**Datawarehousing Options**

Open Source Datawarehouse services

Apache Doris

PostgreSQL Datawarehouse

Apache Hive

Apache Kylin

Cloud Datawarehouse Services

Azure Synapse Analytics

AWS Redshift

Microsoft Fabric

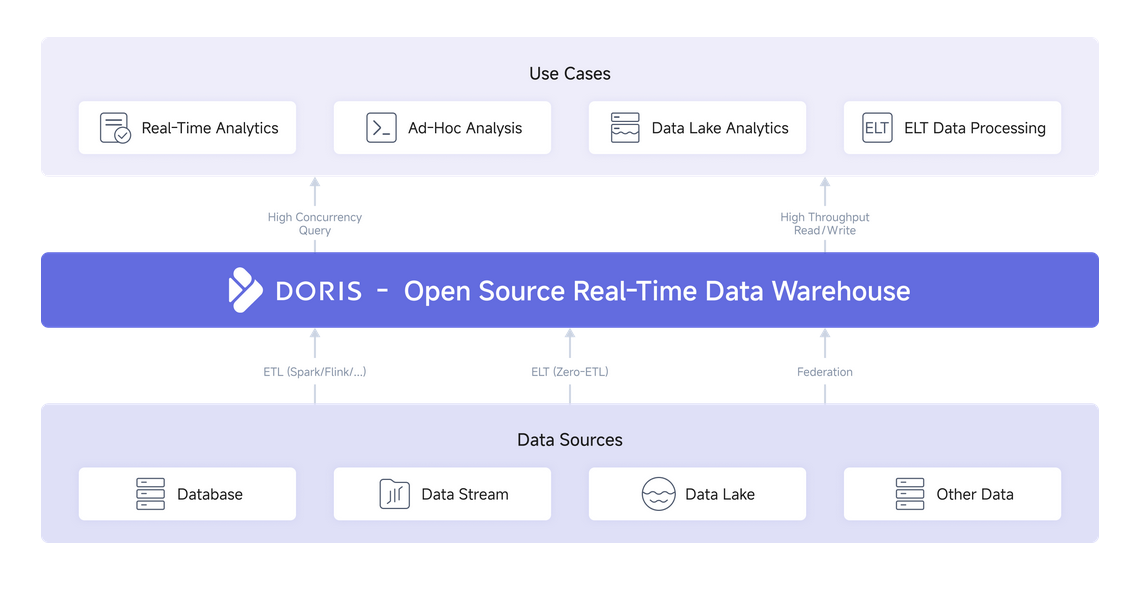
**Apache Doris**

Apache Doris is an MPP-based real-time data warehouse known for its high query speed. For queries on large datasets, it returns results in sub-seconds. It supports both high-concurrent point queries and high-throughput complex analysis. It can be used for report analysis, ad-hoc queries, unified data warehouse, and data lake query acceleration. Based on Apache Doris, users can build applications for user behavior analysis, A/B testing platform, log analysis, user profile analysis, and e-commerce order analysis.

Apache Doris, formerly known as Palo, was initially created to support Baidu's ad reporting business. It was officially open-sourced in 2017 and donated by Baidu to the Apache Software Foundation in July 2018, where it was operated by members of the incubator project management committee under the guidance of Apache mentors. In June 2022, Apache Doris graduated from the Apache incubator as a Top-Level Project. By 2024, the Apache Doris community has gathered more than 600 contributors from hundreds of companies in different industries, with over 120 monthly active contributors.

Apache Doris has a wide user base. It has been used in production environments of over 4000 companies worldwide, including giants such as TikTok, Baidu, Cisco, Tencent, and NetEase. It is also widely used across industries from finance, retailing, and telecommunications to energy, manufacturing, medical care, etc.

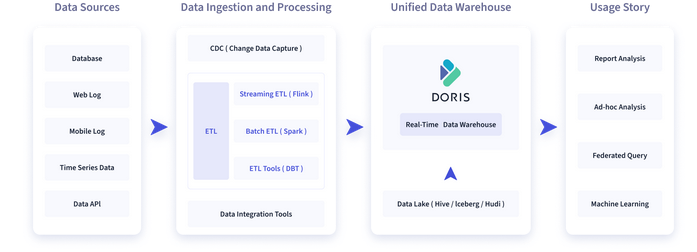
**High Level Architecture**

****

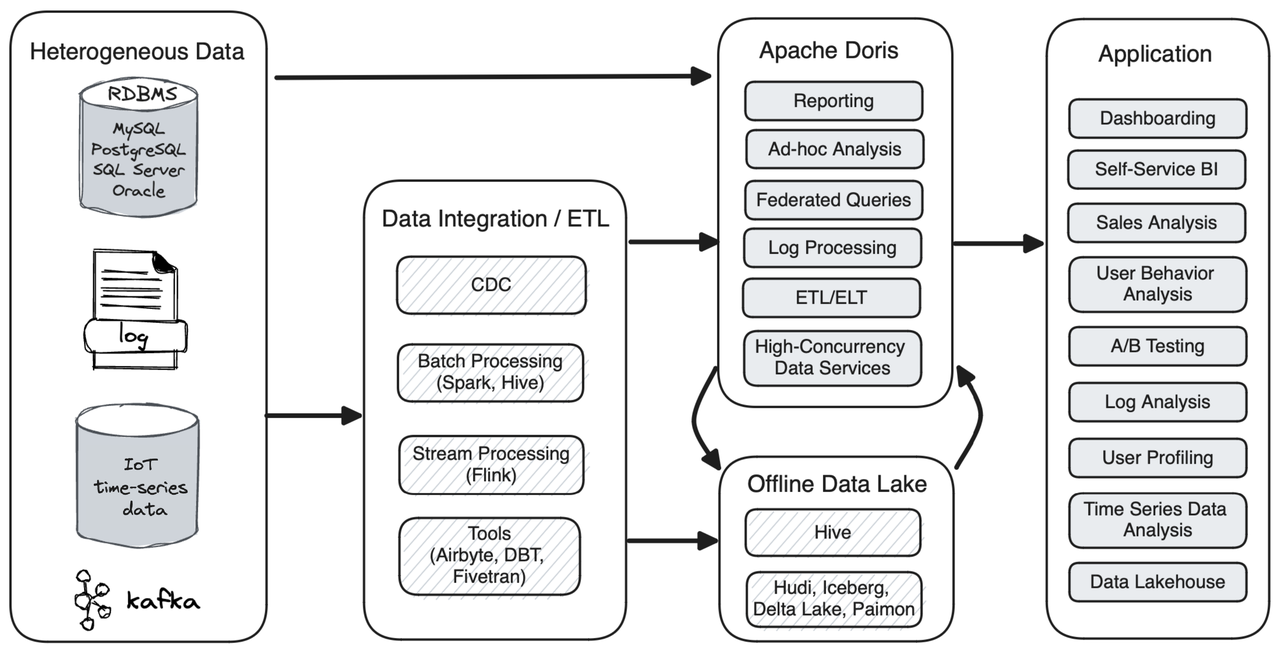
**Capabilities**

* Real time Data Ingestion
* MPP architecture – fast query results
* Data Lake and Database support
* Supports Semi- Structured data
* Open Source
* Real-time dashboards
* Push-based micro-batch and pull-based streaming data ingestion within a second. Storage engine with real-time upsert, append and pre-aggregation.
* Lightning-fast query
* Optimize for high-concurrency and high-throughput queries with columnar storage engine, MPP architecture, cost based query optimizer, vectorized execution engine.Unified Data Warehouse Construction
* Federated querying
* Federated querying of data lakes such as Hive, Iceberg and Hudi, and databases such as MySQL and PostgreSQL.
* Semi-structured data
* Compound data types such as Array, Map and JSON. Variant data type to support auto data type inference of JSON data. NGram bloomfilter and inverted index for text searches.
* Elastic architecture
* Distributed design for linear scalability. Workload isolation and tiered storage for efficient resource management. Supports shared-nothing clusters as well as separation of storage and compute.
* Data Lake Query
* Doris as a high-performance federated query engine provides a powerful way to directly map external data lakes and databases to Doris' databases and tables. This combination of the openness of data lakes and the high performance of data warehouses offers enterprises a unified and efficient way to access, analyze, and manage their data

**Usage Scenario**



The figure below shows what Apache Doris can do in a data pipeline. Data sources, after integration and processing, are ingested into the Apache Doris real-time data warehouse and offline data lakehouses such as Hive, Iceberg, and Hudi. Apache Doris can be used for the following purposes:



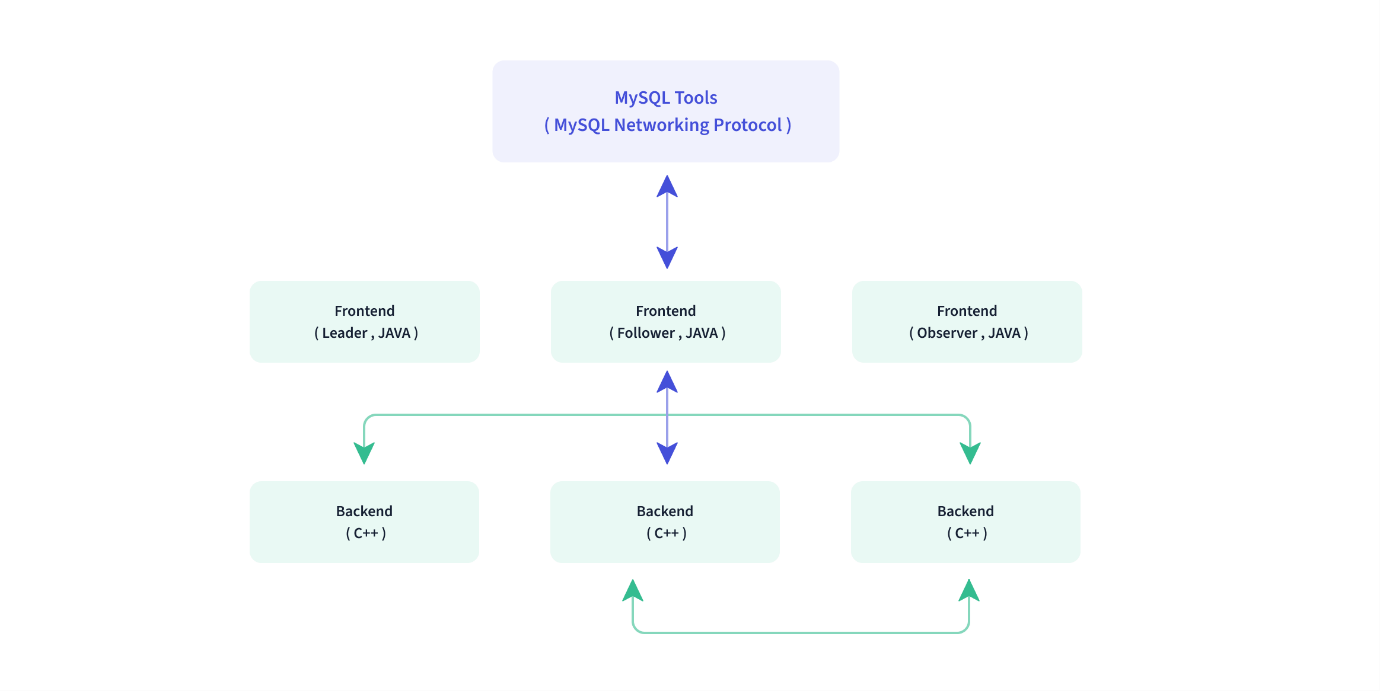
* **Report analysis**
  + Real-time dashboards
  + Reports for internal analysts and managers
  + Customer-facing reports: such as site analysis for website owners and advertising reports for advertisers. Such cases typically require high concurrency (thousands of QPS) and low query latency (measured in milliseconds). For example, the e-commerce giant JD.com uses Apache Doris for ad reporting. It ingests 10 billion rows of data per day and achieves over 10,000 QPS and P99 latency of 150ms.
* **Ad-hoc query**: analyst-facing self-service analytics with irregular query patterns and high throughput requirements. For example, Xiaomi builds a Growth Analytics platform based on Doris. Handling 10,000s of SQL queries every day, it delivers an average query latency of 10 seconds and a P95 latency of 30 seconds.
* **Data lake analytics**: Apache Doris allows federated queries on external tables in offline data lakehouses such as Hive, Hudi, and Iceberg and achieves outstanding query performance by avoiding data copying.
* **Log analysis**: Apache Doris supports inverted index and full-text search since version 2.0. Relying on its highly efficient query and storage engines, Apache Doris enables 10 times higher cost-effectiveness than common log analytic solutions.
* **Unified data warehouse**: Apache Doris can work as a unified data processing platform for various analytic workloads, saving users from handling complicated data components and tech stacks. For example, Haidilao, a world-renowned chain restaurant, replaces its old architecture consisting of Spark, Hive, Kudu, HBase, and Phoenix with Apache Doris.

## Technical overview

Apache Doris has a simple and neat architecture with only two types of processes.

* **Frontend (FE)**: user request processing, query parsing and planning, metadata management, and node management
* **Backend (BE)**: data storage and query execution

Both frontend and backend processes are scalable, supporting up to hundreds of machines and tens of petabytes of storage capacity in a single cluster. Both types of processes guarantee high service availability and high data reliability through consistency protocols. This highly integrated architecture design greatly reduces the operation and maintenance costs of a distributed system.



## Interface

Apache Doris adopts the MySQL protocol, supports standard SQL, and is highly compatible with MySQL syntax. Users can access Doris through various client tools and seamlessly integrate it with BI tools, including but not limited to SmartBI, DataEase, FineBI, Tableau, Power BI, and SuperSet. It can work as the data source for any BI tools that support the MySQL protocol.

## Storage engine

Apache Doris has a columnar **storage engine**, which encodes, compresses, and reads data by column. This enables a very high data compression ratio and largely reduces unnecessary data scanning, thus making more efficient use of IO and CPU resources.

Doris supports various **index** structures to minimize data scans:

* Sorted Compound Key Index: Users can specify three columns at most to form a compound sort key. This can effectively prune data to better support highly concurrent reporting scenarios.
* MIN/MAX Indexing: This enables effective data filtering in equivalence and range queries of numeric types.
* Bloom Filter: This is very effective in equivalence filtering and pruning of high-cardinality columns.
* Invert Index: This enables fast searching for any field.

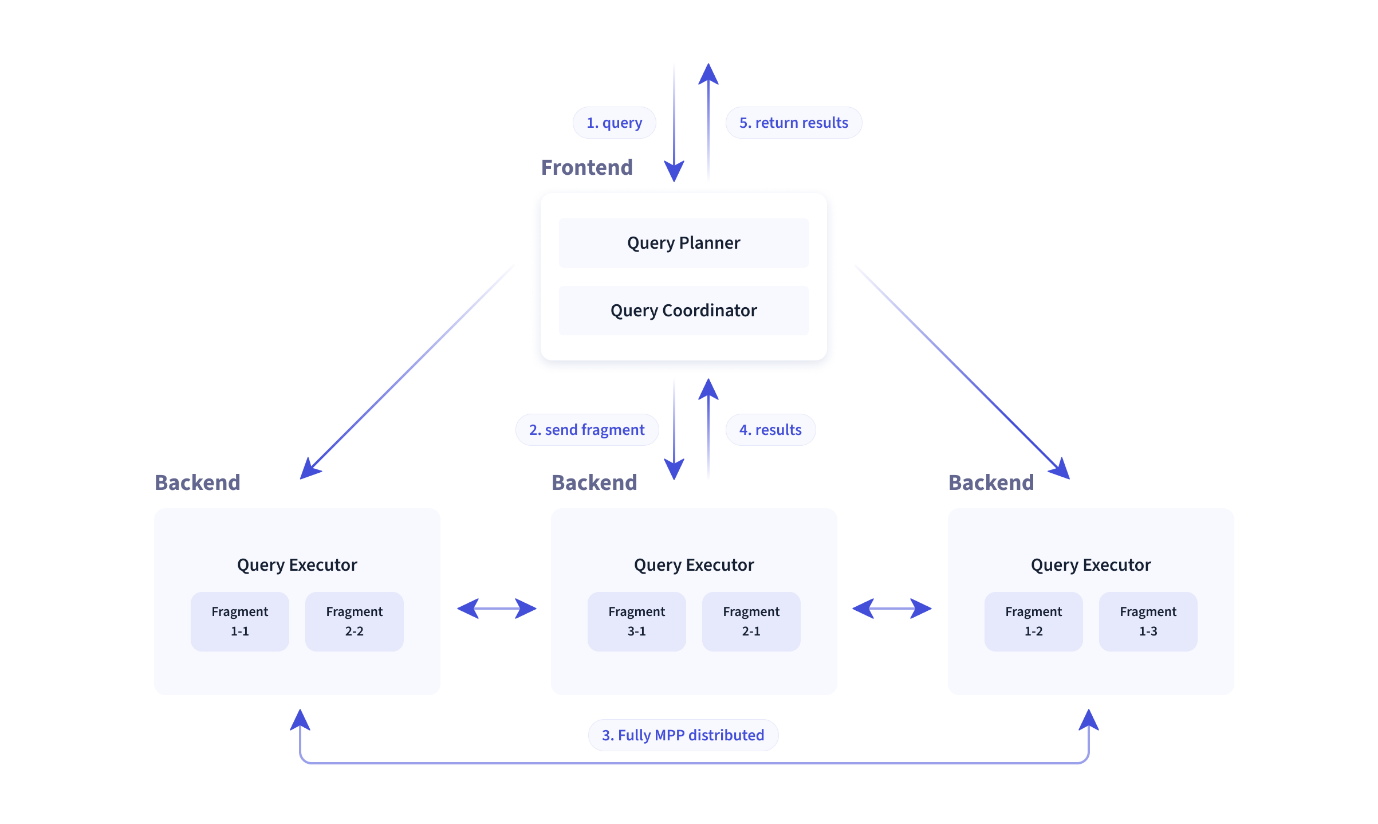
Doris supports a variety of **data models** and has optimized them for different scenarios:

* Aggregate Key Model: merges the value columns with the same keys and improves performance by pre-aggregation
* Unique Key Model: ensures uniqueness of keys and overwrites data with the same key to achieve row-level data updates
* Duplicate Key Model: stores data as it is without aggregation, capable of detailed storage of fact tables

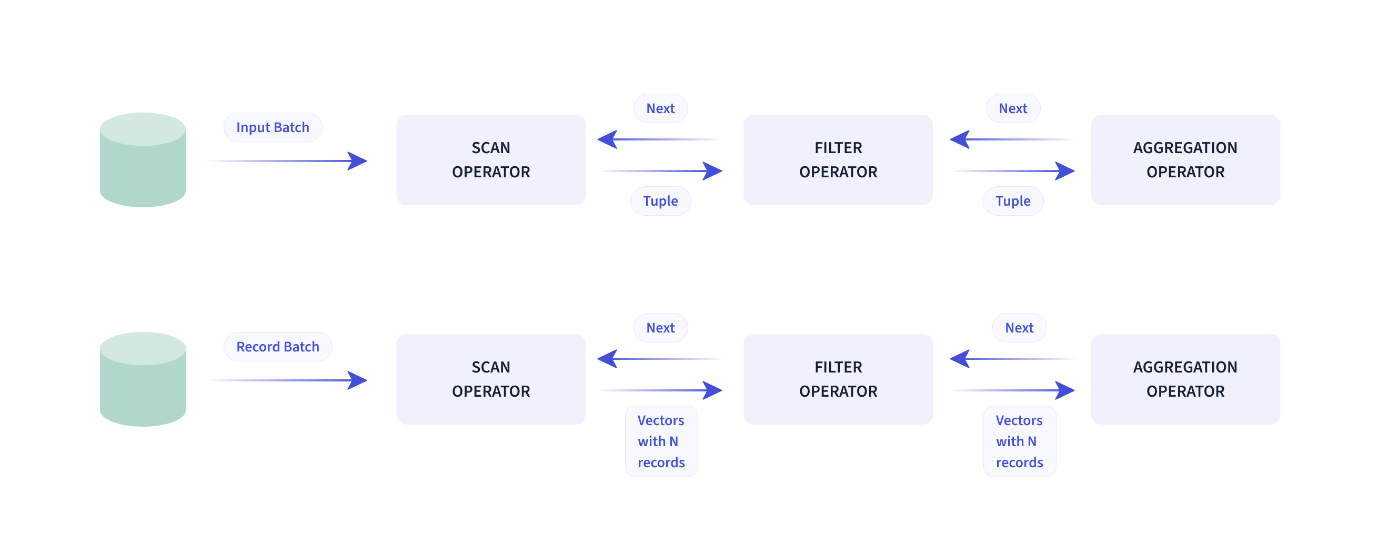
Doris also supports **strongly consistent materialized views**. Materialized views are automatically selected and updated within the system without manual efforts, which reduces maintenance costs for users.

## Query engine

Doris has an MPP-based query engine for parallel execution between and within nodes. It supports distributed shuffle join for large tables to better handle complicated queries.



The Doris query engine is fully vectorized, with all memory structures laid out in a columnar format. This can largely reduce virtual function calls, increase cache hit rates, and make efficient use of SIMD instructions. Doris delivers a 5~10 times higher performance in wide table aggregation scenarios than non-vectorized engines.



Doris uses **adaptive query execution** technology to dynamically adjust the execution plan based on runtime statistics. For example, it can generate a runtime filter and push it to the probe side. Specifically, it pushes the filters to the lowest-level scan node on the probe side, which largely reduces the data amount to be processed and increases join performance. The Doris runtime filter supports In/Min/Max/Bloom Filter.

The Doris query **optimizer** is a combination of CBO and RBO. RBO supports constant folding, subquery rewriting, and predicate pushdown while CBO supports join reorder. The Doris CBO is under continuous optimization for more accurate statistics collection and inference as well as a more accurate cost model.

**Data Type**

Apache Doris support standard SQL syntax, using MySQL Network Connection Protocol, highly compatible with MySQL syntax protocol. Therefore, in terms of data type support, Apache Doris aligns as closely as possible with MySQL-related data types

The list of data types supported by Doris is as follows:

| **Type name** | **Number of bytes** | **Description** |
| --- | --- | --- |
| BOOLEAN | 1 | Boolean data type hat stores only two types of values , 0 represents false, 1 represents true. |
| TINYINT | 1 | Integer value, signed range is from -128 to 127. |
| SMALLINT | 2 | Integer value, signed range is from -32768, 32767. |
| INT | 4 | Integer value, signed range from -2147483648 to 2147483647. |
| BIGINT | 8 | Integer value, signed range from -9223372036854775808 to 9223372036854775807. |
| LARGEINT | 16 | Integer value, range [-2 ^ 127 + 1~ 2 ^ 127 - 1]. |
| FLOAT | 4 | Single precision, a floating ponit number, range [-3.4 *10 ^ 38~ 3.4* 10 ^ 38]. |
| DOUBLE | 8 | Double precision, a floating ponit number, range [-1.79 *10 ^ 308~ 1.79* 10 ^ 308] |
| DECIMAL | 2004/8/16 | An exact fixed-point number, defined by its precision (total number of digits) and scale (number of digits to the right of the decimal point). Format:DECIMAL(M[,D]), M stands for precision, D stands scale. The range for the significant digits M is [1, 38], while the range for the decimal digits D is [0, precision]. The storage requirements for different precision values are as follows: - When 0 < precision <= 9, it occupies 4 bytes. - When 9 < precision <= 18, it occupies 8 bytes. - When 16 < precision <=38, it occupies 16 bytes |
| DATE | 16 | DATE holds values for a calendar year, month and day, the supported range is ['0000-01-01', '9999-12-31']. Default print format: 'yyyy-MM-dd'. |
| DATETIME | 16 | A DATE and TIME combination Format: DATETIME ([P]). The optional parameter P represents time precision, with a value range of [0,6], supporting up to 6 decimal places (microseconds). When not set, it is 0. The supported range is ['0000-01-01 00:00:00 [.000000]', '9999-12-31 23:59:59 [.999999]']. Default print format: 'yyy-MM-dd HH: mm: ss. SSSSSS '. |
| CHAR | M | A FIXED length string, the parameter M specifies the column length in characters. The range of M is from 1 to 255. |
| VARCHAR | M | A VARIABLE length string , the parameter M specifies the maximum string length in characters. The range of M is from 1 to 65533. The variable-length string is stored in UTF-8 encoding. English characters occupy 1 byte, and Chinese characters occupy 3 bytes. |
| STRING | / | A VARIABLE length string, default supports 1048576 bytes (1 MB), and a limit of maximum precision of 2147483643 bytes (2 GB). Size can be configured string\_type\_length\_soft\_limit\_bytes adjusted through BE. String type can only be used in value column, not in key column and partition bucket column. |
| HLL | / | HLL stands for HyperLogLog, is a fuzzy deduplication. It performs better than Count Distinct when dealing with large datasets. The error rate of HLL is typically around 1%, and sometimes it can reach 2%. HLL cannot be used as a key column, and the aggregation type is HLL\_UNION when creating a table. Users do not need to specify the length or default value as it is internally controlled based on the aggregation level of the data. HLL columns can only be queried or used through the companion functions such as hll\_union\_agg, hll\_raw\_agg, hll\_cardinality, and hll\_hash. |
| BITMAP | / | BITMAP type can be used in Aggregate tables or Unique tables. - When used in a Unique table, BITMAP must be employed as non-key columns. - When used in an Aggregate table, BITMAP must also serve as non-key columns, and the aggregation type must be set to BITMAP\_UNION during table creation. Users do not need to specify the length or default value as it is internally controlled based on the aggregation level of the data. BITMAP columns can only be queried or used through the companion functions such as bitmap\_union\_count, bitmap\_union, bitmap\_hash, and bitmap\_hash64. |
| QUANTILE\_STATE | / | A type used to calculate approximate quantile values. When loading, it performs pre-aggregation for the same keys with different values. When the number of values does not exceed 2048, it records all data in detail. When the number of values is greater than 2048, it employs the TDigest algorithm to aggregate (cluster) the data and store the centroid points after clustering. QUANTILE\_STATE cannot be used as a key column and should be paired with the aggregation type QUANTILE\_UNION when creating a table. Users do not need to specify the length or default value as it is internally controlled based on the aggregation level of the data. QUANTILE\_STATE columns can only be queried or used through the companion functions such as QUANTILE\_PERCENT, QUANTILE\_UNION, and TO\_QUANTILE\_STATE. |
| ARRAY | / | Arrays composed of elements of type T cannot be used as key columns. Currently supported for use in tables with Duplicate and Unique models. |
| MAP | / | Maps consisting of elements of type K and V, cannot be used as Key columns. These maps are currently supported in tables using the Duplicate and Unique models. |
| STRUCT | / | A structure composed of multiple Fields can also be understood as a collection of multiple columns. It cannot be used as a Key. Currently, STRUCT can only be used in tables of Duplicate models. The name and number of Fields in a Struct are fixed and are always Nullable. |
| JSON | / | Binary JSON type, stored in binary JSON format, access internal JSON fields through JSON function. Supported up to 1048576 bytes (1MB) by default, and can be adjusted to a maximum of 2147483643 bytes (2GB). This limit can be modified through the BE configuration parameter 'jsonb\_type\_length\_soft\_limit\_bytes'. |
| AGG\_STATE | / | Aggregate function can only be used with state/merge/union function combiners. AGG\_STATE cannot be used as a key column. When creating a table, the signature of the aggregate function needs to be declared alongside. Users do not need to specify the length or default value. The actual data storage size depends on the function's implementation. |

You can also view all the data types supported by Doris with the SHOW DATA TYPES; statement

**Data Models**

The data models in Doris are primarily classified into three types:

* Duplicate: This data model allows for storing duplicate rows based on the specified key columns. It is suitable for scenarios where preserving all the original data records is essential.
* Unique: In this data model, each row is uniquely identified by the combination of values in the key columns. This ensures that no duplicate rows exist for a given set of key values.
* Aggregate: This model enables the aggregation of data based on the key columns. It is commonly used for scenarios where summary or aggregated information, such as totals or averages, is required.

References:

https://doris.apache.org/