# Runtime Encoding Execution in AnalyticDB: Efficient Query Executor for Cloud Database

Qiaoyi Ding
qiaoyi.dingqy@alibaba-inc.com
Alibaba Group
Hangzhou, China

The performance of analytical processing is the key concern of cloud database customers. With the help of proper tuning on the data model according to the characteristics of the workload, it is able to maximize the performance of the Query Executor (QE). However, based on the observations from Alibaba AnalyticDB [4], a columnar OLAP database system which is designed for high-concurrency, low-latency and real-time queries, and serves a diversity of workloads for thousands of cloud customers, cloud customers prefer more flexible data types of `VARCHAR` to fixed length `VARCHAR(n)`, and `BIGINT` to store small numbers. Statistics shows that over 90% `BIGINT` values can be represented within 4 bytes. `VARCHAR` used for aggregation, join or sorting are usually enumeration or fixed-length serial numbers. Apparently, using wide data types makes the system easy to expand with the growth of their business, but it consumes more memory and involves higher computational overhead. How to achieve high query performance under such circumstances has become a new challenge to the QE.

Recent researchers have studied the *compression execution*, which directly utilizes the raw compressed data or other auxiliary data built by the storage layer to speed up scanning and filtering [1–3]. However, to further accelerate higher-level operators such as sorting, aggregation or join, it requires the compression schemes to be computational-friendly to these operators and relies on the feasible delayed decompression mechanism. To achieve those, the storage and execution engines should be redesigned to collaborate closely. For cloud databases which are possibly built over the open store storage, the opportunities are more limited.

In this paper, we propose **Runtime Encoding Execution** concept to accelerate queries by operating directly on the data encoded during the runtime in the QE layer. It decouples from the storage layer, so the limitations due to the data model or compression ability are no longer existed. and it is capable to deal with intermediate results so that higher-level operators can have a chance to be accelerated as well. The key principle is we **only apply runtime encoding when the benefit gained from the computation surpasses the coding overhead**, which requires the encoding schemes to have the following features:

- light-weighted encoding and decoding.
- fit operator's feature: The encoded data is better to be order preserved for sorting, or unique for join and aggregation.
- less memory consumption or more cache friendly.
- computation saving.
- facilitate the apply of a faster operating algorithms.

 12th Annual Conference on Innovative Data Systems Research (CIDR '22). January 9-12, 2022, Chaminade, USA.

We designed three kinds of runtime encoding schemes which conform to the principle and features. The **Compression Encoding Scheme** is used for single key compression to gain better temporal locality and conditionally to save computation by only computing identical values for once. It includes `Delta Encoding`, `Roaring Bitmaps`, `Run-length Encoding` and `Dictionary`. **Identity Encoding Scheme** and **Order Preserving Encoding Scheme** are used to serialize multiple keys into a binary string in a certain layout to alleviate the cache miss problem in multiple keys computation cases for hashing and sorting. The serialized binary can bring extra benefits by reducing computation and applying faster algorithms. For example, either the comparison or hashing function is called only once for a 8-byte encoded value instead of 4 times for the original 4 `SMALLINT` keys, and it is possible to sort using `RadixSort` on the order-preserved binary string. Note that both the schemes can apply to null-able and variable-lengthed keys. While orders can be preserved using the *Order Preserving Encoding Scheme* for complex sorting cases with a mixed of DESC, ASC, NULL FIRST and NULL LAST ordering.

According to the ratio of the keys' number computed by sorting, aggregation and join per query in a day based on the AnalyticDB's workloads. Over 90.35% of the queries compute over 1 to 3 keys, which are able to gain benefits from our designed schemes. The results of simple experiments on the TPCH benchmark are listed in Table 1. It shows 65.42% and 51.52% improvements for the aggregation on variable length string and multiple grouping keys separately. While for the sorting queries, 61.37% and 82.83% improvements are produced because RadixSort is applied after the encoding, and a 50.18% enhancement is attained for the prefix based sorting in the third case due to fewer fallbacks when comparing the same prefix.

**Table 1: Experiments on TPCH benchmark**

| Key(s) | Description | Orig. | Opt. |
|---|---|---|---|
| E2E improvement of aggregation quires on TPCH1T | | | |
| l_shipstruct | 4~11 bytes `VARCHAR` | 10.7s | 3.7s |
| l_linenumber, l_shipdate | `INTEGER` (small), `date` | 16.5s | 8.0s |
| CPU wall time improvement of sorting queries on TPCH10G | | | |
| l_shipmode | 3~7 bytes `VARCHAR` | 32.1s | 12.4s |
| l_returnflag, l_suppkey | 1 byte `VARCHAR`, `INTEGER` | 2.98m | 30.7s |
| l_returnflag, l_comment | 1 byte, 10~43 bytes `VARCHAR` | 5.32m | 2.65m |

## REFERENCES

[1] 2021. Apache HBase. https://hbase.apache.org.
[2] 2021. Apache Hudi. https://hudi.apache.org.
[3] 2021. Apache Iceberg. https://iceberg.apache.org.
[4] Feifei Li. 2019. Cloud-native database systems at Alibaba: Opportunities and challenges. *Proceedings of the VLDB Endowment* 12, 12 (2019), 2263–2272.