Integration of Spatial Data in Database Acceleration for Analytics

Knut Stolze* Josepine Rückert* Frank Butt†

Abstract:

The IBM DB2 Analytics Accelerator (IDAA) integrates the strong OLTP capabilities of DB2 for z/OS with very fast processing of OLAP and analytical SQL workload in Netezza. Today, not all data types supported by DB2 are available for acceleration. One step to narrow this gap is to enable spatial data - in form of polygons, linestrings, and points - for OLAP processing. Since DB2 and Netezza have very different internal architectures and different limitations, a straightforward mapping from the DB2 data type to Netezza data types is not possible.

We developed a prototype geared toward performance and simplicity for spatial data support in IDAA. In particular, all SQL statement that work against DB2 shall work against the accelerator without any changes. In this paper, we describe how the integration with the accelerator is accomplished so that ingestion and access to the spatial data becomes as seamless and transparent as possible for existing applications. The architecture and implementation aspects for spatial data representation in Netezza as well as the access during query processing are discussed in detail.

1 Introduction

Data warehouses have evolved from dedicated systems using specialized schemas like star and snowflake [Leh03] to solutions that strive for a full integration with operational data. Limiting the database schema in those ways gives a good theoretical model, but it does not always fit well with real (reporting) applications. For instance, the IBM® data warehouse industry model for banking and financial markets [IBM12] is neither a star nor snowflake schema, but surely it is intended for data warehousing. Along with such schema-based restrictions came many other requirements and limitations. For example, typical data warehouses and OLAP systems support only a subset of data types like integers, floating point numbers, and varcharS. However, emerging integrated systems that target OLTP and OLAP with equally excellent performance cannot allow for such restrictions. Existing applications use data types like large objects (LOBs), XML, or spatial data.

In this paper, we present our works on extending the IBM® DB2 Analytics Accelerator (IDAA) [BAF+14] to support spatial data. IDAA is a hybrid system, which uses DB2® for z/OS® [IBM14b] for transactional workload. A copy of the data resides in the accelerator, which is based on Netezza technology [Fra11], and deals with analytical workload in a

* IBM Germany Research & Development, Böblingen, Germany
† IBM Silicon Valley Lab, San Jose, USA
high performing way. The DB2 optimizer is responsible for the query routing decision, i.e., whether to run the query in DB2 itself or to offload it to the accelerator. For us, this architecture is a stepping stone towards a single operational data store [Inm99]. The benefits can be found in an overall reduced complexity of the IT infrastructure and better integration, all reducing costs for exploiters.

Many application areas take advantage of storing, retrieval, and querying of geometric, geographic, and spatial data. The data space for geographic data is the surface of the Earth, possibly as a two-dimensional abstraction (projection). In geographic contexts, it models static objects in the real world like streets, buildings, political boundaries, etc. Moving objects like ships or cars with a changing location can also be described. Other examples are disaster control and emergency management requiring information about the exact location of a catastrophe or accident and details about the available infrastructure like roads and railways. Yet another important user group are military entities, who use spatial data for centuries already for their strategic and tactical planning.

![Figure 1: Visualized Map Data Extracted from a DB2 Database System](image)

DB2 for z/OS as well as Netezza already have support for spatial data. Both follow the SQL/MM: Spatial standard [ISO03a, Sto03]. However, there are still differences in the scope of supported functionality and the storage of spatial objects, especially size limitations. Those differences must be solved together with processing of spatial queries in IDAA and high performing data ingestion for a fully functional product feature.

In all our efforts, the overriding design principle for providing acceleration capabilities of spatial data was to avoid changes or requiring rewrites of an application’s SQL statements. A spatial query send to DB2 is rewritten by the DB2 optimizer to the Netezza SQL dialect and then passed on to IDAA, which delegates the actual execution to the Netezza backend. The data structures in Netezza are designed in a way that is just based on function mapping, but without any further changes to the original SQL statement. That means, DB2 is not responsible for introducing additional joins, for example.

The remainder of the paper is structured as follows. The architecture of the IBM DB2 Analytics Accelerator is summarized in section 2. Section 3 gives a very brief overview on Netezza spatial and DB2 spatial, highlighting details that are important for the integration with IDAA. Our new approach to deal with space limitations in Netezza and the related integration of spatial support in IDAA are presented in section 4, together with

\[1\] Henceforth, the terms geometry, spatial data, and spatial information are used synonymously.
some preliminary performance measurements based on a prototype implementation currently under way. Finally, the paper concludes in section 5 with a summary and general outlook to future direction for the development of this new product feature.

2 Overview on the IBM DB2 Analytics Accelerator

IDAA [BAF+14] is based on the Netezza appliance [Fra11], which is used as backend. It provides the data storage capabilities to manage large amounts of data and to provide exceptional performance for querying that data. IDAA’s functionality is extended over time to better satisfy customer requirements. [Sto13]

Figure 2 illustrates the high-level architecture of IDAA, also called Data Warehouse Accelerator. An additional process (DWA) – that implements the integration layer with DB2 for z/OS – runs on the Netezza hardware and operating system. This integration layer is the entry point for all requests originating either from DB2 or IDAA stored procedures running in DB2. The DRDA protocol [DRD03] is used for the communication between both hardware platforms. Requests to execute queries are passed from DB2 to the Netezza backend by translating DRDA protocol to CLI/ODBC. Administrative requests, e.g., to provide a list of accelerated tables, are handled in DWA itself. If necessary, SQL queries against the Netezza backend are executed to collect backend-related meta data and/or statistical information.

**Figure 2:** IDAA System Architecture

It is possible to associate multiple accelerators with a single DB2 system in order to establish an environment that supports high availability and disaster recovery. An appropriate workload balancing is applied by DB2 in case the connected accelerators have different hardware and workload characteristics. Similarly, a single accelerator can be connected to multiple DB2 systems, sharing its resources. Another workload balancing layer is applied on that level.
Aside from query processing, it is also necessary to maintain and refresh the snapshot of the data in the accelerator in case the original data in DB2 changes. IDAA offers 3 options for the data refresh, which are explained in more detail.

The IBM DB2 Analytics Accelerator comes with an extremely fast loader for whole tables or (a set of) partitions thereof. Refreshing the data of an entire table (cf. figure 3) is typically done for rather static data and for non-partitioned tables. Partition-based load (cf. figure 4) is targeted at partitioned tables, where updates are performed rather infrequently and only to a small subset of partitions.

Figure 3: Full Table Refresh  Figure 4: Partition Update

If a low latency for the data currency in IDAA is not very important and queries return acceptable results, even if the data is slightly outdated by a few minutes or hours, both options are viable. Additionally, the initial load of the data into the accelerator is accomplished that way with very good performance, exceeding a throughput of 1.5 TB/h and maxing out at the network bandwidth between DB2 and IDAA.

For tables with a higher update frequency and where a high data currency on the accelerator is desired, a third option for data maintenance is the Incremental Update feature (cf. figure 5). Incremental update is based on replication technology [BNP+12], which reads DB2 logs and extracts all changes to accelerated DB2 tables. A latency of about 1 minute is achieved, which is usually fast enough for reporting systems and data warehouses. That is even more true considering complex accelerated queries may take several minutes (or hours) in DB2 and still a few seconds with IDAA.

Figure 5: Refreshing Table Data With Incremental Update

It is the responsibility of the database user/administrator to trigger the refresh of the data in each accelerator individually (or to set up incremental update where appropriate). Since many customers have thousands of tables in DB2 and also in IDAA, it is important to understand query access patterns to the individual tables and monitoring the accelerated workload.
3 Spatial Data

Points, linestrings, and polygons are the basic building blocks for storing spatial data in relational database systems. The geometries are treated as scalar values with a dedicated set of functions. Some systems, e.g., DB2 for LUW, employ structured types and establish a hierarchy while others just have a single data type. Other systems use large objects (BLOBs) and encode all details required to represent the geometry in an opaque way. This does not require complex infrastructure for object-relational extensions. DB2 for z/OS and Netezza have gone that route. [Sto06]

There are over 100 spatial functions, which can loosely be categorized in:

- convert between geometries and external data formats,
- retrieve properties/attributes or measures from a geometry,
- compare two geometries with respect to their spatial relationship, and
- generate new geometries from others.

Probably, the most important category is the comparison based on a spatial relationship since those are used for spatial joins. Specific functions exist to test whether 2 geometries are disjoint, touch each other, overlap, or if one is completely contained in the other.

A few systems have extended their spatial support even further and introduced topology and network models [Ora05]. Those features will not be further discussed here.

3.1 Spatial Support in DB2 z/OS vs. Netezza

DB2 for z/OS has introduced a built-in data type ST_Geometry [IBM14a], which is internally a derivate of a BLOB. The binary representation holds all information about the specific type (point vs. linestring vs. polygon), the actual coordinates of the points defining the geometry, and additional attributes like the minimum and maximum X and Y coordinates. The (compressed) binary representation of geometries is limited in size to a maximum of 2 MB. This is sufficient for most practical considerations.

Netezza does not support large objects or structured types. Instead, Netezza’s spatial feature [IBM14c] stores each geometry as a scalar value using VARBINARY underneath. The
maximum size for a scalar value is limited by the 64K page size. The size and precision of geometry data can vary considerably. There are geometries that exceed the page size limit very quickly, e.g. one instance of the border lines of the USA needs 300 K (compressed) alone. In such cases, the too large geometries are split (chopped) into multiple, smaller but consistent geometries when the data is inserted into the Netezza tables by a separate tool [Sof14]. Naturally, that splitting adds overhead.

Figure 7: Example for Chopping Geometries (Source: [Sof14])

Furthermore, not all spatial functions can be applied correctly to the individual, smaller geometries. An example is shown in figure 8. Assuming that the octagon in figure 8(a) is actually a much more complex geometry, it is broken down into several smaller geometries as figure 8(b) demonstrates. Attempting to query those multiple geometries may yield incorrect results as is illustrated using figure 8(c). If the spatial predicate is `ST_Within` and tests whether the blue query rectangle is fully contained within the (yellow) geometry, the answers on the smaller geometries are all false and there is no way to determine the correct answer of true without recomposing the original geometry. Due to these issues, Netezza does not provide all spatial functions.

Figure 8: Functional Problems with Breaking Down Geometries
Our approach below avoids the functional problems for spatial support in IDAA. We take advantage of user-defined aggregate functions, table functions, and the presence of IDAA as a layer on top of the Netezza database system, which is running on the same system as Netezza itself. That allows us to properly handle geometries of all sizes when loading data from DB2 into IDAA and also when querying the data with any of the spatial predicates supported by DB2.

4 Integrating Spatial Data in IDAA

The fundamental differences for spatial support in DB2 and Netezza (cf. section 3.1) are really major obstacles. Breaking down and splitting geometries loses key functionality. Additionally, the overhead for the splitting during ingestion is not acceptable from a performance point of view. Therefore, our chosen path is not a light-weight integration of the existing products but rather a deep integration of spatial functionality in IDAA.

4.1 Storage in Netezza

Our first step resolves the size limitations of Netezza spatial. Naturally, it is still necessary to break down large geometries in such a way that each part can fit into a single scalar VARBINARY value. But instead of constructing several smaller, consistent geometries, we take the binary representation of the geometry in DB2 and chunk that binary representation. Figure 9 illustrates this. The individual parts (or chunks) are all equally sized (except the last one), and each chunk fits into a VARBINARY value.

![Figure 9: Binary Chunking for Netezza](image)

The first consequence of the chunking is that we can usually not store a geometry together with all its non-spatial attributes (e.g., street name) in one relational table. The chunked geometries are stored in an additional side table.\(^2\) Denormalization would be an alternate, but not desirable option.

```
-- user table
CREATE TABLE country {
   id INTEGER NOT NULL PRIMARY KEY,
```

\(^2\)Side tables are also the means used by Netezza spatial for the same purpose.
Theoreticaly, it woul be posible to store (large) geometries only in the file system and a reference in the user table [ISO03b]. The side table could be avoided. However, all issues related to managing data external to the database system (like proper cleanup, replication for high availability, security, ...) come into play. The complexity and implementation efforts exceed the benefits by a fair margin.

4.2 Spatial Data Ingestion

With the storage structure being defined, the first question arises with the insertion of the data into the Netezza tables when copying it from DB2. While the chunking could be done by IDAA itself, the same functionality will be needed in other contexts as well. For instance, if the spatial function ST_Union is used in a query, the resulting geometry may become too large and exceed the 64K page size limit again. If that geometry shall be inserted into another Netezza table, it has to be chunked. Therefore, we have opted to implement the chunking via SQL table functions in Netezza itself. The table function ChunkGeometry receives the geometry as input and simply generates the run of contiguous chunks where each chunk has an appropriate length. Those chunks are inserted into the side table

```
INSERT INTO country_side SELECT * FROM TABLE ( ChunkGeometry(id, <geom>) ) AS chunks(id, chunk)
```

Listing 2: Inserting geometries in side tables

**Geometry Locators** Since the chunking is realized by IDAA in the Netezza backend itself, it is necessary to transfer the geometry data to Netezza before the table function can be called. But Netezza does not allow for any data value to exceed the page size limit. In order to still be able to transfer larger values from IDAA to Netezza, we use geometry locators – a technique similar to LOB locators.

A geometry locator (cf. figure 10) is a means to describe different storage formats. The first byte in the locator’s data identifies the specific format. If a geometry is small enough,
it is directly embedded \textit{(inline)} and we use an indicator of 0x00. Larger geometries are stored in a file in the file system (or in memory). The file based transfer informs the \texttt{ChunkGeometry} function about the file name and the function reads the data from there based on indicator 0x01.

\begin{verbatim}
geometric value 'inline storage'
  0 data
  e.g. '0x00c38d15d6289684c'

geometric value 'file storage'
  1 filepath
  2 offset
  e.g. '/home/logkit/spacecore/file.txt1.83'

geometric value 'memory storage'
  1 ip-address
  2 process-id
  3 memory-location
  e.g. '2192.168.0.121:2350:0:0:0:0:0:0:0.00000000000000000001'

geometric value 'compressed storage'
  1 compressed-data
  e.g. '39bouf58woscsx94icea'
\end{verbatim}

Figure 10: Geometry Locator

Of course, other options are possible and figure 10 also shows a compressed inline storage. Although, the geometry data is already compressed, it could happen that the binary data exceeds the page size limit only slightly. Another compression step, e.g. using LZW [Wel84], may reduce the data size just by those few bytes that are needed to squeeze in the geometry – and, thus, avoid indirections as with the other storage formats.

Note that the (simple) file system based approach works because the IDAA server process runs on the same system as the Netezza backend. Thus, both components have access to the same file system. If that were not the case, a shared file system or remote file transfer could be used.

**Evaluation** As of today, we have implemented a prototype of the \texttt{ChunkGeometry} function and use it to insert the spatial data from IDAA. The binary chunking beats FME’s [Sof14] chopping into small, consistent geometries. FME needed more than 2 hours for 23K geometries (on a very slow system), while our approach finished in 20 minutes. But note that we have not conducted any serious performance comparison yet.

4.3 Query and Result Set Processing in IDAA Context

The binary chunking comes with a penalty during query processing: whenever predicates shall be evaluated on a geometry or other spatial functions are used, it is mandatory to recompose the original geometry. This is accomplished using our new aggregate function

573
RecomposeGeometry. The aggregate gets all chunks of the geometry as input (as individual rows from the side table) and internally constructs the full geometry. The result is a geometry locator.

Recomposition is only needed as the first step in query processing. Once a chunked geometry is recomposed and its geometry locator exists, all stacked spatial functions work with that locator. Also, the locator is used when binding out the geometry from Netezza to IDAA.

IDAA receives its queries from DB2, and DB2 is not aware of the storage details for spatial data in the Netezza backend. But we want to avoid complex logic for query rewrite in DB2 itself. For each accelerated table with spatial data, we have the above mentioned side table. We use a view to hide this side table and to inject the RecomposeGeometry aggregate at the same time. The view takes the place of the actual accelerated table, so any query sent by DB2 to IDAA is not even aware that a view is accessed. Netezza resolves the view transparently into the join over both tables.

```sql
CREATE VIEW country_table(id, name, abbreviation, geom) AS
SELECT c.id, c.name, c.abbreviation,
     RecomposeGeometry(s.chunk)
FROM country AS c JOIN country_side AS s
     ON (c.id = s.country_id )
GROUP BY c.id;
```

Listing 3: View to hide side tables and recompose aggregate function

For example, a SQL query in DB2 like `SELECT ST_Buffer(geom, 5, ’Miles’) FROM country WHERE ...` needs to be rewritten to access the specific table name used in Netezza (country_table in our example), and that’s nearly all there is to it. Thus, we have achieved a very simple decoupling of DB2 from the IDAA and Netezza internal storage mechanism for the spatial data.

We have mentioned above that geometry values are bound out from Netezza to IDAA using geometry locators. Thus, the result set of the query contains a locator. Given that result sets are always consumed by IDAA, IDAA has to resolve the locator and extract the real geometry either from it directly (if inline) or by accessing the shared file system. This is the reverse procedure as for spatial data ingestion into the Netezza backend.

**Evaluation** Our prototypical implementation has reached the point where we can verify the functionality. So far, it has proven that we can implement all spatial functions available in DB2 with the correct semantics. The next steps will be a performance evaluation to measure the impact and acceleration for query processing. Due to the additional join with the side table, we do not expect to achieve the same very good acceleration factors of 10x-100x as IDAA delivers for non-spatial queries. However, our expectations are still such that spatial queries scanning over TBs of data should still run very well in our environment.
5 Summary and Outlook

In this paper we have presented a prototype implementation for the integration of spatial data support into IDAA. The main issue are limitations of the size for `VARBINARY` values in Netezza, which we resolve by chunking the binary representation of large geometries and storing those chunks in a side table, which is always joined at query time. We have proven that functional limitations of Netezza spatial can be resolved this way.

Our next steps in this project are more in-depth performance evaluations. We want to experiment with additional storage formats to exchange large geometries between IDAA and Netezza as well as passing such geometries from one spatial function to the next within the same SQL statement. Although, we always keep the geometry values stored in the Netezza tables, we will evaluate whether caching of recomposed geometries can be done in order to speed up the recomposition steps. The primary question with such a (file system based) cache will become the maintenance, i.e. which geometry objects to cache and which to discard in order to stay within the available resources like disk space.

6 Trademarks

IBM, DB2, and z/OS are trademarks of International Business Machines Corporation in USA and/or other countries. Other company, product or service names may be trademarks, or service marks of others. All trademarks are copyright of their respective owners.

References


[IBM14a] IBM. *Administering IBM Spatial Support for DB2 for z/OS*, 2014.


[Leh03] W. Lehner. Datenbanktechnologie für Data-Warehouse-Systeme – Konzepte und Meth-


[Sto06] K. Stolze. Integration of Spatial Vector Data in Enterprise Relational Database Envi-

[Sto13] K. Stolze. Seamless Integration of Archiving Functionality in OLTP/OLAP Database Systems Using Accelerator Technologies. In BTW 2013, Datenbanksysteme für Busi-