POLYGON ((longitude1 latitude1, longitude2 latitude2, …))*

Implementation:



Following steps are in involved in query processing:

1. Get all the polygons by querying the polygon column from the polygon table.

2. Respectively generate the QuadTree spatial ranges from the input each polygon in the fixed initial bounding rectangle, and get the GeoId range lists, each list representing a polygon region.

3. Combine the range lists to one range list (Calculate the union of all polygons). Each element in the list will have minimum and maximum GeoId values for that range.

4. Build the filter expression with the geohash column from Table provided, containing all the GeoIds of all the ranges to be filtered.

5. Prune block/blocklet using filter expression. And get the points in the identified block/blockets.