TDSQL Boundless

Product Introduction

Product Documentation

Contents

Product Introduction

Overview

Last updated：2026-02-10 11:05:58

TDSQL Boundless is a new generation distributed database developed in-house by Tencent Cloud. As a key member of the TDSQL family, it effectively solves the complexity challenges of traditional distributed databases through its transparent distributed architecture and elastic scalability. This enables your enterprises to focus on business innovation instead of wrestling with the complex data architectures, while providing enterprise-grade high availability and security, delivering reliable support for critical business systems.

Key Features

Multi-Master Writes & Seamless Scaling

TDSQL Boundless features a multi-master architecture where every node can handle write operations, dramatically increasing total throughput. The platform scales linearly and transparently with zero downtime, growing smoothly from a minimal two-node instance to several hundred nodes. Whether handling peak order volumes during an e-commerce sale or high-concurrency interactions in a live stream, TDSQL Boundless manages the load effortlessly. At the same time, its storage engine delivers high data compression, enabling significant cost savings for massive datasets.

Higher Resource Utilization，Lower Costs

TDSQL employs a peer-to-peer architecture where all nodes are active. This eliminates the resource waste common in traditional MySQL primary-replica setups, where secondary nodes often sit idle. Its storage engine, built on an LSM-Tree architecture with intelligent compression, achieves a far higher data compression ratio than traditional engines, leading to a significant reduction in storage costs.

Zero Data Loss & Flexible Disaster Recovery

TDSQL Boundless delivers enterprise-grade reliability. Through multi-replica disaster recovery and automatic failover mechanisms, it ensures zero data loss (RPO=0) and high service availability. The system's built-in resilience allows it to recover from node failures quickly and automatically, with no manual intervention required. Furthermore, it offers flexible deployment models-including 1 AZ, 2 AZ, and 3 AZ setups-to meet your specific disaster recovery requirements.

High Compatibility with MySQL & HBase

TDSQL Boundless is highly compatible with most general features of MySQL, including protocol, syntax, and its broader ecosystem of tools. It also offers compatibility with the HBase API. This means existing MySQL applications can migrate to TDSQL Boundless without code modifications while significantly reducing migration costs and risks.

Automated Data Scheduling for Access Locality

TDSQL Boundless employs advanced "data affinity scheduling" technology that intelligently arranges primary keys, indexes, and related table data on the same node. This transforms complex distributed transactions into simple, single-node operations, not only enhancing performance but also ensuring data consistency.

Scenarios

Last updated：2026-02-10 11:05:58

Historical Data Archiving Scenarios

In industries such as e-commerce, retail, finance, O2O, and social applications, vast amounts of historical data accumulate alongside the latest data in MySQL databases over time. This hybrid storage of cold data and hot data leads to challenges like high storage costs, degraded database performance, and backup difficulties. Leveraging TDSQL Boundless' capabilities—massive storage, low cost, high reliability, and strong scalability—users can implement large-capacity data archival solutions at minimal cost while still supporting transactional processing and online data access. Tencent services like telecom top-ups and credit card repayment records already utilize TDSQL Boundless.

Massive Datasets Scenarios

Traditional databases face numerous challenges when storing and managing massive data, especially in high-concurrency read/write and high-throughput scenarios. In massive-scale scenarios like log streams, IoT, and smart homes, large data volumes, extensive single-table records, high write throughput, and stringent consistency requirements are common business demands. TDSQL Boundless offers robust horizontal scalability, enabling near-linear improvements in system read/write throughput and processing capabilities when nodes are added, supporting hundreds of terabytes or more of data processing. Concurrently, its engine incorporates the Multi-Raft protocol to deliver strong consistency and high availability, along with rapid online DDL and online capabilities, allowing users to stably and efficiently store and manage massive data. Additionally, TDSQL Boundless provides efficient data compression and storage policies, effectively reducing storage costs.

Agile Business Scenarios

With modern market demands becoming increasingly volatile and business development more agile, flexible Schema and unrestricted online scaling capabilities are particularly crucial for users in industries like gaming and SaaS. TDSQL Boundless provides robust support for agile business needs, with its scalability enabling rapid resource scaling up or down as required. It supports horizontal scaling, vertical scaling, and replica scaling. Concurrently, TDSQL Boundless' rapid online DDL capabilities allow users to effortlessly adjust database structures without disrupting business operations, effectively addressing unpredictable business changes and delivering strong support for complex modern requirements.

Centrialized Data Hub Scenarios

While traditional MySQL solutions using core business shard keys for database and table partitioning address transactional processing (TP) scenarios, they prove unfriendly for data operations or data aggregation scenarios (Data Serving/Data Hub). These scenarios often require lightweight statistical analysis across multiple dimensions rather than queries based solely on shard keys. In such cases, neither broadcast queries nor index tables can achieve multi-dimensional queries simply, flexibly, and efficiently. TDSQL Boundless excels in these scenarios. It supports high-speed aggregation of data from multiple shards into a single instance, enables creation of globally unique indexes, and ensures consistent queries across multi-shard, multi-dimensional data. This meets the high-performance read/write demands of data middle platform scenarios while supporting users in conducting data operations and analysis on massive datasets.

Product Architecture

Last updated：2026-02-10 11:05:58

TDSQL Boundless instances are divided into two kinds: cluster edition and basic edition.

- **Cluster edition**: Composed of multiple (≥3) nodes, it provides high-performance and available database services in the form of a triple-replica Raft cluster, suitable for enterprise production environments.
- **Basic edition**: Composed of a single node, it provides complete database features at a relatively low cost without high availability, suitable for individual users.

> **Note:**
> After a basic edition instance is created, it can be upgraded to a cluster edition instance through the console. Once a cluster edition instance is created, it cannot be downgraded to a basic edition instance.

The nodes in a TDSQL Boundless instance are divided into two types: peer-to-peer architecture and separated computing and storage architecture.

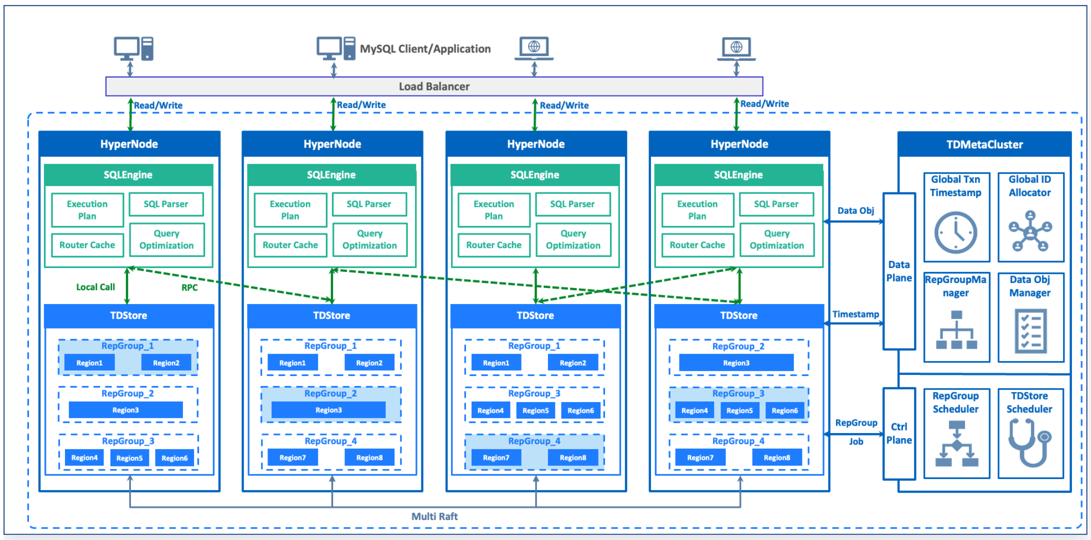
- **Peer-to-peer architecture**: The computing layer SQLEngine and data layer TDStore are integrated into a single physical node, reducing the number of hardware nodes and cross-node communication and thereby lowering costs and improving performance.
- **Separation architecture**: The computing layer SQL engine and the data layer TDStore are located in different physical nodes.

Technical Architecture of TDStore

The feature modules of centralized stand-alone databases and distributed databases can be divided into three components:

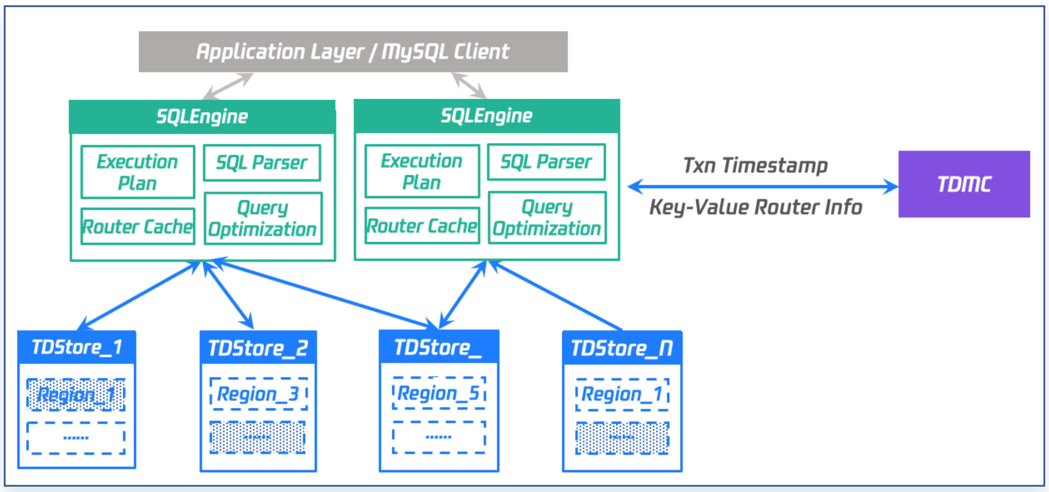
- Computing engine: Mainly includes SQL parsing, optimizers, and executors.
- Storage engine: Mainly includes transaction processing and data storage.
- Metadata service: Mainly includes global logical clock services, global ID generators, metadata storage, scheduling engines (data/disaster recovery scheduling), and load collection.

The overall architecture of TDSQL Boundless is as shown below:



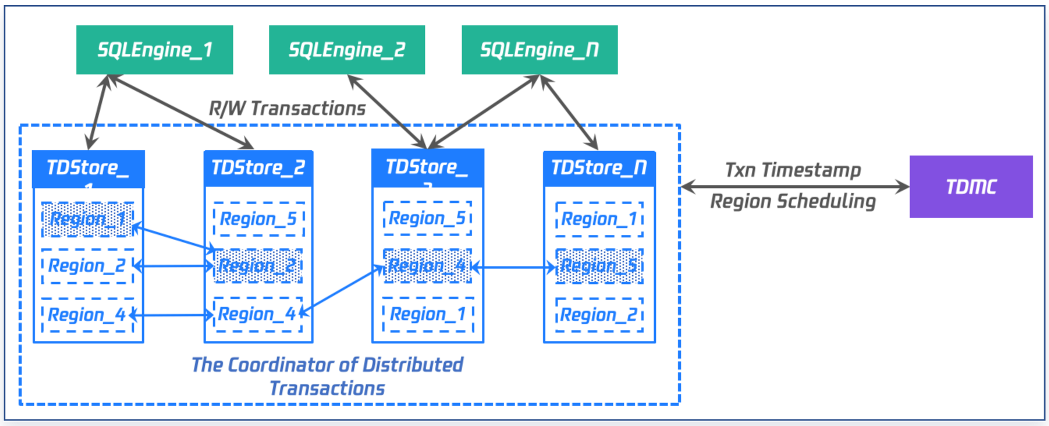
Computing Engine - SQL Engine

- Kernel: It is implemented based on MySQL 8.0 and is highly compatible with MySQL.
- Architecture: The computing layer adopts a multi-leader architecture with a stateless design. Each SQL engine node can be read or written.
- Interaction: Obtain the global transaction timestamp and data routing information from the control node, interact with the storage node for transactions, and return results to the client.



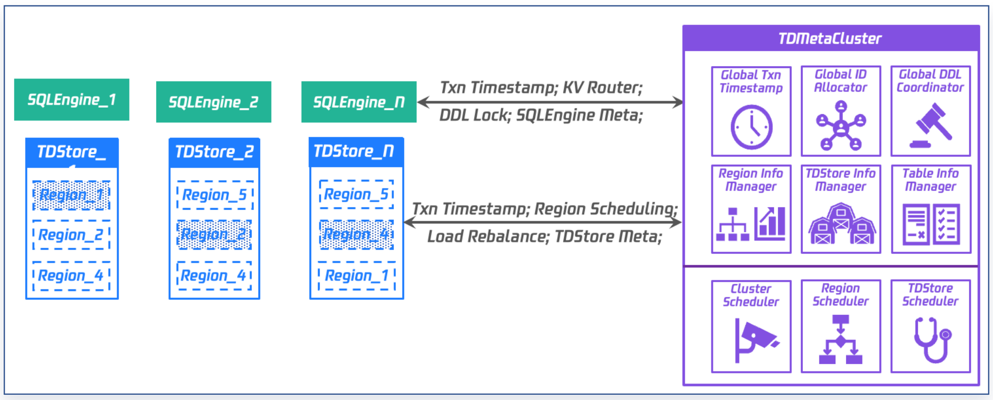
Storage Engine - TDStore

- Architecture: Distributed KV storage engine based on LSM-Tree and Multi-Raft.
- Data: Multi-replica storage based on Raft synchronization, with data distributed across different Regions according to Key ranges. Multiple Regions jointly compose a replication group, performing data replication and high-availability switch with the aid of the Raft consensus protocol.
- Interaction: TDStore receives requests from compute nodes, processes them, and returns results. The primary replica of each region is responsible for receiving and handling read-write requests.



Metadata Service – TDMC

- Architecture: Raft-based metadata management cluster with one leader and two followers, with services provided by the leader.
- Data:
  - Assign a globally unique and incremental transaction ID.
  - Manage TDStore and SQLEngine metadata.
  - Manage region data routing information.
  - Manage global MDL locks.
- Control:
  - Schedule the split, merge, migration, and master switch of a replication group.
  - Schedule the scaling of the storage layer.
  - Schedule the load balancing of the storage layer.
  - Issue alarms for abnormal events in various dimensions.



Storage Model

TDSQL Boundless's storage engine will map all users' data to a linear ordered infinite Key space (−∞, +∞), with each line of data corresponding to a certain point in the Key space.

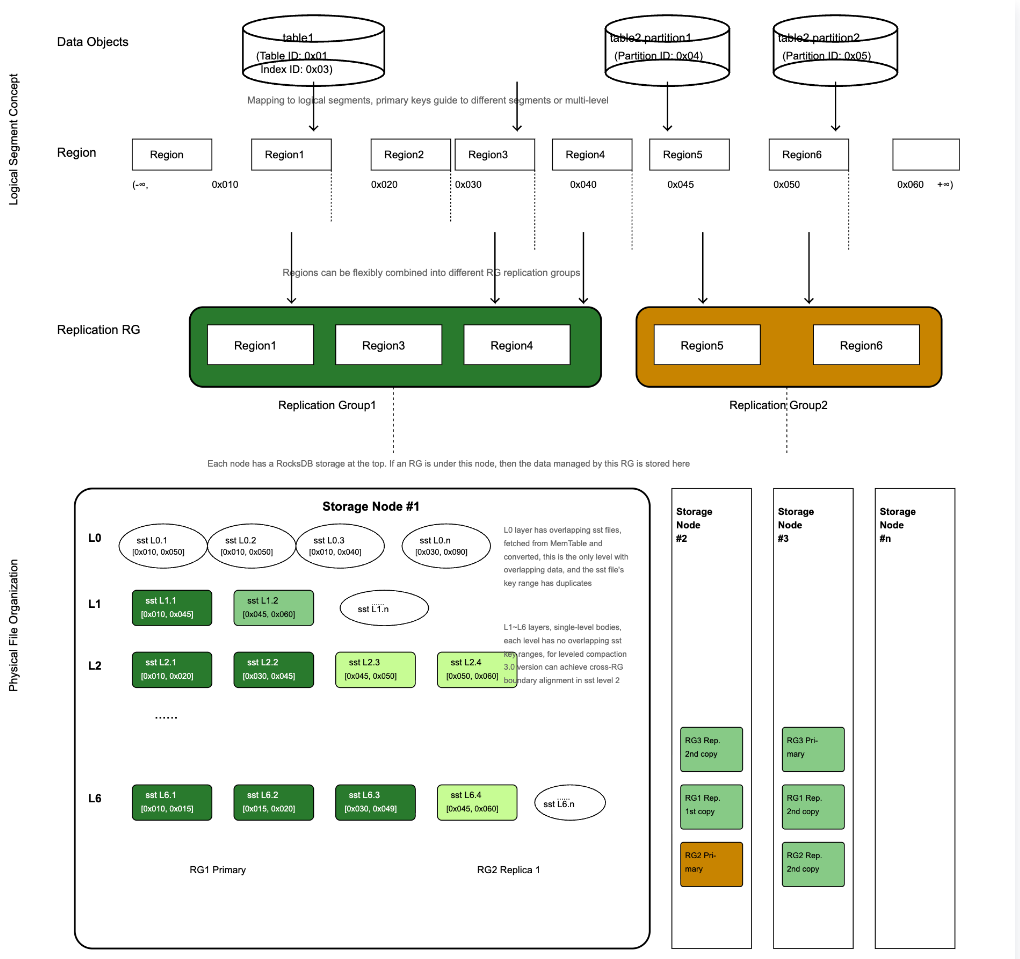
```
//Example
```

```
create table t1 (
  f1 varchar(50),
  f2 varchar(20),
  f3 varchar(20),
  f4 varchar(20),
  primary key(f1,f2),
  index idx_f3(f3)
);
insert into t1 values('a', 'b', 'c', 'd');
```

In the example above, inserting a row of data will occur two key-value pairs:
Primary key: pk_encode('t1', 'a', 'b') -> pk_value_encode('c', 'd').
Index: sk_encode('idx_f3', 'c', 'a', 'b') -> sk_value_encode().
Since the frontmost part of each key is the ID of the database object (table/index/partition table), all data of this database object is continuously distributed in a segment of the key space.

Three-Layer Organizational Mode (DO-Region-RG)

Database object: For example, table, partition, and index. Usually, a globally unique ID (index_id) is assigned for these database objects.
Data shard (region):
①: A consecutive, left-closed and right-open key space (startkey, endkey).
②: A database object corresponds to one or more regions. During operation, a region can be split or multiple regions can be merged into one based on data scheduling policies.
③: Basic unit of data scheduling. Different regions can be distributed on any node in the instance.
④: The region is a logical concept. In the underlying storage, it is not a completely independent segment or file.
Replication group (RG):
①: A replication group can contain one or more regions, and these regions belong to one or more data objects.
②: During operation, RGs can be merged or split (by creating RGs and migrating regions into or out of RGs).
③: A single RG corresponds to a Raft log stream (redo log/WAL). If all keys involved in a transaction are within one RG, the transaction can be converted to a non-distributed transaction. If a transaction involves data across RGs, a two-phase distributed transaction protocol is required.



Peer-to-Peer Architecture Node

In the architecture design of TDSQL Boundless, we aim to keep compute close to storage (cache/storage) while achieving compute-storage separation for high elasticity. Therefore, TDSQL Boundless is designed as follows:

- The peer-to-peer architecture (HyperNode) design is adopted. Each peer node (process) contains three complete feature engines: computing, storage, and logging.
- The computing layer and local storage use the local access mode, while accessing remote storage uses the network RPC access mode to ensure the performance of accessing local data as much as possible.
- Based on the needs of different business scenarios, roles can also be assigned for nodes through metadata and the scheduling module, for example, full-featured nodes (which support all three services: computing, storage, and logging), compute nodes (which only support computing, with all data accessed remotely), storage nodes (which only support data storage), and logging services (which only provide log subscription services).

> **Note:**
> The features designated for node roles are not yet available.

Instance Types

Last updated：2026-02-10 11:05:58

This article introduces the instance types of TDSQL Boundless, including Cluster Edition and Basic Edition.

Instance Type	Description
Cluster Edition	Cluster Edition targets enterprise production environments, adopting a multi-node and multi-replica disaster recovery architecture to provide high-performance, highly available database services.
Basic Edition	Basic Edition targets individual users with a single-node database service, providing a full range of features at a lower cost, suitable for learning and trial scenarios. **Note:** Basic Edition instances support upgrading to Cluster Edition via the console after creation, but they cannot be downgraded back to Basic Edition after the upgrade. **Warning:** Basic Edition does not have high availability capabilities and may experience brief service interruptions. Enterprises should not use it for production environments.

Compatibility Notes
Compatibility with MySQL
MySQL Compatibility

Last updated：2026-02-10 11:05:59

TDSQL Boundless is highly compatible with the MySQL 8.0 protocol. This allows you to use your existing ecosystem of MySQL tools, such as phpMyAdmin, Navicat, MySQL Workbench, and DBeaver, without modification.

Unsupported Features

- DDL operations on the `mysql` system database
- Generated Columns
- Events
- Tablespaces (icluding Transportable Tablespace)
- Resource Groups
- Foreign Keys
- Partial updates of LOB data types
- Multi-valued indexes on JSON data type
- Spatial functions, data types, and indexes （GEOMETRY/GIS)
- Full-text Indexes and syntax
- Descending Indexes
- Savepoint
- REPAIR TABLE syntax
- Exchanging partitions (ALTER TABLE ... EXCHANGE PARTITION)
- SKIP LOCKED syntax
- XA-related statements return success but do not take effect.
- Group Replication
- X Protocol

Differences in Behavior from MySQL

Auto-increment Columns

TDSQL Boundless offers two modes for auto-incrementing IDs:

- **High-Performance Mode (Default):** This mode improves performance by caching a batch of auto-increment values on each node. It guarantees that the auto-increment values are globally unique, but not that they are strictly sequential across the instance. The batch size of the cache is controlled by the `tdsql_auto_increment_batch_size`, with a default value of 100.
- **MySQL Compatibility Mode:** By setting `tdsql_auto_increment_batch_size = 1`, you can ensure that auto-increment values are both globally unique and strictly sequential.

> **Warning:**
> Risk of Duplicate Values: To prevent duplicate AUTO_INCREMENT values, the auto-increment column must be defined as a standalone primary key or unique index. If it is only part of a composite primary key, manually inserted IDs can conflict with system-generated IDs.

Unlike MySQL, the auto-increment column in TDSQL Boundless may contain duplicate values under specific circumstances. When the auto-increment column is not a primary key or unique index (such as when it is only part of a composite primary key), TDSQL Boundless cannot guarantee the uniqueness of its values. This issue occurs when manually inserted auto-increment values conflict with system-generated values. For example, when a test table `t1` is created where the auto-increment column `id` forms a composite primary key with `f1`:

```
CREATE TABLE `t1` (
  `id` int NOT NULL AUTO_INCREMENT,
  `f1` int NOT NULL,
  `f2` int DEFAULT NULL,
  PRIMARY KEY (`id`,`f1`)
) ENGINE=ROCKSDB AUTO_INCREMENT=3 DEFAULT CHARSET=utf8mb3
```

At this point, perform the operation to write data on node 1, automatically generating data with `id = 1`:

```
-- Using system-generated auto-increment.
insert into t1(f1, f2) values(1,1);
```

On node 2, manually specify `id = 1` and perform the write commit:

```
-- The user manually specified the auto-increment.
insert into t1(id, f1, f2) values(1, 1,1);
```

This may result in duplicate `id` values. Therefore, when an auto-increment column is defined, it is recommended to define it as a standalone primary key/unique index.

Storage Engine

TDSQL Boundless only supports creating and using tables with the RocksDB storage engine, and does not allow modification of the table engine. Any ENGINE clause specified in a CREATE TABLE statement will be automatically converted to ENGINE=ROCKSDB.

Note:
The Blackhole engine is an exception and can be used.

Character Sets and Collations

Note:
Starting from version 8.0.29, the official MySQL changed its default character set from UTF8 to UTF8MB4.

TDSQL Boundless retains the older default of utf8mb3 (aliased as utf8). To ensure maximum compatibility and portability, we strongly recommend explicitly defining the character set when creating tables and database connections.

Item	TDSQL Boundless	MySQL (before 8.0.29)	MySQL (8.0.29+)
Default Character Set	UTF8MB3	UTF8MB3	UTF8MB4
Default Collation	UTF8_GENERAL_CI	UTF8_GENERAL_CI	UTF8MB4_0900_AI_CI

SQL Broadcast

TDSQL Boundless provides a `/*# broadcast */` SQL hint to broadcast a query to all nodes in the instance. This is useful for commands that are otherwise node-specific. For example,

- `SET GLOBAL` normally only applies to the current node. Use `/*# broadcast */ SET GLOBAL` to apply it to all nodes to make it take effect globally.
- `SHOW PROCESSLIST` displays the processlist of the currently connected node. Use `/*# broadcast */SHOW PROCESSLIST;` or `SELECT * FROM information_schema.processlist` to view processes from all nodes.

Log Output

The general log (`general_log`) and slow query log (`slow_log`) can only be written to files. Setting `log_output=TABLE` is not supported.

DDL Differences

TDSQL Boundless supports more online DDL operations than standard MySQL, including online changes to column types, character sets, and converting between partitioned and non-partitioned tables. However, unlike MySQL, less commonly used DDL operations—such as adding/dropping primary keys, adding a column with an expression as a default value (`ADD COLUMN DEFAULT (a+a)`), or altering tables with triggers—are not supported online in TDSQL Boundless. If errors occur during such DDL operations, disable Online DDL mode by executing `set tdsql_use_online_copy_ddl = 0` before retrying.
For more detailed differences, please refer to the OnlineDDL documentation.

Note:
If errors occur during operations such as adding/dropping primary keys, adding columns with an expression as a default value (`ADD COLUMN DEFAULT (a+a)`), or modifying tables involving triggers, disable Online DDL mode by executing `set tdsql_use_online_copy_ddl = 0` before retrying. Note that disabling Online DDL mode will lock the table and block writes to it during DDL execution.

Other Differences

- `SHOW VARIABLES;` and `SHOW GLOBAL STATUS;` display the status information of the currently connected node only. When different nodes are connected to, the information displayed may differ.
- LOCK TABLE statements do not actually lock tables or block reads/writes. Tools that rely on LOCK TABLE, such as mysqldumper, may not function as expected.

Experimental Features

The following features are considered experimental. Use them only under specific conditions.

Note:
Each experimental feature has its own independent toggle and default value. To enable/disable a feature, connect to your TDSQL Boundless instance and execute the following SQL statement (using tdsql_enable_trigger as an example): `SET PERSIST tdsql_enable_trigger=ON;` .

Feature Name	Switch	Default Value	Description
Trigger	tdsql_enable_trigger	OFF	Not recommended for production use.
View	tdsql_enable_view	ON	Avoid UPDATE operations on views.

			Avoid ALTER on view definitions.
Stored Procedure	tdsql_enable_procedure	ON	Avoid ALTER on procedure definitions.
Function	tdsql_enable_function	ON	Avoid ALTER on function definitions.

Use Limits

Last updated：2026-02-10 11:05:59

Table Creation Limits per Instance

Each TDSQL Boundless instance has a limit on the number of tables it can create. This limit depends on the node specification and the number of nodes in the instance. If this limit is exceeded, the corresponding error code (EC_MC_CREATE_DATA_OBJ_TASK_MAX_TABLE_COUNT_EXCEEDED) will be returned.

Node Specification (CPU & Memory)	Max Tables (Basic Edition)	Max Tables (Cluster Edition)
1 Core 2 GB	10000	10000
2 Cores and 4 GB	10000	10000
4 Cores 8 GB	10000	20000
8 Cores and 16 GB	10000	20000
16 Cores 32 GB	20000	40000
32 Cores 64 GB	40000	80000

Row Size Limit

Parameter	Default Value	Description
tdstore_txn_max_entry_size	64M	Limits the maximum size of a single row within a transaction. The maximum configurable value is 512 MB.

Identifier Length Limits

Identifier Type	Maximum Length (Characters)
Database	64
Table	64
Column	64
Index	64

View	64

Per–Table Limits

Limit Type	Upper Limit
Columns	4096
Indexes	64
Rows	Unlimited
Size	Unlimited
Partitions	8,192

Index Limits

Limit Type	Upper Limit	Description
Index length	3,072 bytes	Equivalent to 768 characters using a 4–byte UTF–8 encoding.
Number of index columns	16 columns	The maximum number of columns that can be included in a single index.

Online DDL in TDSQL Boundless

Last updated：2026-02-10 11:05:59

This document outlines support for Online DDL (Data Definition Language) operations in TDSQL Boundless. Most common DDL operations are supported online, including many inplace operations that only require metadata changes.

Support for Online DDL

Secondary Index Operations

Operation	Syntax Supported	Online Change	Reason for Not Supporting Online Change	Metadata-Only	Data Operation	Permits Concurrent DML
Creating/Adding a secondary index	Yes	Yes	/	No	Requires data backfill	Yes
Dropping an index	Yes	Yes	/	No	Requires asynchronous data deletion	Yes
Renaming an index	Yes	Yes	/	No	/	Yes
Adding a FULLTEXT index	No	No	Not support this operation	/	/	/
Adding a SPATIAL index	No	No	Not support this operation	/	/	/
Changing the index type	No	No	Reason for no support: The engine only supports the default index type, not B-tree, Hash, etc.	/	/	/

Primary Key Operations

DDL operations that modify a primary key are not supported in online mode. They require locking the table and performing a full copy.

Operation	Syntax Supported	Online Change	Reason for Not Supporting Online Change	Metadata-Only	Data Operation	Permits Concurrent DML
Adding a primary key	Yes	No	Data reorganization is required.	No	Requires table lock and copy	No
Dropping a primary key	Yes	No	Data reorganization is required.	No	Requires table lock and copy	No
Dropping a primary key and adding another	Yes	Yes	/	No	Requires table lock and copy	Yes

Column Operations

Operatio	Syntax Supported	Online Change	Reason for Not Supporting Online Change	Metadata-Only	Data Operation	Permits Concurrent DML
Adding a column (at the end)	Yes	Yes	/	Yes		Yes
Adding a column (not at the end)	Yes	Yes	/	No	Requires Online Copy	Yes
Dropping a column	Yes	Yes	/	No	Requires Online Copy	Yes
Renaming a column	Yes	Yes	/	Yes	/	Yes

Reordering columns	Yes	Yes	/	No	Requires Online Copy	Yes
Setting a column default value	Yes	Yes	/	Yes	/	Yes
Changing a column data type	Yes	Yes	/	No	Requires Online Copy	Yes
Extending VARCHAR column size (not in PK)	Yes	Yes	/	Varies*	Varies*	Yes
Extending VARCHAR column size (in PK)	Yes	Yes	/	No	Requires Online Copy	Yes
Dropping a column default value	Yes	Yes	/	Yes	/	Yes
Changing the auto-increment value	Yes	Yes	/	Yes	/	Yes
Making a column NULL	Yes	Yes	/	No	Requires Online Copy	Yes
Making a column NOT NULL	Yes	Yes	/	No	Requires Online Copy	Yes
Modifying ENUM or SET Definition	Yes	Yes	/	Yes	/	Yes

Table Operations

Operation	Syntax Supported	Online Change	Reason for Not Supporting	Metadata-Only	Data Operation	Permits Concurrent DML

			Online Change			
Changing the ROW_FORMAT	No	No	Not support this operation	/	/	/
Changing the KEY_BLOCK_SIZE	No	No	Not support this operation	/	/	/
Setting persistent table statistics	No	No	Not support this operation	/	/	/
Specifying a character set	Yes	Yes	/	No	Requires Online Copy	Yes
Converting a character set	Yes	Yes	/	No	Requires Online Copy	Yes
Optimizing a table	Yes	/	This command returns success immediately but performs no action, as the engine's automatic compaction makes it unnecessary	/	/	/
Rebuilding with the FORCE option	Yes	Yes	/	No	Requires Online Copy	Yes
Performing a null rebuild	No	No	Not support this operation	/	/	/
Renaming a table	Yes	Yes	/	Yes	/	Yes

Partitioning Operations

Operation	Syntax Supported	Online Change	Reason for Not Supporting Online Change	Metadata-Only	Data Operation	Permits Concurrent DML
PARTITION BY	Yes	Yes	/	No	Requires Online Copy	Yes
ADD PARTITION	Yes	Yes	/	Yes	/	Yes
DROP PARTITION	Yes	Yes	/	Yes	/	Yes
DISCARD PARTITION	No	No	Not support this operation	/	/	/
IMPORT PARTITION	No	No	Not support this operation	/	/	/
TRUNCATE PARTITION	Yes	Yes	/	/	Async data deletion	Yes
COALESCE PARTITION	Yes	Yes	/	No	Requires Online Copy	Yes
REORGANIZE PARTITION	Yes	Yes	/	No	Requires Online Copy	Yes
EXCHANGE PARTITION	No	No	Not support this operation	/	/	/
ANALYZE PARTITION	No	No	Not support this operation	/	/	/
CHECK PARTITION	No	No	Not support this operation	/	/	/
OPTIMIZE PARTITION	Yes	/	This command returns success immediately	/	/	/

			but performs no action, as the engine's automatic compaction makes it unnecessary			
REBUILD PARTITION	Yes	Yes	/	No	Requires Online Copy	Yes
REPAIR PARTITION	Yes	No	Not counted as DDL	/	/	/
REMOVE PARTITIONING	Yes	Yes	/	No	Requires Online Copy	Yes

Tablespace Operations

Tablespaces operations are not supported.

Generated Column Operations

Generated Column operations are not supported.

Foreign Key Operations

Foreign Key operations are not supported.

Comparison with Community MySQL 8.0

Both the Inplace and Online Copy operations of TDSQL Boundless and the Instant and Inplace operations of Community Edition MySQL 8.0 are considered Online operations. Their differences for common DDL operations are compared as follows.

Operation	TDSQL Boundless	MySQL
Converting a character set	Allows concurrent DML	Blocks concurrent DML
Changing the column data type	Allows concurrent DML	Blocks concurrent DML

PARTITION BY	Allows concurrent DML	Blocks concurrent DML
Extending VARCHAR column size	Allows concurrent DML	Support is conditional, with limits on X in VARCHAR(X).
Adding a primary key	Blocks concurrent DML	Allows concurrent DML

Note:

- TDSQL Boundless currently does not yet support a small number of DDL operations. Therefore, this section only explains the differences between TDSQL Boundless and the community edition MySQL 8.0 regarding their commonly supported features.
- For the type for extending the size of VARCHAR columns, the community edition MySQL 8.0 limits the character length specified by VARCHAR(X), mainly involving the pack length for storage. Assuming that UTF8MB4 is used as the character set, the number of bytes occupied by a field is X*4.

If the old size (X * 4) and new size (Y * 4) are both less than 256 bytes, the operation is INPLACE;
If the old size is less than 256 but the new size is greater than or equal to 256 bytes, the operation is not INPLACE.
Example:

```
CREATE TABLE t1(a INT PRIMARY KEY, b VARCHAR(10)) ENGINE=InnoDB DEFAULT
CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
ALTER TABLE t1 CHANGE COLUMN b b VARCHAR(20), ALGORITHM=INPLACE, LOCK=NO
NE; → Succeeded.
ALTER TABLE t1 CHANGE COLUMN b b VARCHAR(64), ALGORITHM=INPLACE, LOCK=NO
NE; → Failed.
```

Columnar Storage Compatibility Specifications

SQL Syntax Compatibility

Last updated: 2026-02-10 11:05:59

SQL Syntax Compatibility Statement

The TDSQL Boundless columnar vector engine supports most standard SQL queries and all TPC-H queries. However, it has its own characteristics and may fall back to the TDSQL Boundless row storage engine for execution in certain scenarios due to unsupported features. Specific limitations are as follows:

User-Defined Variables

The vector engine does not support user-defined variables. When a query SQL contains user-defined variables (for example, `SELECT @myvar + col FROM table`), the query will not be executed through the vector engine.

```
SET @cnt = 1;

SELECT *
FROM t1 FORCE INDEX (date_col)
WHERE
    col1 + 10000000 > (@cnt := @cnt + 1)
    AND date_col >= DATE('2014-04-01')
    AND date_col < DATE('2014-04-01' + INTERVAL 1 DAY)
ORDER BY date_col DESC;

SELECT @cnt;
```

SQL_CALC_FOUND_ROWS / FOUND_ROWS()

The vector engine temporarily does not support the `SQL_CALC_FOUND_ROWS` related feature. SQL with the following keywords will fall back to the TDSQL TP engine for execution.

WITH RECURSIVE cte

The vector engine currently does not support the Recursive Common Table Expressions feature and will fall back to the TDSQL TP engine for execution.

UNION/UNION ALL + ORDER BY

For UNION/UNION ALL + ORDER BY queries, the `ORDER BY` fields supported by the vector engine must be fields projected in the subquery within the `UNION`. Otherwise, it will fall back to the TDSQL TP engine for execution.

```
--Suppoted--
SELECT a n FROM t1 UNION SELECT b n FROM t2 ORDER BY n;
--Unsuppoted--
SELECT a n FROM t1 UNION SELECT b n FROM t2 ORDER BY n * 2;
```

Character Set and Collation

The vector engine supports limited character sets and collations. Others will fall back to the TDSQL TP engine for execution.

- Supported character sets: "utf8", "utf8mb3", "utf8mb4", "binary".
- Unsupported collations: "utf8_german2_ci", "utf8mb4_czech_ci", "latin1_german2_ci", "latin1_bin", "utf8mb4_lithuanian_ci", "utf8_lithuanian_ci", "utf8_czech_ci", "utf8mb4_polish_ci".

  Table-level restriction: as long as any field in the table definition does not belong to the character sets mentioned above, the operation will fall back.
- Character set compatibility: When string types are compared, the vector engine is case-insensitive but accent-sensitive. This may lead to inconsistencies with MySQL's TP results for accented characters. For example:

```
tdsql [db1]> create table t1(c1 varchar(32), c2 varchar(32));
Query OK, 0 rows affected (0.70 sec)

tdsql[db1]> show create table t1;
+-------+-----------------------------------------------------------
-------------------------------------------------------------------+
| Table | Create Table
|
+-------+-----------------------------------------------------------
-------------------------------------------------------------------+
| t1    | CREATE TABLE `t1` (
  `c1` varchar(32) DEFAULT NULL,
  `c2` varchar(32) DEFAULT NULL
) ENGINE=ROCKSDB DEFAULT CHARSET=utf8mb3 |
+--------+-----------------------------------------------------------
--------------------------------------------------------------------+
1 row in set (0.00 sec)

tdsql[db1]> insert into t1 values('hello','hElLÖ');
Query OK, 1 row affected (0.02 sec)

tdsql[db1]> select * from t1;
+--------+--------+
| c1     | c2     |
+--------+--------+
| hello  | hElLÖ  |
+--------+--------+
1 row in set (0.00 sec)

tdsql[db1]> select c1=c2 from t1;
+--------+
| c1=c2  |
+--------+
|      1 |
+--------+
1 row in set (0.00 sec)

tdsql[db1]> set htap_routing_strategy='vector_engine';
Query OK, 0 rows affected (0.00 sec)

tdsql [db1]> set htap_query_passthrough_mode=off;
Query OK, 0 rows affected (0.00 sec)

tdsql[db1]> set htap_print_deparsed_query=on;
Query OK, 0 rows affected (0.00 sec)

tdsql[db1]> select c1=c2 from t1;
+--------+
| c1=c2  |
+--------+
|      0 |
+--------+
1 row in set, 1 warning (0.01 sec)
```

```
tdsql[db1]> show warnings;
+-------+------+---------------------------------------------------
-----------------------------------------------+
| Level | Code | Message
|
+-------+------+---------------------------------------------------
-----------------------------------------------+
| Note  | 1003 | Deparsed query is : select  ("t1"."c1" = "t1"."c2"
collate nocase ) AS "c1=c2" from "db1"."t1" "t1" |
+-------+------+---------------------------------------------------
-----------------------------------------------+
1 row in set (0.00 sec)
```

Parameters Related to the Vector Engine

No.	Variable Name	Type	Description
1	htap_avoid_some_mysql_optimizations	Boolean	Whether to enable optimizer intervention to optimize vector queries, default (on).
2	htap_enable_deparser_warnings	Boolean	For alarms triggered for VE parsing, default (off).
3	htap_routing_strategy	String	Query policy has the following possible options: 'VECTOR_ENGINE': defaults to using the vector engine. 'DEFAULT': defaults to the TP engine.
4	htap_query_passthrough_mode	Boolean	Whether to perform semantic parsing for the vector engine. When set to on, SQL statements are directly passed to the VE engine to retrieve results. Default (off).
5	htap_print_deparsed_query	Boolean	Whether to output logs from the vector parsing engine to the log file, default (off).
6	htap_enable_ve_execution_fallback	Boolean	When the vector engine reports an error, whether to fallback to TP execution or directly report an error, default (on).

Expression Compatible

Last updated：2026-02-10 11:05:59

Data Types

Type	MySQL Data Types	Compatibility Note
Value Type	BOOLEAN	Compatible
	TINYINT,TINYINT UNSIGNED	Compatible
	SMALLINT,SMALLINT UNSIGNED	Compatible
	INT,INTEGER,INT UNSIGNED,INTEGER UNSIGNED	Compatible
	BIGINT,BIGINT UNSIGNED	Compatible
	FLOAT	Compatible
	DOUBLE	Compatible
	DECIMAL(m,d)	Different value ranges • TDSQL-TP engine supports up to `Decimal(65,30)`. • The vector engine supports up to `Decimal(38)`.
Character Types	CHAR,VARCHAR	• The character-set vector engine only supports the UTF-8 character set and collation, and does not support other character sets such as GBK. • Accent-sensitive collation.
	TINYTEXT,TEXT,MEDIUMTEXT,LONGTEXT	
	JSON	Compatible
	SET	Compatible
	ENUM	Compatible

Binary-Character Types	BINARY,VARBINARY	Compatible
	BIT	Compatible, but the vector engine displays binary strings by default, while MySQL displays hexadecimal strings by default.
	TINYBLOB,BLOB,MEDIUMBLOB, LONGBLOB	Compatible
Datetime and Interval Data Types	YEAR	Compatible
	TIME	• The valid range is '00:00:00'~'23:59:59' in 'HH:MM:SS' format. Values outside this range may exhibit inconsistent behavior; during insertion, the system automatically performs a modulo operation to wrap around to the valid range. • The range for the TDSQL TP engine is '-838:59:59' to '838:59:59' in the format 'HHH:MM:SS'.
	DATE	The vector engine supports the yyyy-mm-dd format but does not support representations like yyyy.mm.dd or yyyy#mm#dd. The valid range is '0001-01-01' to '9999-12-31'. In lenient mode, '0000-00-00' to '0001-01-01' is also supported; however, function behavior may be inconsistent for values outside the valid range.
	DATETIME	The valid range is '0001-01-01 00:00:00' to '9999-12-31 23:59:59' in the format 'YYYY-MM-DD HH:MM:SS'. In lenient mode, '0000-00-00 00:00:00' to '0001-01-01 00:00:00' is also supported; however, function behavior may be inconsistent for values outside the valid range.
	TIMESTAMP	Compatible
Spatial data type	GEOMETRY,POINT,LINESTRING,POLYGON,MULTIPOINT,MULTILINESTRING,MULTIPOLYGON,GEOMETRYCOLLECTION	Incompatible

Note:

- The TDSQL TP engine handles string > double conversions flexibly. Even when strings contain non-numeric characters, conversion is still performed. For example: ' ' → double (converts empty strings to 0.0), '1x' → double (outputs 1.0).
- The Vector Engine cannot handle conversions of such invalid values. If such conversions are encountered, you can enable the `htap_enable_ve_execution_fallback` switch to fall back to the TP engine for computation, or correct the invalid value conversion to continue leveraging the efficient computation of the Vector Engine.

Control Function

Function Name	Whether Supported	Use limits
CASE	Supported	-
IF()	Supported	-
IFNULL()	Supported	-
NULLIF()	Supported	-

Numerical Functions

Function Name	Whether Supported	Use limits
%, MOD	Supported	-
*	Supported	-
+	Supported	-
-	Supported	-
-	Supported	-
/	Supported	-
ABS()	Supported	-
ACOS()	Supported	-
ASIN()	Supported	-
ATAN()	Supported	-
ATAN2(), ATAN()	Not supported	-

CEIL()	Supported but with differences.	Parameters in the vector engine are treated as floating-point numbers.
CEILING()	Supported but with differences.	Parameters in the vector engine are treated as floating-point numbers.
CONV()	Not supported	-
COS()	Supported	-
COT()	Supported	-
CRC32()	Not supported	-
DEGREES()	Supported	-
DIV	Supported	-
EXP()	Supported	-
FLOOR()	Supported but with differences.	Parameters in the vector engine are treated as floating-point numbers.
LN()	Supported but with differences.	When illegal parameters are entered, the vector engine will throw an error, while MySQL will return `NULL` .
LOG()	Supported but with differences.	When illegal parameters are entered, the vector engine will throw an error, while MySQL will return `NULL` .
LOG10()	Supported but with differences.	When illegal parameters are entered, the vector engine will throw an error, while MySQL will return `NULL` .
LOG2()	Supported but with differences.	When illegal parameters are entered, the vector engine will throw an error, while MySQL will return `NULL.`
MOD()	Supported	-
PI()	Supported	-
POW()	Supported	-
POWER()	Supported	-
RADIANS()	Supported	-

RAND()	Not supported	–
ROUND()	Supported but with differences.	The scenario where d is negative in `ROUND(x, d)` is not supported.
SIGN()	Supported	–
SIN()	Supported	–
SQRT()	Supported	–
TAN()	Supported	–
TRUNCATE()	Not supported	–

Date and Time Functions

Function Name	Whether Supported	Use limits
ADDDATE()	Supported but with differences.	• In the vector engine, the type of return value of `ADDDATE()` is uniformly DATETIME. • In MySQL, the return type of `ADDDATE()` corresponds to the type of the input parameter.
ADDTIME()	Not supported	–
CONVERT_TZ()	Not supported	–
CURDATE()	Supported	–
CURRENT_DATE(),CURRENT_DATE	Supported	–
CURRENT_TIME(),CURRENT_TIME	Not supported	–
CURRENT_TIMESTAMP(),CURRENT_TIMESTAMP	Supported	–
CURTIME()	Not supported	–
DATE()	Supported	–

DATE_ADD()	Supported but with differences.	Vector engine does not support the `SECOND_MICROSECOND` unit.
DATE_FORMAT()	Not supported	Vector engine lacks the native `DATE_FORMAT()` function.
DATE_SUB()	Supported but with differences.	Vector engine does not support the `SECOND_MICROSECOND` unit.
DATEDIFF()	Not supported	–
DAY()	Supported	–
DAYNAME()	Supported	–
DAYOFMONTH()	Supported	–
DAYOFWEEK()	Supported but with differences.	• Vector engine: Return value ranges from 0 to 6, where 0=Sunday, 1=Monday, ..., 6=Saturday. • MySQL: Return value ranges from 1 to 7, where 1=Sunday, 2=Monday, ..., 7=Saturday.
DAYOFYEAR()	Supported	–
EXTRACT()	Supported but with differences.	Vector engine only supports YEAR, MONTH, DAY, HOUR, MINUTE, SECOND, QUARTER, WEEK.
FROM_DAYS()	Not supported	–
FROM_UNIXTIME()	Not supported	–
GET_FORMAT()	Not supported	–
HOUR()	Supported but with differences.	• Vector engine: `HOUR()` only supports DATETIME and TIMESTAMP parameter types. • MySQL: `HOUR()` supports DATETIME, TIMESTAMP, and TIME parameter types.
LAST_DAY()	Supported but with	When illegal parameters are entered, the vector engine will throw an error, while MySQL will return `N`

	differences.	`ULL` .
LOCALTIME(),LOCALTIME	Supported but with differences.	Vector engine does not support the format for millisecond precision of `LOCALTIME(fsp)` .
LOCALTIMESTAMP,LOCALTIMESTAMP()	Supported but with differences.	Vector engine does not support the format for millisecond precision of `LOCALTIME(fsp)` .
MAKEDATE()	Not supported	–
MAKETIME()	Not supported	–
MICROSECOND()	Supported but with differences.	• Vector engine: `MICROSECOND()` supports DATETIME and TIMESTAMP parameter types. • MySQL: `MICROSECOND()` supports DATETIME, TIMESTAMP, and TIME parameter types.
MINUTE()	Supported	–
MONTH()	Supported	–
MONTHNAME()	Supported	–
NOW()	Supported	–
PERIOD_ADD()	Not supported	–
PERIOD_DIFF()	Not supported	–
QUARTER()	Supported	–
SEC_TO_TIME()	Not supported	–
SECOND()	Supported but with differences.	• Vector Engine: `SECOND()` only supports DATETIME and TIMESTAMP parameter types. • MySQL: `SECOND()` supports DATETIME, TIMESTAMP, and TIME parameter types.

STR_TO_DATE()	Supported but with differences.	Vector Engine does not support `%X` , `%V` , `%u` , and returns `NULL` for format mismatch.
SUBDATE()	Supported but with differences.	• Vector Engine: The return type of `SUBDATE()` is uniformly DATETIME. • MySQL: The return type of `SUBDATE()` is consistent with the type of the input parameter.
SUBTIME()	Not supported	–
SYSDATE()	Not supported	–
TIME()	Supported	–
TIME_FORMAT()	Not supported	–
TIME_TO_SEC()	Not supported	–
TIMEDIFF()	Not supported	–
TIMESTAMP()	Supported but with differences.	Vector Engine supports only `TIMESTAMP(expr)` , but not `TIMESTAMP(expr1, expr2)` .
TIMESTAMPADD()	Supported	–
TIMESTAMPDIFF()	Not supported	–
TO_DAYS()	Not supported	–
TO_SECONDS()	Not supported	–
UNIX_TIMESTAMP()	Supported but with differences.	Vector engine does not support the `UNIX_TIMESTAMP()` syntax, only supports the `UNIX_TIMESTAMP(date)` syntax.
UTC_DATE()	Not supported	–

UTC_TIME()	Not supported	–
UTC_TIMESTAMP()	Not supported	–
WEEK()	Not supported	–
WEEKDAY()	Supported but with differences.	• Vector engine: Return values ranging from 0 to 6, where 0=Sunday, 1=Monday, ..., 6=Saturday. • MySQL: Return values ranging from 1 to 7, indicating 1=Sunday, 2=Monday, ..., 7=Saturday.
WEEKOFYEAR()	Supported	–
YEAR()	Supported	–
YEARWEEK()	Supported	–

String Functions

Function Name	Whether Supported	Use limits
ASCII()	Supported	–
BIN()	Not supported	–
BIT_LENGTH()	Supported	–
CHAR()	Not supported	–
CHAR_LENGTH()	Supported	–
CHARACTER_LENGTH()	Supported	–

CONCAT()	Supported	–
CONCAT_WS()	Supported	–
ELT()	Not supported	–
EXPORT_SET()	Not supported	–
FIELD()	Not supported	–
FIND_IN_SET()	Not supported	–
FORMAT()	Not supported	–
FROM_BASE64()	Supported	–
HEX()	Supported	–
INSERT()	Not supported	–
INSTR()	Supported	–
LCASE()	Supported	–
LEFT()	Supported	–
LENGTH()	Not supporte	–

	d	
LIKE	Supported	–
LOAD_FILE()	Not supported	–
LOCATE()	Not supported	–
LOWER()	Supported	–
LPAD()	Supported but with differences.	In the vector engine, the `LPAD()` function does not support parameters being empty strings simultaneously, and also does not support other forms such as binary.
LTRIM()	Supported	–
MAKE_SET()	Not supported	–
MATCH()	Not supported	–
MID()	Supported	–
NOT LIKE	Supported	–
NOT REGEXP	Supported	–
OCT()	Not supported	–
OCTET_LENGTH()	Supporte	–

	d	
ORD()	Supported but with differences.	• Vector engine: empty string returns `-1`. • MySQL: empty string returns `0`.
POSITION()	Not supported	–
QUOTE()	Not supported	–
REGEXP	Supported	–
REGEXP_INSTR()	Not supported	–
REGEXP_LIKE()	Supported	–
REGEXP_REPLACE()	Supported	–
REGEXP_SUBSTR()	Not supported	–
REPEAT()	Supported	–
REPLACE()	Supported	–
REVERSE()	Supported	–
RIGHT()	Supported	–
RLIKE	Supported	–

RPAD()	Supported but with differences.	In the vector engine, the `RPAD()` function does not support both parameters being empty strings.
RTRIM()	Supported	–
SOUNDEX()	Not supported	–
SOUNDS LIKE	Not supported	–
SPACE()	Supported	–
STRCMP()	Not supported	–
SUBSTR()	Supported	–
SUBSTRING()	Supported	–
SUBSTRING_INDEX()	Supported	–
TO_BASE64()	Supported	–
TRIM()	Supported	–
UCASE()	Supported	–
UNHEX()	Supported but with differences.	When illegal parameters are entered, the vector engine will throw an error, while MySQL will return `NULL`.

UPPER()	Supported	–
WEIGHT_STRING()	Not supported	–

Type Conversion Functions

Function Name	Whether Supported	Use limits
CAST()	Supported but with differences.	The vector engine only supports conversion between valid formats.
CONVERT()	Not supported	–

Bit Manipulation Functions

Function Name	Whether Supported	Use limits
&	Supported	
>>	Supported but with differences.	The vector engine shifts based on signed numbers, and the results are presented as unsigned numbers. Execution example: (see below) MySQL execution example: (see below)
<<	Supported but with differences.	

```
tdsql[***]> SELECT -8 >> 1 from t;
+----------------------+
| -8 >> 1              |
+----------------------+
| 18446744073709551612 |
| 18446744073709551612 |
+----------------------+
2 rows in set, 1 warning (0.02 sec)
```

MySQL execution example:

```
MySQL [***]> SELECT -8 >> 1 ;
+---------------------+
| -8 >> 1             |
+---------------------+
| 9223372036854775804 |
```

		`+--------------------+` `1 row in set, 1 warning (0.00 sec)`
^	Supported	–
BIT_COUNT	Supported	–
|	Supported	–
~	Supported	–

JSON Functions

Function Name	Whether Supported	Use limits
->	Supported	–
->>	Not supported	–
JSON_ARRAY()	Supported	–
JSON_ARRAY_APPEND()	Not supported	–
JSON_ARRAY_INSERT()	Not supported	–
JSON_CONTAINS()	Not supported	–
JSON_CONTAINS_PATH()	Not supported	–
JSON_DEPTH()	Not supported	–
JSON_EXTRACT()	Supported	–
JSON_INSERT()	Not supported	–
JSON_KEYS()	Supported	–
JSON_LENGTH()	Not supported	–
JSON_MERGE()	Not supported	–
JSON_MERGE_PATCH()	Supported	–
JSON_MERGE_PRESERVE()	Not supported	–
JSON_OBJECT()	Supported	–

JSON_OVERLAPS()	Not supported	–
JSON_PRETTY()	Supported	–
JSON_QUOTE()	Not supported	–
JSON_REMOVE()	Not supported	–
JSON_REPLACE()	Not supported	–
JSON_SCHEMA_VALID()	Not supported	–
JSON_SCHEMA_VALIDATION_REPORT()	Not supported	–
JSON_SEARCH()	Not supported	–
JSON_SET()	Not supported	–
JSON_STORAGE_FREE()	Not supported	–
JSON_STORAGE_SIZE()	Not supported	–
JSON_TABLE()	Not supported	–
JSON_TYPE()	Not supported	–
JSON_UNQUOTE()	Not supported	–
JSON_VALID()	Supported	–
JSON_VALUE()	Not supported	–
MEMBER OF()	Not supported	–

Aggregation Functions

Function Name	Whether Supported	Use limits
AVG()	Supported but with differences.	Vector engines only support numeric and string values.
BIT_AND()	Not supported	–
BIT_OR()	Not supported	–
BIT_XOR()	Not supported	–

COUNT()	Supported	–
COUNT(DISTINCT)	Supported	–
GROUP_CONCAT()	Supported but with differences.	Vector engines do not support multiple columns.
JSON_ARRAYAGG()	Not supported	–
JSON_OBJECTAGG()	Not supported	–
MAX()	Supported but with differences.	Vector engines only support numeric and string values.
MIN()	Supported but with differences.	Vector engines only support numeric and string values.
STD()	Supported but with differences.	Vector engines only support numeric and string values.
STDDEV()	Supported but with differences.	Vector engines only support numeric and string values.
STDDEV_POP()	Supported but with differences.	Vector engines only support numeric and string values.
STDDEV_SAMP()	Supported but with differences.	Vector engines only support numeric and string values.
SUM()	Supported but with differences.	Vector engines only support numeric and string values.
VAR_POP()	Supported but with differences.	Vector engines only support numeric and string values.
VAR_SAMP()	Supported but with differences.	Vector engines only support numeric and string values.
VARIANCE()	Supported but with differences.	Vector engines only support numeric and string values.

Window Functions

Function Name	Whether Supported	Limits of use
CUME_DIST()	Supported	–

DENSE_RANK()	Supported	–
FIRST_VALUE()	Supported	–
LAG()	Supported	–
LAST_VALUE()	Supported	–
LEAD()	Supported	–
NTH_VALUE()	Supported	–
NTILE()	Supported	–
PERCENT_RANK()	Supported	–
RANK()	Supported	–
ROW_NUMBER()	Supported	–

Other Functions

Function Name	Whether Supported	Limits of use
<=>	Supported but with differences.	Incompatible with `NULL` behavior.

Statement Structure

Grammar Point	Whether Supported	Use limits
Subqueries	Supported but with differences.	The vector engine does not support the quantifiers `ALL` and `ANY` .

UNION	Supported but with differences.	The vector engine does not support.
		`(SELECT DISTINCT c1`
		` FROM t1`
		` GROUP BY c2)`
		`UNION ALL`
		`(SELECT c1`
		` FROM t1);`

Outer join	Supported but with differences.	Outer join fields do not support using correlated subquery fields.
Grouping with window functions.	Supported but with differences.	When using window functions and grouping, it does not support including primary keys in the grouping keys.

HBase Compatibility

Last updated：2026-02-10 11:05:59

TDSQL Boundless (HBase Compatibility Mode) Supported Data Types

No.	Data Type	Name	Example	Description
1	String	string data	"Hello TDSQL World!"	
2	Integer	integer data	12345	boundary values: Integer.MAX_VALUE, Integer.MIN_VALUE
3	Long	long integer data	9876543210L	boundary values: Long.MAX_VALUE, Long.MIN_VALUE
4	Double	double	3.141592654	boundary values: Double.MAX_VALUE, Double.POSITIVE_INFINITY, Double.NaN
5	Float	float	2.71828f	boundary values: Float.MAX_VALUE, Float.POSITIVE_INFINITY, Float.NaN
6	Boolean	Boolean	TRUE	
7	Short	short	32767	
8	Byte	byte data	127	
9	Binary Data	binary data pattern	{0x00, 0x01, 0x02, 0x03, 0x04, 0x05}	
10	ASCII Data	ASCII encoded data	{0x48, 0x65, 0x6C, 0x6C, 0x6F} (Hello)	
11	Empty Data	Null Data	{}	
12	List	List Type	["product_001", "product_002",	

			"product_003", "product_002"]	
13	Set	Collection Type	[1001, 1002, 1003, 1004, 1005]	
14	Text	Text Type		
15	Sparse String	Sparse String		HBase Sparse String is used to store sparse string data, which can save storage space.
16	Sparse Binary	Sparse Byte Array		HBase Sparse Byte Array is used to store sparse binary data, which can save storage space.

Interface Compatibility

HTable Interface

TDSQL Boundless HBase-Client client implements the HTableInterface interface, and its interface compatibility is as follows:

No.	Modifier and Type	Methods and Descriptions	TDSQL Boundless Compatibility
1	Result	append(Append append)	Supported.
2	Object[]	batch(List<? extends Row> actions) Similar to batch(List, Object[]), but returns an array of results rather than using the result parameter reference.	Supported
3	void	batch(List<? extends Row> actions, Object[] results) Methods for executing Deletes, Gets, Puts, Increments, Appends, and RowMutations in batches.	Supported
4	boolean	checkAndDelete(byte[] row, byte[] family, byte[] qualifier, byte[] value, Delete delete)	Not supported

		Atomically checks whether the value of the row/column family/column qualifier matches the expected value.	
5	boolean	checkAndPut(byte[] row, byte[] family, byte[] qualifier, byte[] value, Put put) Atomically checks whether the value of the row/column family/column qualifier matches the expected value.	Not supported
6	void	close() Release all occupied resources or discard pending changes in internal buffers.	Supported
7	<T extends CoprocessorProtocol,R> Map<byte[],R>	coprocessorExec(Class protocol, byte[] startKey, byte[] endKey, Batch.Call<T,R> callable) Invokes the passed Batch.Call on the CoprocessorProtocol instance in the chosen region.	Not supported.
8	<T extends CoprocessorProtocol,R> void	coprocessorExec(Class protocol, byte[] startKey, byte[] endKey, Batch.Call<T,R> callable, Batch.Callback callback) Invoke the passed Batch.Call on the CoprocessorProtocol instance in the selected region, and use a callback to process the results.	Not supported.
9	<T extends CoprocessorProtocol > T	coprocessorProxy(Class protocol, byte[] row) Creates and returns a proxy for accessing the CoprocessorProtocol instance in the region containing the specified row.	Not supported.
10	void	delete(Delete delete) Delete the specified cell or row.	Supported
11	void	delete(List<Delete> deletes) Batch delete the specified cells or rows.	Supported
12	boolean	exists(Get get) Check whether the table contains the columns specified by Get.	Supported
13	void	flushCommits() Execute all buffered Put operations.	Not supported
14	Result	get(Get get) Extract specific cells from the specified row.	Supported.

15	Result[]	get(List<Get> gets) Perform batch extraction of specific cells from the specified row.	Supported
16	org.apache.hadoop.conf.Configuration	getConfiguration() Returns the Configuration object used by the current instance.	Supported
17	ResultScanner	getScanner(byte[] family) Obtain a scanner for the specified column family in the current table.	Supported
18	ResultScanner	getScanner(byte[] family, byte[] qualifier) Obtain a scanner for the specified column family and column qualifier in the current table.	Supported
19	ResultScanner	getScanner(Scan scan) Returns a scanner for the current table based on the Scan object.	Supported.
20	HTableDescriptor	getTableDescriptor() Obtain the current table's table descriptor.	Not supported
21	byte[]	getTableName() Obtain the name of the current table.	Not supported
22	long	getWriteBufferSize() Returns the maximum size of the write buffer for the current table (in bytes).	Not supported
23	Result	increment(Increment increment) Increment the value of one or more columns in a single row.	Supported
24	long	incrementColumnValue(byte[] row, byte[] family, byte[] qualifier, long amount) Atomically increment the value of the specified column.	Supported
25	long	incrementColumnValue(byte[] row, byte[] family, byte[] qualifier, long amount, Durability durability) Atomically increment the value of the specified column.	Not supported
26	boolean	isAutoFlush() Check whether the "Auto Refresh" feature is enabled.	Not supported

27	void	mutateRow(RowMutations rm) Perform multiple atomic mutation operations on a single row.	Supported
28	void	put(List<Put> puts) Batch insert data into the table.	Supported
29	void	put(Put put) Insert data into the table.	Supported
30	void	setAutoFlush(boolean autoFlush) See setAutoFlush(boolean, boolean).	Not supported
31	void	setAutoFlush(boolean autoFlush, boolean clearBufferOnFail) Enables or disables the "auto-refresh" feature and allows choosing to clear the buffer upon failure.	Not supported
32	void	setWriteBufferSize(long writeBufferSize) Set the buffer size (in bytes).	Not supported

Admin Interface

The TDSQL Boundless HBase-Client implements the Admin interface, and its interface compatibility is as follows:

No.	Modifier and Type	Methods and Descriptions	TDSQL Boundless Compatibility
1	void	addColumnFamily(TableName tableName, ColumnFamilyDescriptor columnFamily) Add a column family to an existing table.	Not supported.
2	void	createTable(TableDescriptor desc) Create a new table.	Supported
3	void	deleteColumnFamily(TableName tableName, byte[] columnFamily) Delete a column family from the table.	Not supported.
4	void	deleteTable(TableName tableName) Delete a table.	Supported.
5	void	disableTable(TableName tableName)	Supported

		Disable the table and wait for completion.	
6	void	enableTable(TableName tableName) Enable the table.	Supported
7	org.apache.hadoop.conf.Configuration	getConfiguration() Retrieve the configuration in use for the instance.	Supported
8	Connection	getConnection() Retrieve the connection used by this object.	Supported
9	TableDescriptor	getDescriptor(TableName tableName) Obtain the table descriptor.	Supported
10	boolean	isTableDisabled(TableName tableName) Check whether the table is disabled.	Supported
11	boolean	isTableEnabled(TableName tableName) Check whether the table is enabled.	Supported
12	List<TableDescriptor>	listTableDescriptors(List<TableName> tableNames) Obtain the list of table descriptors.	Supported
13	List<TableDescriptor>	listTableDescriptors(Pattern pattern) List all user-space tables matching a given pattern.	Supported
14	TableName[]	listTableNames() List all user-space table names.	Supported.
15	void	majorCompact(TableName tableName) Primary compression table.	Not supported
16	void	majorCompact(TableName tableName, byte[] columnFamily) Column families in the primary compression table.	Not supported
17	void	modifyColumnFamily(TableName tableName, ColumnFamilyDescriptor columnFamily) Modify existing column families in the table.	Supported
18	void	modifyTable(TableDescriptor td) Modify existing tables (more IRB-friendly version).	Supported
19	boolean	tableExists(TableName tableName) Check whether the table exists.	Supported.

20	void	truncateTable(TableName tableName) Truncate table.	Supported