Apache ShardingSphere document

Apache ShardingSphere

Aug 27, 2022

Contents

This chapter mainly introduces what Apache ShardingSphere is, as well as its design philosophy and deployment architecture.

For frequently asked questions, please refer to FAQ.

1 What is ShardingSphere

1.1 Introduction

Apache ShardingSphere is an open source ecosystem that allows you to transform any database into a distributed database system. The project includes a JDBC and a Proxy, and its core adopts a micro-kernel and pluggable architecture. Thanks to its plugin-oriented architecture, features can be flexibly expanded at will.

The project is committed to providing a multi-source heterogeneous, enhanced database platform and further building an ecosystem around the upper layer of the platform. Database Plus, the design philosophy of Apache ShardingSphere, aims at building the standard and ecosystem on the upper layer of the heterogeneous database. It focuses on how to make full and reasonable use of the computing and storage capabilities of existing databases rather than creating a brand new database. It attaches greater importance to the collaboration between multiple databases instead of the database itself.

1.1.1 ShardingSphere-JDBC

ShardingSphere-JDBC is a lightweight Java framework that provides additional services at Java's JDBC layer.

1.1.2 ShardingSphere-Proxy

ShardingSphere-Proxy is a transparent database proxy, providing a database server that encapsulates database binary protocol to support heterogeneous languages.

1.2 Product Features

Feature	Definition
Data Sharding	Data sharding is an effective way to deal with massive data storage and computing. ShardingSphere provides distributed database solutions that can scale out computing and storage levels on top of the underlying database.
Distributed Transaction	Transactional capability is key to ensuring database integrity and security and is also one of the databases' core technologies. ShardingSphere provides distributed transaction capability on top of a single database, which can achieve data security across underlying data sources.
Read/write Splitting	Read/write splitting can be used to cope with business access with high stress. Based on its understanding of SQL semantics and the topological awareness of the underlying database, ShardingSphere provides flexible and secure read/write splitting capabilities and can achieve load balancing for read access.
High Availability	High availability is a basic requirement for a data storage and computing platform. ShardingSphere provides access to high-availability computing services based on stateless services. At the same time, it can sense and use the underlying database's HA solution to achieve its overall high availability.
Data Migration	Data migration is the key to connecting data ecosystems. ShardingSphere provides full-scenario data migration capability for users, which can cope with the surge of business data volume.
Federated Query	Federated queries are effective in utilizing data in a complex data environment. ShardingSphere is capable of querying and analyzing complex data across data sources, simplifying and improving the data usage experience.
Data Encryption	Data Encryption is a basic way to ensure data security. ShardingSphere provides a set of data encryption solutions that are complete, secure, transparent, and with low transformation costs.
Shadow Database	In the full-link stress testing scenario, ShardingSphere shadow DB is used for providing data isolation support for complex testing work. The obtained testing result can accurately reflect the system's true capacity and performance.

1.3 Advantages

- Ultimate Performance

Having been polished for years, the driver is close to a native JDBC in terms of efficiency, with ultimate performance.

- Ecosystem Compatibility

The proxy can be accessed by any application using MySQL/PostgreSQL protocol, and the driver can connect to any database that implements JDBC specifications.

- Zero Business Intrusion

In response to database switchover scenarios, ShardingSphere can achieve smooth business migration without business intrusion.

- Low Ops & Maintenance Cost

ShardingSphere offers a flat learning curve to DBAs and is interaction-friendly while allowing the original technology stack to remain unchanged.

- Security & Stability

It can provide enhancement capability based on mature databases while ensuring security and stability.

- Elastic Extention

It supports computing, storage, and smooth online expansion, which can meet diverse business needs.

- Open Ecosystem

It can provide users with flexibility thanks to custom systems based on multi-level (kernel, feature, and ecosystem) plugin capabilities.

1.4 Roadmap

1.5 How to Contribute

ShardingSphere became an Apache Top-Level Project on April 16, 2020. You are welcome to check out the mailing list and discuss via mail.

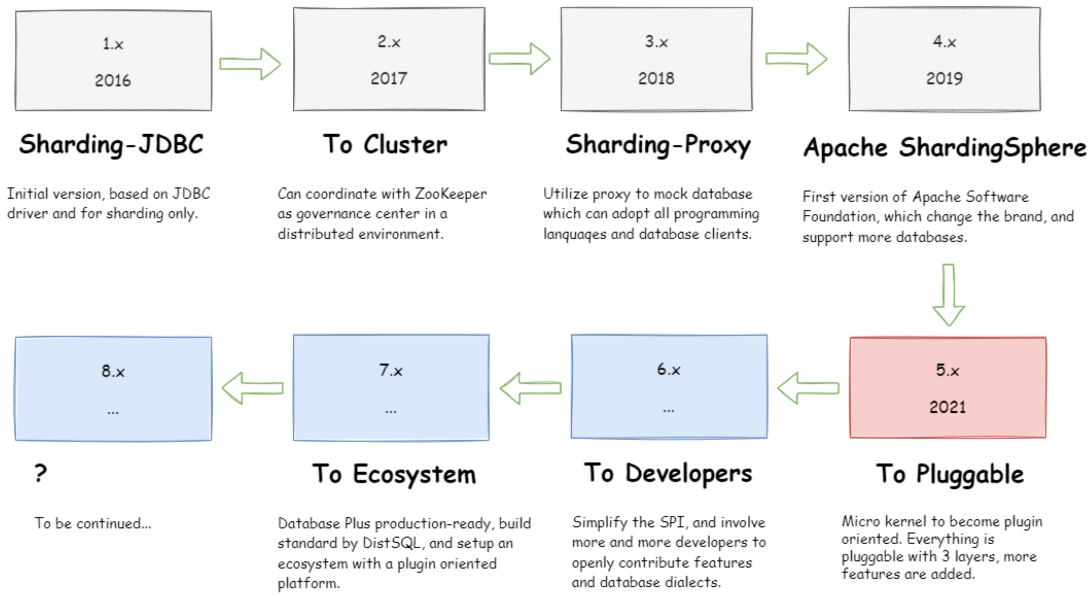


Figure1: Roadmap

2 Design Philosophy

ShardingSphere adopts the database plus design philosophy, which is committed to building the standards and ecology of the upper layer of the database and supplementing the missing capabilities of the database in the ecology.

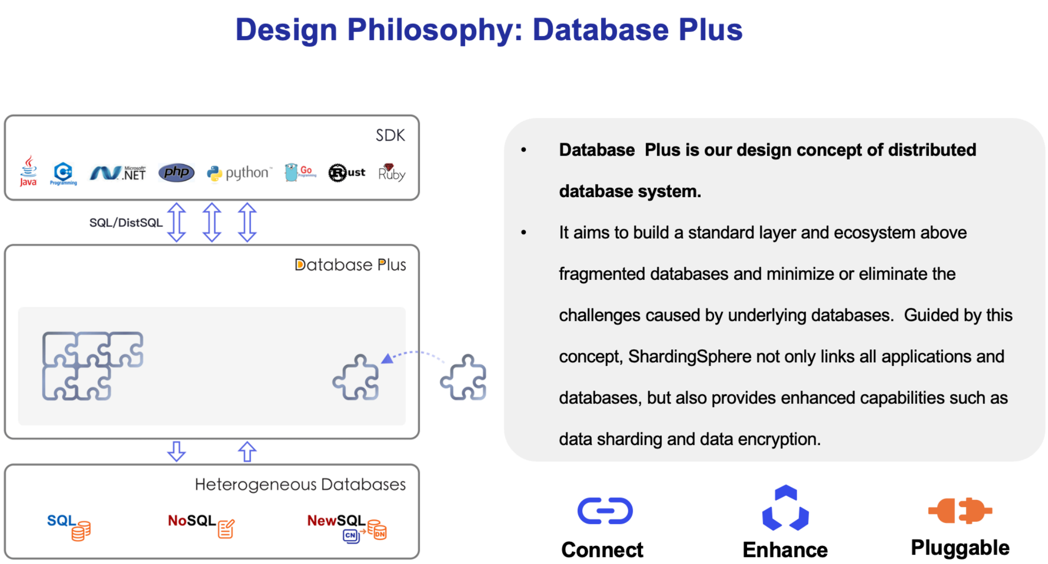


Figure1: Design

2.1 Connect: Create database upper level standard

Through flexible adaptation of database protocols, SQL dialects, and database storage, it can quickly build standards on top of multi-modal heterogeneous databases, while providing standardized connection mode for applications through built-in DistSQL.

2.2 Enhance: Database computing enhancement engine

It can further provide distributed capabilities and traffic enhancement functions based on native database capabilities. The former can break through the bottleneck of the underlying database in computing and storage, while the latter provides more diversified data application enhancement capabilities through traffic deformation, redirection, governance, authentication, and analysis.

2.3 Pluggable: Building database function ecology

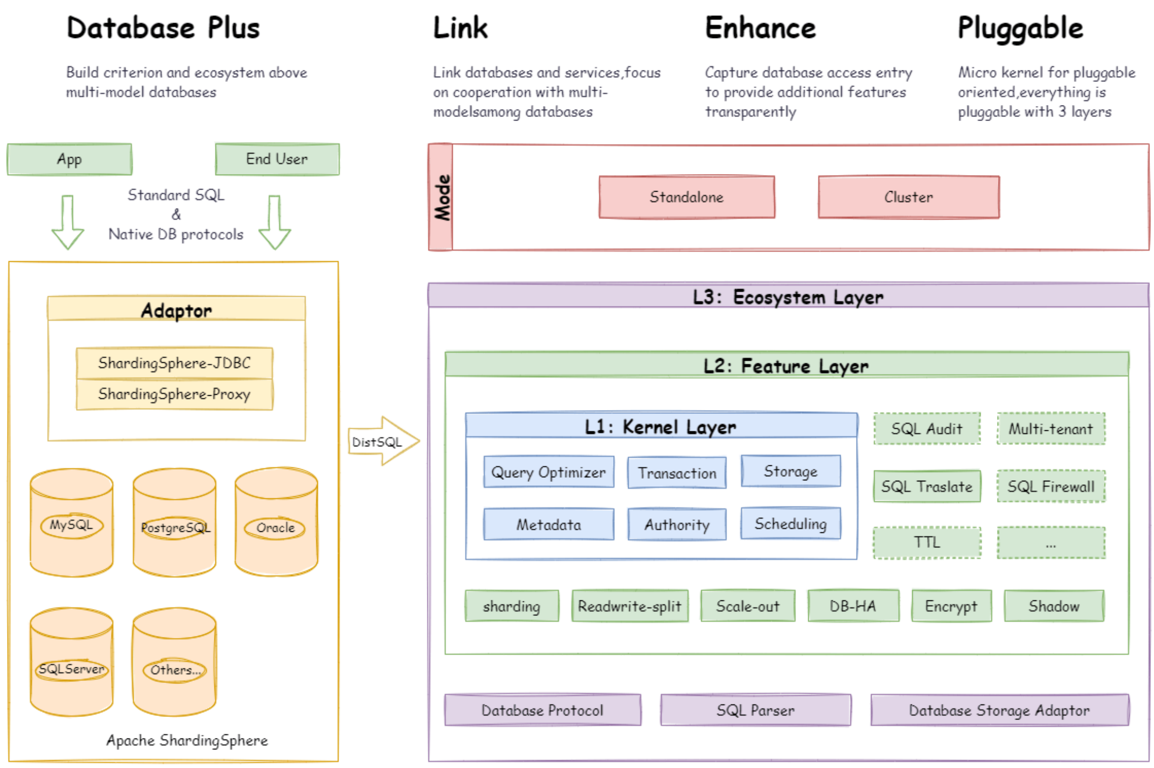


Figure2: Overview

The pluggable architecture of Apache ShardingSphere is composed of three layers - L1 Kernel Layer, L2 Feature Layer and L3 Ecosystem Layer.

2.3.1 L1 Kernel Layer

An abstraction of databases' basic capabilities. All the components are required and the specific implementation method can be replaced thanks to plugins. It includes a query optimizer, distributed transaction engine, distributed execution engine, permission engine and scheduling engine.

2.3.2 L2 Feature Layer

Used to provide enhancement capabilities. All components are optional, allowing you to choose whether to include zero or multiple components. Components are isolated from each other, and multiple components can be used together by overlaying. It includes data sharding, read/write splitting, database high availability, data encryption and shadow database and so on. The user-defined feature can be fully customized and extended for the top-level interface defined by Apache ShardingSphere without changing kernel codes.

2.3.3 L3 Ecosystem Layer

It is used to integrate and merge the current database ecosystems. The ecosystem layer includes database protocol, SQL parser and storage adapter, corresponding to the way in which Apache ShardingSphere provides services by database protocol, the way in which SQL dialect operates data, and the database type that interacts with storage nodes.

3 Deployment

3.1 Deployment

Apache ShardingSphere includes two independent clients: ShardingSphere-JDBC & ShardingSphere-Proxy. They all provide functions of data scale-out, distributed transaction and distributed governance, applicable in a variety of scenarios such as Java isomorphism, heterogeneous languages, and a cloud-native environment.

3.1.1 Independent ShardingSphere-JDBC

ShardingSphere-JDBC is a lightweight Java framework that provides additional services at Java's JDBC layer. With the client connecting directly to the database, it provides services in the form of jar and requires no extra deployment and dependence. It can be considered as an enhanced version of the JDBC driver, which is fully compatible with JDBC and all kinds of ORM frameworks.

- Applicable in any ORM framework based on JDBC, such as JPA, Hibernate, Mybatis, Spring JDBC Template, or direct use of JDBC;
- Support any third-party database connection pool, such as DBCP, C3P0, BoneCP, HikariCP;
- Support any kind of JDBC standard database: MySQL, PostgreSQL, Oracle, SQLServer and any JDBC adapted databases.

	ShardingSphere-JDBC	ShardingSphere-Proxy
Database	Any	MySQL/PostgreSQL
Connections Count Cost	More	Less
Heterogeneous language	Java Only	Any
Performance	Low loss	Relatively High loss
Decentralization	Yes	No
Static entry	No	Yes

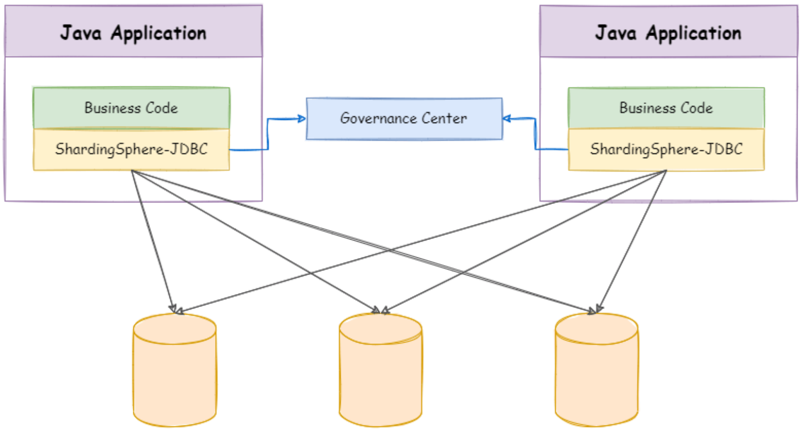


Figure1: ShardingSphere-JDBC Architecture

3.1.2 Independent ShardingSphere-Proxy

ShardingSphere-Proxy is a transparent database proxy, providing a database server that encapsulates database binary protocol to support heterogeneous languages. Currently, MySQL and PostgreSQL protocols are provided. It can use any kind of terminal that is compatible with MySQL or PostgreSQL protocol to operate data, which is more friendly to DBAs.

- Transparent to applications, it can be used directly as MySQL/PostgreSQL;
- Compatible with MySQL-based databases, such as MariaDB, and PostgreSQL-based databases, such as openGauss;
- Applicable to any kind of client that is compatible with MySQL/PostgreSQL protocol, such as MySQL Command Client, MySQL Workbench, etc.

	ShardingSphere-JDBC	ShardingSphere-Proxy
Database	Any	MySQL/PostgreSQL
Connections Count Cost	More	Less
Heterogeneous language	Java Only	Any
Performance	Low loss	Relatively High loss
Decentralization	Yes	No
Static entry	No	Yes

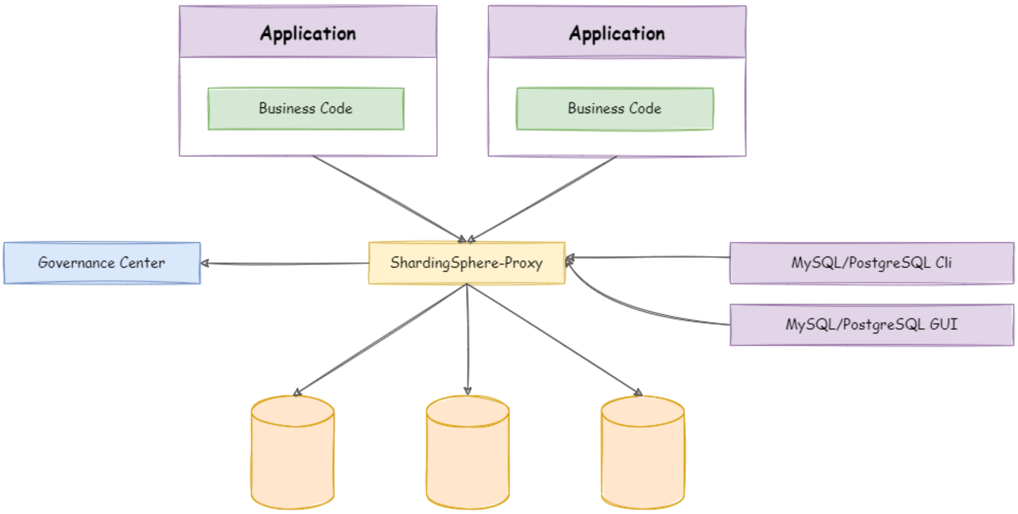


Figure2: ShardingSphere-Proxy Architecture

3.1.3 Hybrid Architecture

ShardingSphere-JDBC adopts a decentralized architecture, applicable to high-performance light-weight OLTP applications developed with Java. ShardingSphere-Proxy provides static entry and supports all languages, applicable to OLAP applications and the sharding databases management and operation situation.

Apache ShardingSphere is an ecosystem composed of multiple access ports. By combining ShardingSphere-JDBC and ShardingSphere-Proxy, and using the same registry to configure sharding strategies, it can flexibly build application systems for various scenarios, allowing architects to freely adjust the system architecture according to the current businesses.

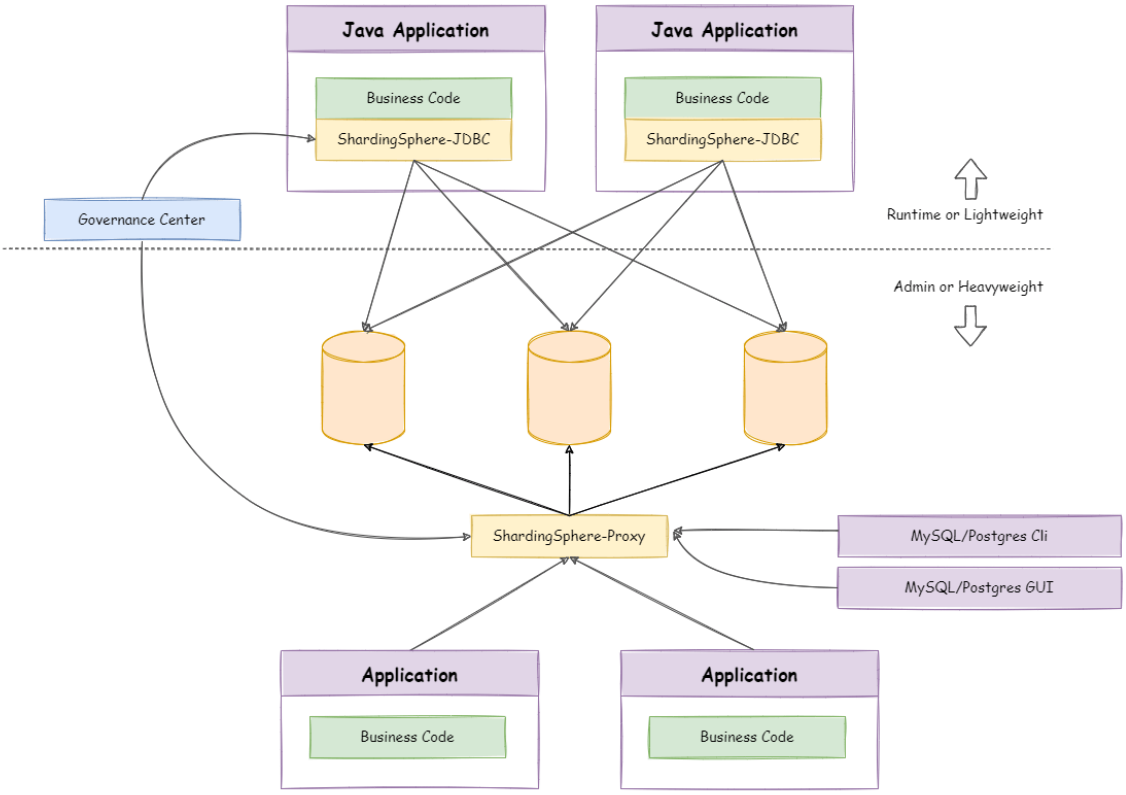


Figure3: ShardingSphere Hybrid Architecture

3.2 Running Modes

Apache ShardingSphere provides two running modes: standalone mode and cluster mode.

3.2.1 Standalone mode

It can achieve data persistence in terms of metadata information such as data sources and rules, but it is not able to synchronize metadata to multiple Apache ShardingSphere instances or be aware of each other in a cluster environment. Updating metadata through one instance causes inconsistencies in other instances because they cannot get the latest metadata.

It is ideal for engineers to build a ShardingSphere environment locally.

3.2.2 Cluster mode

It provides metadata sharing between multiple Apache ShardingSphere instances and the capability to coordinate states in distributed scenarios.

It provides the capabilities necessary for distributed systems, such as horizontal scaling of computing capability and high availability. Clustered environments need to store metadata and coordinate nodes' status through a separately deployed registry center.

We suggest using cluster mode in production environment.

4 Quick Start

In shortest time, this chapter provides users with a simplest quick start with Apache ShardingSphere.

Example Codes: https://github.com/apache/shardingsphere/tree/master/examples

4.1 ShardingSphere-JDBC

4.1.1 Scenarios

There are four ways you can configure Apache ShardingSphere: `Java`, `YAML`, `Spring namespace` and `Spring boot starter`. Developers can choose the preferred method according to their requirements.

4.1.2 Limitations

Currently only Java language is supported.

4.1.3 Requirements

The development environment requires Java JRE 8 or later.

4.1.4 Procedure

1. Rules configuration.

Please refer to User Manual for more details.

2. Import Maven dependency

```
<dependency>
    <groupId>org.apache.shardingsphere</groupId>
    <artifactId>shardingsphere-jdbc-core-spring-boot-starter</artifactId>
```

```
    <version>${latest.release.version}</version>
</dependency>
```

Notice: Please change `${latest.release.version}` to the actual version.

3. Edit `application.yml`.

```
spring:
  shardingsphere:
    datasource:
      names: ds_0, ds_1
      ds_0:
        type: com.zaxxer.hikari.HikariDataSource
        driverClassName: com.mysql.cj.jdbc.Driver
        jdbcUrl: jdbc:mysql://localhost:3306/demo_ds_0?serverTimezone=UTC&
useSSL=false&useUnicode=true&characterEncoding=UTF-8
        username: root
        password:
      ds_1:
        type: com.zaxxer.hikari.HikariDataSource
        driverClassName: com.mysql.cj.jdbc.Driver
        jdbcUrl: jdbc:mysql://localhost:3306/demo_ds_1?serverTimezone=UTC&
useSSL=false&useUnicode=true&characterEncoding=UTF-8
        username: root
        password:
    rules:
      sharding:
        tables:
            ...
```

4.2 ShardingSphere-Proxy

4.2.1 Scenarios

ShardingSphere-Proxy is positioned as a transparent database proxy. It theoretically supports any client operation data using MySQL, PostgreSQL and openGauss protocols, and is friendly to heterogeneous languages and operation and maintenance scenarios.

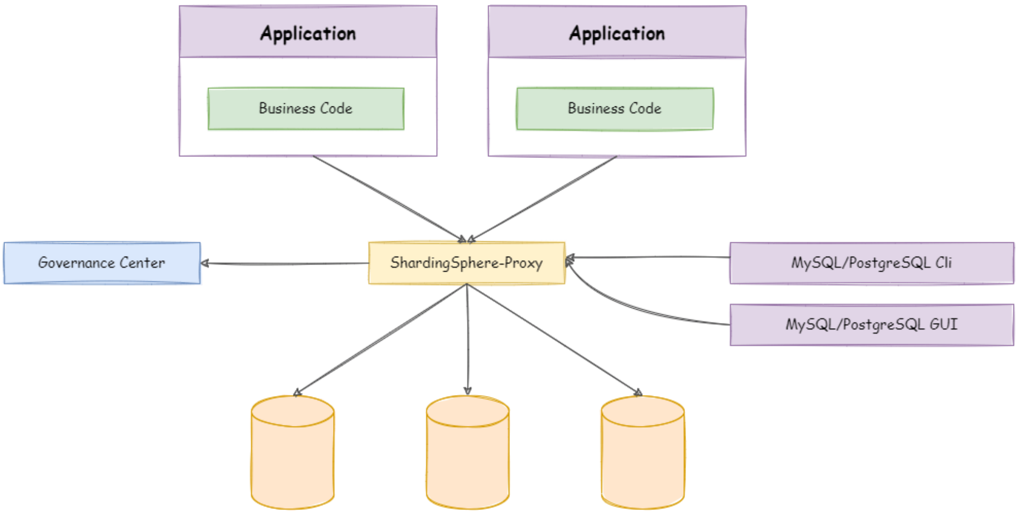


Figure1: shardingsphere-proxy

4.2.2 Limitations

Proxy provides limited support for system databases / tables (such as information_schema, pg_catalog). When connecting to Proxy through some graph database clients, the client or proxy may have an error prompt. You can use command-line clients (`mysql`, `psql`, `gsql`, etc.) to connect to the Proxy's authentication function.

4.2.3 Requirements

Starting ShardingSphere-Proxy with Docker requires no additional dependency. To start the Proxy using binary distribution, the environment must have Java JRE 8 or higher.

4.2.4 Procedure

1. Get ShardingSphere-Proxy.

ShardingSphere-Proxy is available at: - Binary Distribution - Docker - Helm

2. Rule configuration.

Edit `%SHARDINGSPHERE_PROXY_HOME%/conf/server.yaml`.

Edit `%SHARDINGSPHERE_PROXY_HOME%/conf/config-xxx.yaml`.

%SHARDINGSPHERE_PROXY_HOME% is the proxy extract path. for example: /opt/shardingsphere-proxy-bin/

Please refer to Configuration Manual for more details.

3. Import dependencies.

If the backend database is PostgreSQL or openGauss, no additional dependencies are required.

If the backend database is MySQL, please download mysql-connector-java-5.1.47.jar or mysql-connector-java-8.0.11.jar and put it into the %SHARDINGSPHERE_PROXY_HOME%/ext-lib directory.

4. Start server.

- Use the default configuration to start

```
sh %SHARDINGSPHERE_PROXY_HOME%/bin/start.sh
```

The default port is 3307, while the default profile directory is %SHARDINGSPHERE_PROXY_HOME%/conf/.

- Customize port and profile directory

```
sh %SHARDINGSPHERE_PROXY_HOME%/bin/start.sh ${proxy_port} ${proxy_conf_directory}
```

5. Use ShardingSphere-Proxy.

Use MySQL or PostgreSQL or openGauss client to connect ShardingSphere-Proxy.

Use the MySQL client to connect to the ShardingSphere-Proxy:

```
mysql -h${proxy_host} -P${proxy_port} -u${proxy_username} -p${proxy_password}
```

Use the PostgreSQL client to connect to the ShardingSphere-Proxy:

```
psql -h ${proxy_host} -p ${proxy_port} -U ${proxy_username}
```

Use the openGauss client to connect to the ShardingSphere-Proxy:

```
gsql -r -h ${proxy_host} -p ${proxy_port} -U ${proxy_username} -W ${proxy_password}
```

5 Features

Apache ShardingSphere provides a variety of features, from database kernel and database distributed solution to applications closed features.

There is no boundary for these features, warmly welcome more open source engineers to join the community and provide exciting ideas and features.

5.1 Sharding

5.1.1 Background

The traditional solution that stores all the data in one concentrated node has hardly satisfied the requirement of massive data scenario in three aspects, performance, availability and operation cost.

In performance, the relational database mostly uses B+ tree index. When the data amount exceeds the threshold, deeper index will increase the disk IO access number, and thereby, weaken the performance of query. In the same time, high concurrency requests also make the centralized database to be the greatest limitation of the system.

In availability, capacity can be expanded at a relatively low cost and any extent with stateless service, which can make all the pressure, at last, fall on the database. But the single data node or simple primary-replica structure has been harder and harder to take these pressures. Therefore, database availability has become the key to the whole system.

From the aspect of operation costs, when the data in a database instance has reached above the threshold, DBA's operation pressure will also increase. The time cost of data backup and data recovery will be more uncontrollable with increasing amount of data. Generally, it is a relatively reasonable range for the data in single database case to be within 1TB.

Under the circumstance that traditional relational databases cannot satisfy the requirement of the Internet, there are more and more attempts to store the data in native distributed NoSQL. But its incompatibility with SQL and imperfection in ecosystem block it from defeating the relational database in the competition, so the relational database still holds an unshakable position.

Sharding refers to splitting the data in one database and storing them in multiple tables and databases according to some certain standard, so that the performance and availability can be improved. Both methods can effectively avoid the query limitation caused by data exceeding affordable threshold. What' s more, database sharding can also effectively disperse TPS. Table sharding, though cannot ease the database pressure, can provide possibilities to transfer distributed transactions to local transactions, since cross-database upgrades are once involved, distributed transactions can turn pretty tricky sometimes. The use of multiple primary-replica sharding method can effectively avoid the data concentrating on one node and increase the architecture availability.

Splitting data through database sharding and table sharding is an effective method to deal with high TPS and mass amount data system, because it can keep the data amount lower than the threshold and evacuate the traffic. Sharding method can be divided into vertical sharding and horizontal sharding.

Vertical Sharding

According to business sharding method, it is called vertical sharding, or longitudinal sharding, the core concept of which is to specialize databases for different uses. Before sharding, a database consists of many tables corresponding to different businesses. But after sharding, tables are categorized into different databases according to business, and the pressure is also separated into different databases. The diagram below has presented the solution to assign user tables and order tables to different databases by vertical sharding according to business need.

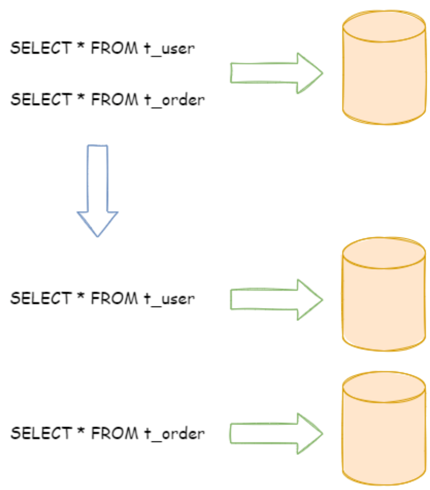


Figure1: Vertical Sharding

Vertical sharding requires to adjust the architecture and design from time to time. Generally speaking, it is not soon enough to deal with fast changing needs from Internet business and not able to really solve the single-node problem. it can ease problems brought by the high data amount and concurrency amount, but cannot solve them completely. After vertical sharding, if the data amount in the table still exceeds the single node threshold, it should be further processed by horizontal sharding.

Horizontal Sharding

Horizontal sharding is also called transverse sharding. Compared with the categorization method according to business logic of vertical sharding, horizontal sharding categorizes data to multiple databases or tables according to some certain rules through certain fields, with each sharding containing only part of the data. For example, according to primary key sharding, even primary keys are put into the 0 database (or table) and odd primary keys are put into the 1 database (or table), which is illustrated as the following diagram.

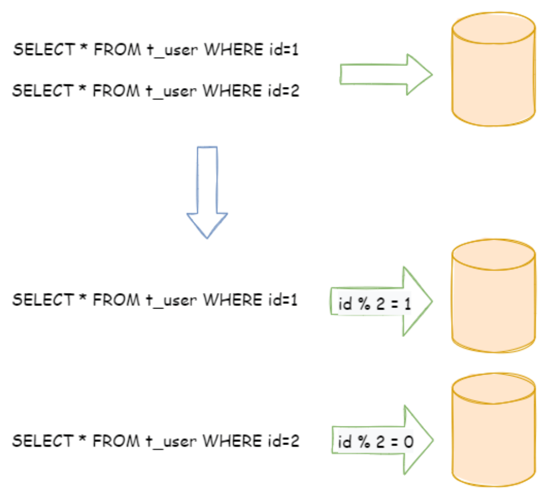


Figure2: Horizontal Sharding

Theoretically, horizontal sharding has overcome the limitation of data processing volume in single machine and can be extended relatively freely, so it can be taken as a standard solution to database sharding and table sharding.

5.1.2 Challenges

Although data sharding solves problems regarding performance, availability, and backup recovery of single points, the distributed architecture has introduced new problems while gaining benefits.

One of the major challenges is that application development engineers and database administrators become extremely overwhelmed with all these operations after such a scattered way of data sharding. They need to know from which specific sub-table can they fetch the data needed.

Another challenge is that SQL that works correctly in one single-node database does not necessarily work correctly in a sharded database. For example, table splitting results in table name changes, or incorrect handling of operations such as paging, sorting, and aggregate grouping.

Cross-library transactions are also tricky for a distributed database cluster. Reasonable use of table splitting can minimize the use of local transactions while reducing the amount of data in a single table, and appropriate use of different tables in the same database can effectively avoid the trouble caused by distributed transactions. In scenarios where cross-library transactions cannot be avoided, some businesses might still be in the need to maintain transaction consistency. The XA-based distributed transactions are not used by Internet giants on a large scale because their performance cannot meet the needs in scenarios with high concurrency, and most of them use flexible transactions with ultimate consistency instead of strong consistent transactions.

5.1.3 Goal

The main design goal of the data sharding modular of Apache ShardingSphere is to try to reduce the influence of sharding, in order to let users use horizontal sharding database group like one database.

5.1.4 Application Scenarios

Mass data high concurrency in OLTP scenarios

Most relational databases use B+ tree indexes, but when the amount of data exceeds the threshold, the increase in index depth will also increase the number of I/O in accessing the disk, which will lower the query performance. Data sharding through ShardingSphere enables data stored in a single database to be dispersed into multiple databases or tables according to a business dimension, which improves performance. The ShardingSphere-JDBC access port can meet the performance requirements of high concurrency in OLTP scenarios.

Mass data real-time analysis in OLAP scenarios

In traditional database architecture, if users want to analyze data, they need to use ETL tools first, synchronize the data to the data platform, and then perform data analysis. However, ETL tools will greatly reduce the effectiveness of data analysis. ShardingSphere-Proxy provides support for static entry and heterogeneous languages, independent of application deployment, which is suitable for real-time analysis in OLAP scenarios.

5.1.5 Related References

- User Guide: sharding
- Developer Guide: sharding

5.1.6 Core Concept

Table

Tables are a key concept for transparent data sharding. Apache ShardingSphere adapts to the data sharding requirements under different scenarios by providing diverse table types.

Logic Table

The logical name of the horizontally sharded database (table) of the same structure is the logical identifier of the table in SQL. Example: Order data is split into 10 tables according to the primary key endings, are `t_order_0` to `t_order_9`, and their logical table names are `t_order`.

Actual Table

Physical tables that exist in the horizontally sharded databases. Those are, `t_order_0` to `t_order_9` in the previous example.

Binding Table

Refers to a set of sharded tables with consistent sharding rules. When using binding tables for multi-table associated query, a sharding key must be used for the association, otherwise, Cartesian product association or cross-library association will occur, affecting query efficiency.

For example, if the `t_order` table and `t_order_item` table are both sharded according to `order_id` and are correlated using `order_id`, the two tables are binding tables. The multi-table associated queries between binding tables will not have a Cartesian product association, so the associated queries will be much more effective. Here is an example,

If SQL is:

```sql
SELECT i.* FROM t_order o JOIN t_order_item i ON o.order_id=i.order_id WHERE o.
order_id in (10, 11);
```

In the case where no binding table relationships are being set, assume that the sharding key `order_id` routes the value 10 to slice 0 and the value 11 to slice 1, then the routed SQL should be 4 items, which are presented as a Cartesian product:

```sql
SELECT i.* FROM t_order_0 o JOIN t_order_item_0 i ON o.order_id=i.order_id WHERE o.
order_id in (10, 11);

SELECT i.* FROM t_order_0 o JOIN t_order_item_1 i ON o.order_id=i.order_id WHERE o.
order_id in (10, 11);

SELECT i.* FROM t_order_1 o JOIN t_order_item_0 i ON o.order_id=i.order_id WHERE o.
order_id in (10, 11);

SELECT i.* FROM t_order_1 o JOIN t_order_item_1 i ON o.order_id=i.order_id WHERE o.
order_id in (10, 11);
```

After the relationships between binding tables are configured and associated with order_id, the routed SQL should then be 2 items:

```sql
SELECT i.* FROM t_order_0 o JOIN t_order_item_0 i ON o.order_id=i.order_id WHERE o.
order_id in (10, 11);

SELECT i.* FROM t_order_1 o JOIN t_order_item_1 i ON o.order_id=i.order_id WHERE o.
order_id in (10, 11);
```

The t_order table will be used by ShardingSphere as the master table for the entire binding table since it specifies the sharding condition. All routing calculations will use only the policy of the primary table, then the sharding calculations for the `t_order_item` table will use the `t_order` condition.

Broadcast data frame

Refers to tables that exist in all sharded data sources. The table structure and its data are identical in each database. Suitable for scenarios where the data volume is small and queries are required to be associated with tables of massive data, e.g., dictionary tables.

Single Table

Refers to the only table that exists in all sharded data sources. Suitable for tables with a small amount of data and do not need to be sharded.

Data Nodes

The smallest unit of the data shard, consists of the data source name and the real table. Example: `ds_0.t_order_0`.

The mapping relationship between the logical table and the real table can be classified into two forms: uniform distribution and custom distribution.

Uniform Distribution

refers to situations where the data table exhibits a uniform distribution within each data source. For example:

```
db0
  ├── t_order0
  └── t_order1
db1
  ├── t_order0
  └── t_order1
```

The configuration of data nodes:

```
db0.t_order0, db0.t_order1, db1.t_order0, db1.t_order1
```

Customized Distribution

Data table exhibiting a patterned distribution. For example:

```
db0
  ├── t_order0
  └── t_order1
db1
  ├── t_order2
  ├── t_order3
  └── t_order4
```

configuration of data nodes:

```
db0.t_order0, db0.t_order1, db1.t_order2, db1.t_order3, db1.t_order4
```

Sharding

Sharding key

A database field is used to split a database (table) horizontally. Example: If the order primary key in the order table is sharded by modulo, the order primary key is a sharded field. If there is no sharded field in SQL, full routing will be executed, of which performance is poor. In addition to the support for single-sharding fields, Apache ShardingSphere also supports sharding based on multiple fields.

Sharding Algorithm

Algorithm for sharding data, supporting `=`, `>=`, `<=`, `>`, `<`, `BETWEEN` and `IN`. The sharding algorithm can be implemented by the developers themselves or can use the Apache ShardingSphere built-in sharding algorithm, syntax sugar, which is very flexible.

Automatic Sharding Algorithm

Sharding algorithm—syntactic sugar is for conveniently hosting all data nodes without users having to concern themselves with the physical distribution of actual tables. Includes implementations of common sharding algorithms such as modulo, hash, range, and time.

Customized Sharding Algorithm

Provides a portal for application developers to implement their sharding algorithms that are closely related to their business operations, while allowing users to manage the physical distribution of actual tables themselves. Customized sharding algorithms are further divided into: - Standard Sharding Algorithm Used to deal with scenarios where sharding is performed using a single key as the sharding key `=`, `IN`, `BETWEEN AND`, `>`, `<`, `>=`, `<=`. - Composite Sharding Algorithm Used to cope with scenarios where multiple keys are used as sharding keys. The logic containing multiple sharding keys is very complicated and requires the application developers to handle it on their own. - Hint Sharding Algorithm For scenarios involving Hint sharding.

Sharding Strategy

Consisting of a sharding key and sharding algorithm, which is abstracted independently due to the independence of the sharding algorithm. What is viable for sharding operations is the sharding key + sharding algorithm, known as sharding strategy.

Mandatory Sharding routing

For the scenario where the sharded field is not determined by SQL but by other external conditions, you can use SQL Hint to inject the shard value. Example: Conduct database sharding by employee login primary key, but there is no such field in the database. SQL Hint can be used both via Java API and SQL annotation. See Mandatory Sharding Routing for details.

Row Value Expressions

Row expressions are designed to address the two main issues of configuration simplification and integration. In the cumbersome configuration rules of data sharding, the large number of repetitive configurations makes the configuration itself difficult to maintain as the number of data nodes increases. The data node configuration workload can be effectively simplified by row expressions.

For the common sharding algorithm, using Java code implementation does not help to manage the configuration uniformly. But by writing the sharding algorithm through line expressions, the rule configuration can be effectively stored together, which is easier to browse and store.

Row expressions are very intuitive, just use `${ expression }` or `$->{ expression }` in the configuration to identify the row expressions. Data nodes and sharding algorithms are currently supported. The content of row expressions uses Groovy syntax, and all operations supported by Groovy are supported by row expressions. For example:

`${begin..end}` denotes the range interval

`${[unit1, unit2, unit_x]}` denotes the enumeration value

If there are multiple `${ expression }` or `$->{ expression }` expressions in a row expression, the final result of the whole expression will be a Cartesian combination based on the result of each sub-expression.

e.g. The following row expression:

```
${['online', 'offline']}_table${1..3}
```

Finally, it can be parsed as this:

```
online_table1, online_table2, online_table3, offline_table1, offline_table2,
offline_table3
```

Distributed Primary Key

In traditional database software development, automatic primary key generation is a basic requirement. Various databases provide support for this requirement, such as self-incrementing keys of MySQL, self-incrementing sequences of Oracle, etc. After data sharding, it is very tricky to generate global unique primary keys for different data nodes. Self-incrementing keys between different actual tables within the same logical table generate repetitive primary keys because they are not mutually aware. Although collisions can be avoided by constraining the initial value and step size of self-incrementing primary keys, additional operational and maintenance rules are necessary to be introduced, rendering the solution lacking in completeness and scalability.

Many third-party solutions can perfectly solve this problem, such as UUID, which relies on specific algorithms to self-generate non-repeating keys, or by introducing primary key generation services. To facilitate users and meet their demands for different scenarios, Apache ShardingSphere not only provides built-in distributed primary key generators, such as UUID and SNOWFLAKE but also abstracts the interface of distributed primary key generators to enable users to implement their own customized self-extending primary key generators.

5.1.7 Limitations

Compatible with all commonly used SQL that routes to single data nodes; SQL routing to multiple data nodes is divided, because of complexity issues, into three conditions: stable support, experimental support, and no support.

Stable Support

Full support for DML, DDL, DCL, TCL, and common DALs. Support for complex queries such as paging, de-duplication, sorting, grouping, aggregation, table association, etc. Support SCHEMA DDL and DML statements of PostgreSQL and openGauss database.

Normal Queries

- main statement SELECT

```
SELECT select_expr [, select_expr ...] FROM table_reference [, table_reference ...]
[WHERE predicates]
[GROUP BY {col_name | position} [ASC | DESC], ...]
[ORDER BY {col_name | position} [ASC | DESC], ...]
[LIMIT {[offset,] row_count | row_count OFFSET offset}]
```

- select_expr

```
* |
[DISTINCT] COLUMN_NAME [AS] [alias] |
(MAX | MIN | SUM | AVG)(COLUMN_NAME | alias) [AS] [alias] |
COUNT(* | COLUMN_NAME | alias) [AS] [alias]
```

- table_reference

```
tbl_name [AS] alias] [index_hint_list]
| table_reference ([INNER] | {LEFT|RIGHT} [OUTER]) JOIN table_factor [JOIN ON
conditional_expr | USING (column_list)]
```

Sub-query

Stable support is provided by the kernel when both the subquery and the outer query specify a shard key and the values of the slice key remain consistent. e.g:

```
SELECT * FROM (SELECT * FROM t_order WHERE order_id = 1) o WHERE o.order_id = 1;
```

Sub-query for pagination can be stably supported by the kernel. e.g.:

```
SELECT * FROM (SELECT row_.*, rownum rownum_ FROM (SELECT * FROM t_order) row_
WHERE rownum <= ?) WHERE rownum > ?;
```

Pagination Query

MySQL, PostgreSQL, and openGauss are fully supported, Oracle and SQLServer are only partially supported due to more intricate paging queries.

Pagination for Oracle and SQLServer needs to be handled by subqueries, and ShardingSphere supports paging-related subqueries.

- Oracle Support pagination by rownum

```
SELECT * FROM (SELECT row_.*, rownum rownum_ FROM (SELECT o.order_id as order_id
FROM t_order o JOIN t_order_item i ON o.order_id = i.order_id) row_ WHERE rownum <=
?) WHERE rownum > ?
```

- SQL Server Support pagination that coordinates TOP + ROW_NUMBER() OVER

```
SELECT * FROM (SELECT TOP (?) ROW_NUMBER() OVER (ORDER BY o.order_id DESC) AS
rownum, * FROM t_order o) AS temp WHERE temp.rownum > ? ORDER BY temp.order_id
```

Support pagination by OFFSET FETCH after SQLServer 2012

```
SELECT * FROM t_order o ORDER BY id OFFSET ? ROW FETCH NEXT ? ROWS ONLY
```

- MySQL, PostgreSQL and openGauss all support LIMIT pagination without the need for sub-query:

```
SELECT * FROM t_order o ORDER BY id LIMIT ? OFFSET ?
```

Shard keys included in operation expressions

When the sharding key is contained in an expression, the value used for sharding cannot be extracted through the SQL letters and will result in full routing.

For example, assume `create_time` is a sharding key.

```
SELECT * FROM t_order WHERE to_date(create_time, 'yyyy-mm-dd') = '2019-01-01';
```

Experimental Support

Experimental support refers specifically to support provided by implementing Federation execution engine, an experimental product that is still under development. Although largely available to users, it still requires significant optimization.

Sub-query

The Federation execution engine provides support for subqueries and outer queries that do not both specify a sharding key or have inconsistent values for the sharding key.

e.g:

```
SELECT * FROM (SELECT * FROM t_order) o;

SELECT * FROM (SELECT * FROM t_order) o WHERE o.order_id = 1;

SELECT * FROM (SELECT * FROM t_order WHERE order_id = 1) o;

SELECT * FROM (SELECT * FROM t_order WHERE order_id = 1) o WHERE o.order_id = 2;
```

Cross-database Associated query

When multiple tables in an associated query are distributed across different database instances, the Federation execution engine can provide support. Assuming that t_order and t_order_item are sharded tables with multiple data nodes while no binding table rules are configured, and t_user and t_user_role are single tables distributed across different database instances, then the Federation execution engine can support the following common associated queries.

```
SELECT * FROM t_order o INNER JOIN t_order_item i ON o.order_id = i.order_id WHERE
o.order_id = 1;

SELECT * FROM t_order o INNER JOIN t_user u ON o.user_id = u.user_id WHERE o.user_
id = 1;

SELECT * FROM t_order o LEFT JOIN t_user_role r ON o.user_id = r.user_id WHERE o.
user_id = 1;

SELECT * FROM t_order_item i LEFT JOIN t_user u ON i.user_id = u.user_id WHERE i.
user_id = 1;

SELECT * FROM t_order_item i RIGHT JOIN t_user_role r ON i.user_id = r.user_id
WHERE i.user_id = 1;

SELECT * FROM t_user u RIGHT JOIN t_user_role r ON u.user_id = r.user_id WHERE u.
user_id = 1;
```

Do not Support

CASE WHEN

The following CASE WHEN statements are not supported: - `CASE WHEN` contains sub-query - Logic names are used in `CASE WHEN`(Please use an alias)

Pagination Query

Due to the complexity of paging queries, there are currently some paging queries that are not supported for Oracle and SQLServer, such as: - Oracle The paging method of rownum + BETWEEN is not supported at present

- SQLServer Currently, pagination with WITH xxx AS (SELECT …) is not supported. Since the SQLServer paging statement automatically generated by Hibernate uses the WITH statement, Hibernate-based SQLServer paging is not supported at this moment. Pagination using two TOP + subquery also cannot be supported at this time.

5.1.8 Appendix with SQL operator

Unsupported SQL:

- CASE WHEN contains sub-query
- Logical table names are used in CASE WHEN(Please use an alias)
- INSERT INTO tbl_name (col1, col2, …) SELECT * FROM tbl_name WHERE col3 = ?（The SELECT clause does not support * and the built-in distributed primary key generator）
- REPLACE INTO tbl_name (col1, col2, …) SELECT * FROM tbl_name WHERE col3 = ?（The SELECT clause does not support * and the built-in distributed primary key generator）
- SELECT MAX(tbl_name.col1) FROM tbl_name (If the query column is a function expression, use the table alias instead of the table name）

5.2 Distributed Transaction

5.2.1 Background

Database transactions should satisfy the features of ACID (atomicity, consistency, isolation and durability).

- Atomicity: transactions are executed as a whole, and either all or none is executed.
- Consistency: transactions should ensure that the state of data remains consistent after the transition.
- Isolation: when multiple transactions execute concurrently, the execution of one transaction should not affect the execution of others.

- Durability: when a transaction committed modifies data, the operation will be saved persistently.

In single data node, transactions are only restricted to the access and control of single database resources, called local transactions. Almost all the mature relational databases have provided native support for local transactions. But in distributed application situations based on micro-services, more and more of them require to include multiple accesses to services and the corresponding database resources in the same transaction. As a result, distributed transactions appear.

Though the relational database has provided perfect native ACID support, it can become an obstacle to the system performance under distributed situations. How to make databases satisfy ACID features under distributed situations or find a corresponding substitute solution, is the priority work of distributed transactions.

5.2.2 Challenge

For different application situations, developers need to reasonably weight the performance and the function between all kinds of distributed transactions.

Highly consistent transactions do not have totally the same API and functions as soft transactions, and they cannot switch between each other freely and invisibly. The choice between highly consistent transactions and soft transactions as early as development decision-making phase has sharply increased the design and development cost.

Highly consistent transactions based on XA is relatively easy to use, but is not good at dealing with long transaction and high concurrency situation of the Internet. With a high access cost, soft transactions require developers to transform the application and realize resources lock and backward compensation.

5.2.3 Goal

The main design goal of the distributed transaction modular of Apache ShardingSphere is to integrate existing mature transaction cases to provide an unified distributed transaction interface for local transactions, 2PC transactions and soft transactions; compensate for the deficiencies of current solutions to provide a one-stop distributed transaction solution.

5.2.4 How it works

ShardingSphere provides begin/ commit/rollback traditional transaction interfaces externally, and provides distributed transaction capabilities through LOCAL, XA and BASE modes.

LOCAL Transaction

LOCAL mode is implemented based on ShardingSphere' s proxy database interfaces, that is begin/commit/rolllback. For a logical SQL, ShardingSphere starts transactions on each proxied database with the begin directive, executes the actual SQL, and performs commit/rollback. Since each data node manages its own transactions, there is no coordination and communication between them, and they do not know whether other data node transactions succeed or not. There is no loss in performance, but strong consistency and final consistency cannot be guaranteed.

XA Transaction

XA transaction adopts the concepts including AP(application program), YM(transaction manager) and RM(resource manager) to ensure the strong consistency of distributed transactions. Those concepts are abstracted from DTP mode which is defined by X/OPEN group. Among them, TM and RM use XA protocol to carry out both-way communication, which is realized through two-phase commit. Compared to traditional local transactions, XA transaction adds a preparation stage where the database can also inform the caller whether the transaction can be committed, in addition to passively accepting commit instructions. TM can collect the results of all branch transactions and make atomic commit at the end to ensure the strong consistency of transactions.

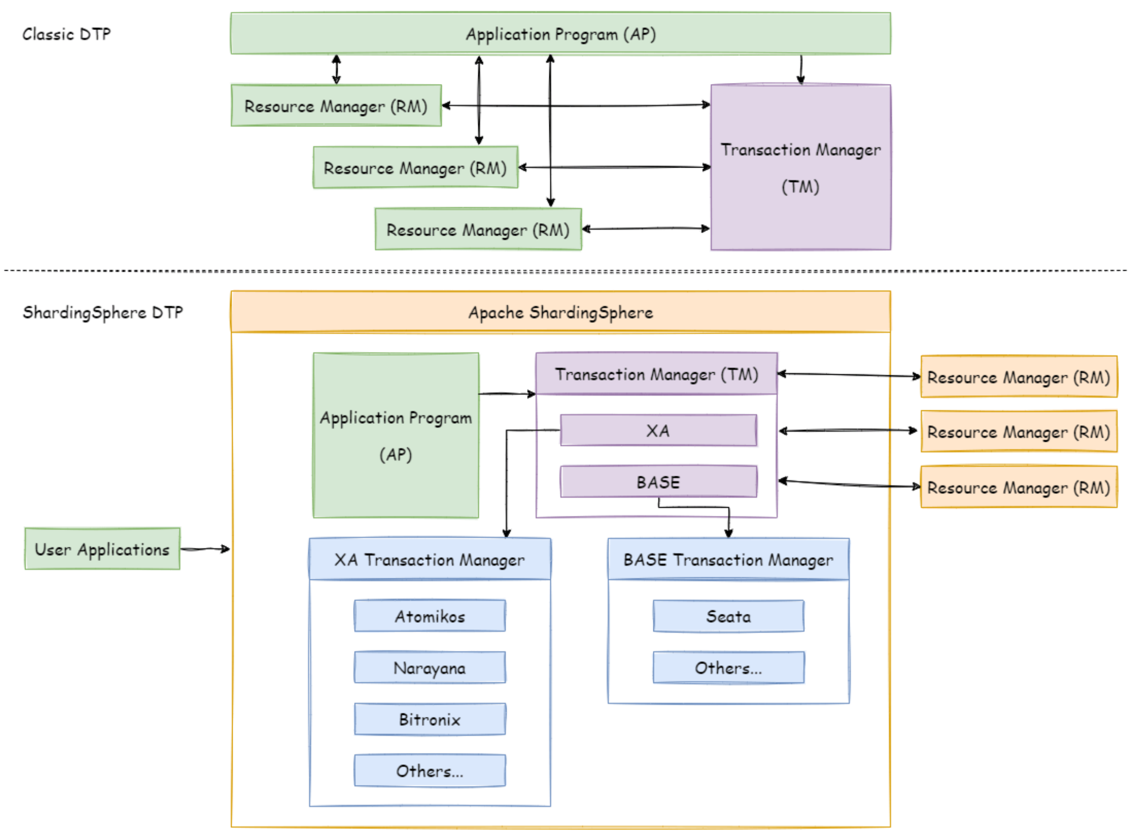


Figure3: Two-phase commit model

XA transaction is implemented based on the interface of ShardingSphere' s proxy database xa start/end/prepare/commit/rollback/recover.

For a logical SQL, ShardingSphere starts transactions in each proxied database with the xa begin directive, integrates TM internally for coordinating branch transactions, and performs xa commit /rollback. Distributed transactions based on XA protocol are more suitable for short transactions with fixed execution time because the required resources need to be locked during execution. For long transactions, data exclusivity during the entire transaction will have an impact on performance in concurrent scenarios.

BASE Transaction

If a transaction that implements ACID is called a rigid transaction, then a transaction based on a BASE transaction element is called a flexible transaction. BASE stands for basic availability, soft state, and eventual consistency.

- Basically Available: ensure that distributed transaction parties are not necessarily online at the same time.
- Soft state: system status updates are allowed to have a certain delay, and the delay may not be recognized by customers.
- Eventually consistent: guarantee the eventual consistency of the system by means of messaging.

ACID transaction puts a high demand for isolation, where all resources must be locked during the execution of transactions. Flexible transaction is to move mutex operations from the resource level to the business level through business logic. Reduce the requirement for strong consistency in exchange for higher system throughput.

ACID-based strong consistency transactions and BASE-based final consistency transactions are not a jack of all trades and can fully leverage their advantages in the most appropriate scenarios. Apache ShardingSphere integrates the operational scheme taking SEATA as the flexible transaction. The following table can be used for comparison to help developers choose the suitable technology.

	LOCAL	*XA*	*BASE*
Business transf ormation	None	None	Seata server needed
Con sistency	Not supported	Not supported	Final consistency
I solation	Not supported	Supported	Business side guaranteed
Co ncurrent per formance	no loss	severe loss	slight loss
Applied s cenarios	Inconsistent processing by the business side	short transaction & low-level concurrency	long transaction & high concurrency

5.2.5 Application Scenarios

The database's transactions can meet ACID business requirements in a standalone application scenario. However, in distributed scenarios, traditional database solutions cannot manage and control global transactions, and users may find data inconsistency on multiple database nodes.

ShardingSphere distributed transaction makes it easier to process distributed transactions and provides flexible and diverse solutions. Users can select the distributed transaction solutions that best fit their business scenarios among LOCAL, XA, and BASE modes.

Application Scenarios for ShardingSphere XA Transactions

Strong data consistency is guaranteed in a distributed environment in terms of XA transactions. However, its performance may be degraded due to the synchronous blocking problem. It applies to business scenarios that require strong data consistency and low concurrency performance.

Application Scenarios for ShardingSphere BASE Transaction

In terms of BASE transactions, final data consistency is guaranteed in a distributed environment. Unlike XA transactions, resources are not locked during the whole transaction process, so its performance is relatively higher.

Application Scenarios for ShardingSphere LOCAL Transaction

In terms of LOCAL transactions, the data consistency and isolation among database nodes are not guaranteed in a distributed environment. Therefore, the business sides need to handle the inconsistencies by themselves. This applies to business scenarios where users would like to handle data inconsistency in a distributed environment by themselves.

5.2.6 Related references

- YAML distributed transaction configuration

5.2.7 Core Concept

XA Protocol

The original distributed transaction model of XA protocol is the "X/Open Distributed Transaction Processing (DTP)" model, XA protocol for short, which was proposed by the X/Open international consortium.

5.2.8 Limitations

Though Apache ShardingSphere intends to be compatible with all distributed scenario and best performance, under CAP theorem guidance, there is no sliver bullet with distributed transaction solution.

Apache ShardingSphere wants to give the user choice of distributed transaction type and use the most suitable solution in different scenarios.

LOCAL Transaction

Supported

- Support none-cross-database transactions. For example, sharding table or sharding database with its route result in same database;
- Support cross-database transactions caused by logic exceptions. For example, update two databases in transaction with exception thrown, data can rollback in both databases.

Unsupported

- Do not support the cross-database transactions caused by network or hardware crash. For example, when update two databases in transaction, if one database crashes before commit, then only the data of the other database can commit.

XA Transaction

Supported

- Support Savepoint;
- PostgreSQL/OpenGauss, in the transaction block, the SQL execution is abnormal, then run `Commit`, transactions are automatically rollback;
- Support cross-database transactions after sharding;
- Operation atomicity and high data consistency in 2PC transactions;
- When service is down and restarted, commit and rollback transactions can be recovered automatically;
- Support use XA and non-XA connection pool together.

Unsupported

- Recover committing and rolling back in other machines after the service is down;
- MySQL,in the transaction block, the SQL execution is abnormal, and run `Commit`, and data remains consistent.

BASE Transaction

Supported

- Support cross-database transactions after sharding;
- Support RC isolation level;
- Rollback transaction according to undo log;
- Support recovery committing transaction automatically after the service is down.

Unsupported

- Do not support other isolation level except RC.

To Be Optimized

- SQL parsed twice by Apache ShardingSphere and SEATA.

5.2.9 Appendix with SQL operator

Unsupported SQL：

- RAL and RDL operations of DistSQL are used in transactions.
- DDL statements are used in XA transactions.

5.3 Readwrite-splitting

5.3.1 Background

Database throughput has faced the bottleneck with increasing TPS. For the application with massive concurrence read but less write in the same time, we can divide the database into a primary database and a replica database. The primary database is responsible for the insert, delete and update of transactions, while the replica database is responsible for queries. It can significantly improve the query performance of the whole system by effectively avoiding row locks.

One primary database with multiple replica databases can further enhance processing capacity by distributing queries evenly into multiple data replicas. Multiple primary databases with multiple replica databases can enhance not only throughput but also availability. Therefore, the system can still run normally, even though any database is down or physical disk destroyed.

Different from the sharding that separates data to all nodes according to sharding keys, readwrite-splitting routes read and write separately to primary database and replica databases according SQL analysis.

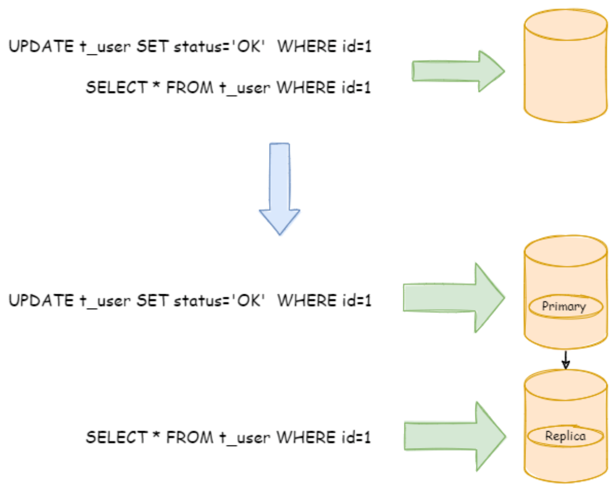


Figure4: background

Data in readwrite-splitting nodes are consistent, whereas that in shards is not. The combined use of sharding and readwrite-splitting will effectively enhance the system performance.

5.3.2 Challenges

Though readwrite-splitting can enhance system throughput and availability, it also brings inconsistent data, including that among multiple primary databases and among primary databases and replica databases. What' s more, it also brings the same problem as data sharding, complicating developer and operator' s maintenance and operation. The following diagram has shown the complex topological relations between applications and database groups when sharding used together with readwrite-splitting.

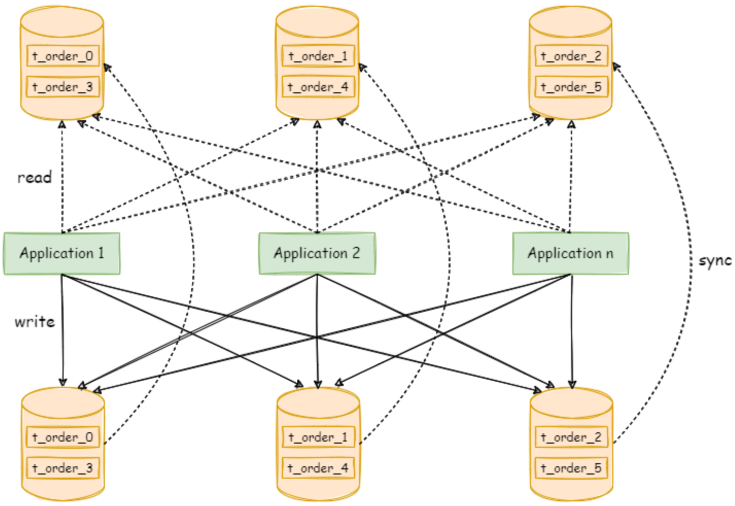


Figure5: challenges

5.3.3 Goal

The main design goal of readwrite-splitting of Apache ShardingSphere is to try to reduce the influence of readwrite-splitting, in order to let users use primary-replica database group like one database.

5.3.4 Application Scenarios

Complex primary-secondary database architecture

Many systems rely on the configuration of primary-secondary database architecture to improve the throughput of the whole system. Nevertheless, this configuration can make it more complex to use services.

After accessing ShardingSphere, the read/write splitting feature can be used to manage primary-secondary databases and achieve transparent read/write splitting, enabling users to use databases with primary/secondary architecture just like using one single database.

5.3.5 Related References

Java API

YAML Configuration

Spring Boot Starter

Spring Namespace

5.3.6 Core Concept

Primary database

The primary database is used to add, update, and delete data operations. Currently, only single primary database is supported.

Secondary database

The secondary database is used to query data operations and multi-secondary databases are supported.

Primary-Secondary synchronization

It refers to the operation of asynchronously synchronizing data from a primary database to a secondary database. Due to the asynchronism of primary-secondary synchronization, data from the primary and secondary databases may be inconsistent for a short time.

Load balancer policy

Channel query requests to different secondary databases through load balancer policy.

5.3.7 Limitations

- Data synchronization of primary and secondary databases is not supported.
- Data inconsistency resulting from data synchronization delays between primary and secondary databases is not supported.
- Multi-write of primary database is not supported.
- Transactional consistency between primary and secondary databases is not supported. In the primary-secondary model, both data reads and writes in transactions use the primary database.

5.4 HA

5.4.1 Background

High availability is the most basic requirement of modern systems. As the cornerstone of the system, the database is also essential for high availability.

In the distributed database system with storage-compute splitting, the high availability solution of storage node and compute node are different. The stateful storage nodes need to pay attention to data consistency, health detection, primary node election and so on; The stateless compute nodes need to detect the changes of storage nodes, they also need to set up an independent load balancer and have the ability of service discovery and request distribution.

Apache ShardingSphere provides compute nodes and reuse database as storage nodes. Therefore, the high availability solution it adopts is to use the high availability solution of the database itself as the high availability of the storage node, and detect the changes automatically.

5.4.2 Challenges

Apache ShardingSphere needs to detect high availability solution of diversified storage nodes automatically, and can also integrate the readwrite splitting dynamically, which is the main challenge of implementation.

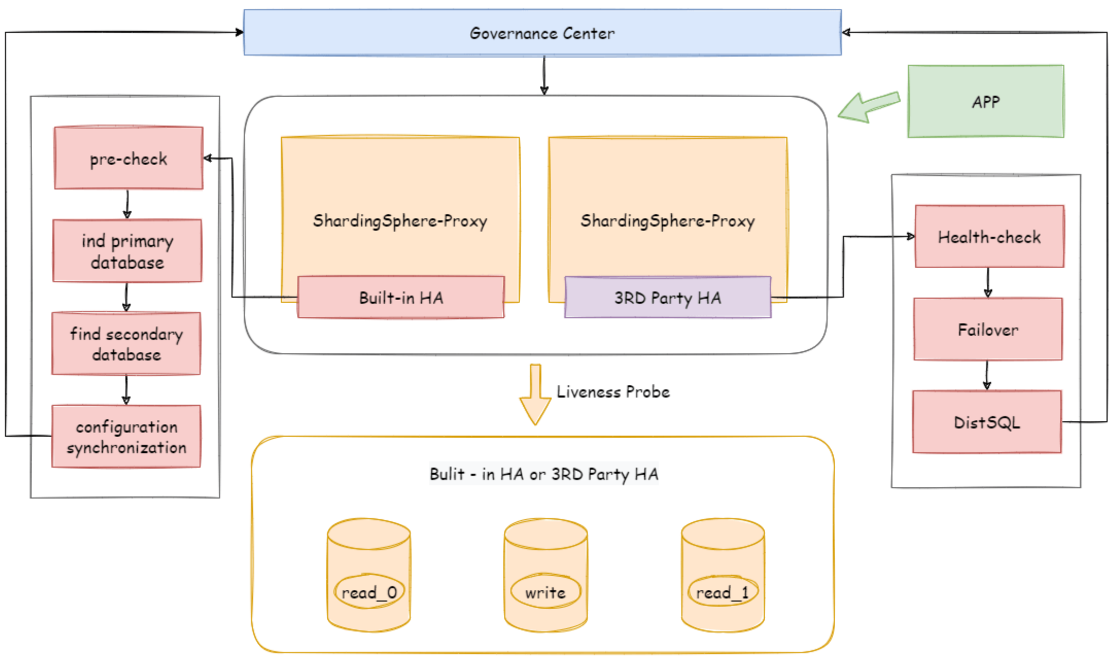


Figure6: Overview

5.4.3 Goal

The main goal of Apache ShardingSphere high availability module which is ensuring 7 * 24-hour uninterrupted database service as much as possible.

5.4.4 Application Scenarios

In most cases, high availability is used in conjunction with read/write splitting. When the relationship between users' write database and read database changes, ShardingSphere dynamically senses and corrects the internal primary/secondary relationship, thus ensuring the correct routing of the read and write traffic. At the same time, when the secondary database breaks down, ShardingSphere can also dynamically correct the state of storage nodes to ensure correct distribution of the read traffic.

5.4.5 Related References

Java API

YAML Configuration

Spring Boot Starter

Spring Namespace

5.4.6 Core Concept

High Availability Type

Apache ShardingSphere does not provide database high availability capability. It senses the change of databases' primary-secondary relationship through a third-party provided high availability solution. Specifically, ShardingSphere is capable of finding databases, automatically sensing the primary/secondary database relationship, and correcting compute nodes' connections to databases.

Dynamic Read/Write Splitting

When high availability and read/write splitting are adopted together, it is not necessary to configure specific primary and secondary databases for read/write splitting. Highly available data sources dynamically correct the primary/secondary relationship of read/write splitting and properly channel read/write traffic.

5.4.7 Limitations

Supported

- MySQL MGR single-primary mode
- MySQL Primary/secondary replication mode
- openGauss Primary/secondary replication mode

Not supported

- MySQL MGR Multi-primary mode

5.5 DB Gateway

5.5.1 Background

With the trend of database fragmentation, using multiple types of databases together has become the norm. The scenario of using one SQL dialect to access all heterogeneous databases is increasing.

5.5.2 Challenges

The existence of diversified databases makes it difficult to standardize the SQL dialect accessing the database. Engineers need to use different dialects for different kinds of databases, and there is no unified query platform.

Automatically translate different types of database dialects into the dialects used by the database, so that engineers can use any database dialect to access all heterogeneous databases, which can reduce development and maintenance cost greatly.

5.5.3 Goal

The goal of database gateway for Apache ShardingSphere is translating SQL automatically among various databases.

5.5.4 Application Scenarios

As business scenarios and database products of enterprises become increasingly diversified, the connection between business applications and various database products becomes extremely complex. ShardingSphere database gateway can shield the connection between business applications and the underlying diversified databases. At the same time, it provides a unified access protocol and syntax system for different business scenarios, which can help enterprises quickly build a unified data access platform.

5.5.5 Core Concept

SQL Dialect

SQL dialect means database dialect, and it indicates that some database projects have their own unique syntax in addition to SQL, which are also called dialects. Different database projects may have different SQL dialects.

5.5.6 Limitations

The SQL dialect translation of Apache ShardingSphere is experimental.

Currently, only MySQL/PostgreSQL dialects can be automatically translated. Engineers can use MySQL dialects and protocols to access PostgreSQL databases and vice versa.

5.6 Traffic Governance

5.6.1 Background

As the scale of data continues to expand, a distributed database has become a trend gradually. The unified management ability of cluster perspective, and control ability of individual components are necessary ability in modern database system.

5.6.2 Challenges

The challenge is ability which are unified management of centralized management, and operation in case of single node in failure.

Centralized management is to uniformly manage the state of database storage nodes and middleware computing nodes, and can detect the latest updates in the distributed environment in real time, further provide information with control and scheduling.

In the overload traffic scenario, circuit breaker and request limiting for a node to ensure whole database cluster can run continuously is a challenge to control ability of a single node.

5.6.3 Goal

The goal of Apache ShardingSphere management module is to realize the integrated management ability from database to computing node, and provide control ability for components in case of failure.

5.6.4 Application Scenarios

Overloaded compute node protection

When a compute node in a ShardingSphere cluster exceeds its load, the circuit breaker function is used to block the traffic to the compute node, to ensure that the whole cluster continues to provide stable services.

Storage node traffic limit

In the read-write splitting scenario where a storage node responsible for the read traffic in a ShardingSphere cluster receives overloaded requests, the traffic limit function is used to block traffic from compute nodes to the storage node, to ensure normal response of the storage node cluster.

5.6.5 Core Concept

Circuit Breaker

Fuse connection between Apache ShardingSphere and the database. When an Apache ShardingSphere node exceeds the max load, stop the node' s access to the database, so that the database can ensure sufficient resources to provide services for other Apache ShardingSphere nodes.

Request Limit

In the face of overload requests, open request limiting to protect some requests can still respond quickly.

5.7 Data Migration

5.7.1 Background

In a scenario where the business continues to develop and the amount of data and concurrency reaches a certain extent, the traditional single database may face problems in terms of performance, scalability and availability.

Although NoSQL solutions can solve the above problems through data sharding and horizontal scale-out, NoSQL databases generally do not support transactions and SQL.

ShardingSphere can also solve the above problems and supports data sharding and horizontal scale-out, while at the same time, also supporting distributed transactions and SQL.

The data migration scheme provided by ShardingSphere can help the traditional single database smoothly switch to ShardingSphere.

5.7.2 Challenges

The data migration process should not affect the running services. So the first challenge is to minimize the time window during which data is not available.

Next, data migration should not affect existing data. So the second challenge is to ensure the data correctness.

5.7.3 Goal

The major goal of Apache ShardingSphere in performing data migration is to reduce the impact of data migration on services and provide a one-stop universal data migration solution.

5.7.4 Application Scenarios

Application scenario one: when an application system is using a traditional single database, and the amount of data in a single table reaches 100 million and is still growing rapidly, a single database that continues to run with a high load will become the bottleneck of the system.

Once the database becomes the bottleneck, it is useless to scale out the application server. Instead, it is the database that needs to be scaled out.

5.7.5 Related References

- Configurations of data migration
- Reference of data migration

5.7.6 Core Concept

Nodes

Instances for running compute or storage tier component processes. These can either be physical machines, virtual machines, or containers, etc.

Cluster

Multiple nodes that are assembled together to provide a specified service.

Source

The storage cluster where the original data resides.

Target

The target storage cluster to which the original data is to be migrated.

Data Migration Process

The entire process of replicating data from one storage cluster to another.

Stock Data

The data that was already in the data node before the data migration operation started.

Incremental Data

New data generated by operational systems during the execution of data migration operations.

5.7.7 Limitations

Procedures Supported

- Migration of peripheral data to databases managed by Apache ShardingSphere.
- Migration of integer or string primary key tables.

Procedures not supported

- Migration without primary key tables.
- Migration of composite primary key tables.
- Migration on top of the current storage node is not supported, so a brand new database cluster needs to be prepared as the migration target cluster.

5.8 Encryption

5.8.1 Background

Security control has always been a crucial link of data governance, data encryption falls into this category. For both Internet enterprises and traditional sectors, data security has always been a highly valued and sensitive topic. Data encryption refers to transforming some sensitive information through encrypt rules to safely protect the private data. Data involves client' s security or business sensibility, such as ID number, phone number, card number, client number and other personal information, requires data encryption according to relevant regulations.

The demand for data encryption is generally divided into two situations in real business scenarios:

1. When the new business start to launch, and the security department stipulates that the sensitive information related to users, such as banks and mobile phone numbers, should be encrypted and stored in the database, and then decrypted when used. Because it is a brand new system, there is no inventory data cleaning problem, so the implementation is relatively simple.

2. For the service has been launched, and plaintext has been stored in the database before. The relevant department suddenly needs to encrypt the data from the on-line business. This scenario generally needs to deal with three issues as followings:

- How to encrypt the historical data, a.k.a.s data clean.
- How to encrypt the newly added data and store it in the database without changing the business SQL and logic; then decrypt the taken out data when use it.
- How to securely, seamlessly and transparently migrate plaintext and ciphertext data between business systems.

5.8.2 Challenges

In the real business scenario, the relevant business development team often needs to implement and maintain a set of encryption and decryption system according to the needs of the company' s security department. When the encryption scenario changes, the encryption system often faces the risk of reconstruction or modification. In addition, for the online business system, it is relatively complex to realize seamless encryption transformation with transparency, security and low risk without modifying the business logic and SQL.

5.8.3 Goal

Provides a security and transparent data encryption solution, which is the main design goal of Apache ShardingSphere data encryption module.

5.8.4 Application Scenarios

Newly launched services

For scenarios requiring the quick launch of new services while respecting encryption regulations. The ShardingSphere encryption feature can be used to quickly achieve compliant data encryption, without requiring users to develop complex encryption systems.

At the same time, its flexibility can also help users avoid complex rebuilding and modification risks caused by encryption scenario changes.

Existing services

For mature services that have already been launched, users need to consider the historical data cleansing and the switchover between old and new features.

By accessing ShardingSphere encrypt, users can easily complete the encryption transformation of the system, and it can also help users securely and quickly switch between old and new features. Users can transparently use encryption and decryption features without changing any business logic and SQL.

5.8.5 Related References

- Configuration: Data Encryption
- Developer Guide: Data Encryption

5.8.6 Core Concept

Logic column

It is used to calculate the encryption and decryption columns and it is the logical identifier of the column in SQL. Logical columns contain ciphertext columns (mandatory), query-helper columns (optional), and plaintext columns (optional).

Cipher column

Encrypted data columns.

Query assistant column

It is a helper column used for queries. For some non-idempotent encryption algorithms with higher security levels, irreversible idempotent columns are provided for queries.

Plain column

The column is used to store plaintext and provide services during the migration of encrypted data. It can be deleted after the data cleansing is complete.

5.8.7 Limitations

- You need to process the original data on stocks in the database by yourself.
- The case-insensitive queries are not supported for encrypted fields.
- Comparison operations are not supported for encrypted fields, such as GREATER THAN, LESS THAN, ORDER BY, BETWEEN, LIKE.
- Calculation operations are not supported for encrypted fields, such as AVG, SUM, and computation expressions.

5.8.8 Appendix with SQL operator

Unsupported SQL：

- The case-insensitive queries are not supported by encrypted fields.
- Comparison operations are not supported for encrypted fields, such as GREATER THAN, LESS THAN, ORDER BY, BETWEEN, LIKE.
- Calculation operations are not supported for encrypted fields, such as AVG, SUM, and computation expressions.

5.9 Shadow

5.9.1 Background

Under the distributed application architecture based on microservices, business requires multiple services to be completed through a series of services and middleware calls. The pressure testing of a single service can no longer reflect the real scenario.

In the test environment, the cost of rebuild complete set of pressure test environment similar to the production environment is too high. It is usually impossible to simulate the complexity and data of the production environment.

So, it is the better way to use the production environment for pressure test. The test results obtained real capacity and performance of the system accurately.

5.9.2 Challenges

pressure testing on production environment is a complex and huge task. Coordination and adjustments between microservices and middlewares required to cope with the transparent transmission of different flow rates and pressure test tags. Usually we will build a complete set of pressure testing platform for different test plans.

Data isolation have to be done at the database-level, in order to ensure the reliability and integrity of the production data, data generated by pressure testing routed to test database. Prevent test data from polluting the real data in the production database.

This requires business applications to perform data classification based on the transparently transmitted pressure test identification before executing SQL, and route the corresponding SQL to the corresponding data source.

5.9.3 Goal

Apache ShardingSphere focuses on data solutions in pressure testing on production environment.

The main goal of the Apache ShardingSphere shadow Database module is routing pressure testing data to user defined database automatically.

5.9.4 Application Scenario

In order to improve the accuracy of stress testing and reduce the testing cost under the distributed application architecture based on microservices, stress testing is usually carried out in production environments, which will notably increase testing risks. However, the ShardingSphere shadow DB function, combined with the flexible configuration of the shadow algorithm, can address data pollution, improve database performance, and meet the requirements of online stress testing in complex business scenarios.

5.9.5 Related References

- Java API: shadow DB
- YAML configuration: shadow DB
- Spring Boot Starter: shadow DB
- Spring Namespace: shadow DB

5.9.6 Core Concept

Production Database

Database for production data

Shadow Database

The Database for stress test data isolation. Configurations should be the same as the Production Database.

Shadow Algorithm

Shadow Algorithm, which is closely related to business operations, currently has 2 types.

- Column based shadow algorithm Routing to shadow database by recognizing data from SQL. Suitable for stress test scenario that has an emphasis on data list.
- Hint based shadow algorithm Routing to shadow database by recognizing comments from SQL. Suitable for stress test driven by the identification of upstream system passage.

5.9.7 Limitations

Hint based shadow algorithm

No

Column based shadow algorithm

Does not support DDL.

Does not support scope, group, subqueries such as BETWEEN, GROUP BY …HAVING, etc.

SQL support list

- INSERT

SQL	*support or not*
INSERT INTO table (column,…) VALUES (value,…)	support
INSERT INTO table (column,…) VALUES (value,…),(value,…),…	support
INSERT INTO table (column,…) SELECT column1 from table1 where column1 = value1	do not support

- SELECT/UPDATE/DELETE

• condition ca tegories*	*SQL*	• support or not*
=	SELECT/UPDATE/DELETE ⋯ WHERE column = value	support
LIKE/NOT LIKE	SELECT/UPDATE/DELETE ⋯ WHERE column LIKE/NOT LIKE value	support
IN/NOT IN	SELECT/UPDATE/DELETE ⋯ WHERE column IN/NOT IN (value1,value2,⋯)	support
BETWEEN	SELECT/UPDATE/DELETE ⋯ WHERE column BETWEEN value1 AND value2	do not support
GROUP BY ⋯HAVING⋯	SELECT/UPDATE/DELETE ⋯ WHERE ⋯GROUP BY column HAVING column > value	do not support
Sub Query	SELECT/UPDATE/DELETE ⋯ WHERE column = (SELECT column FROM table WHERE column = value)	do not support

5.10 Observability

5.10.1 Background

In order to grasp the distributed system status, observe running state of the cluster is a new challenge. The point-to-point operation mode of logging in to a specific server cannot suite to large number of distributed servers. Telemetry through observable data is the recommended operation and maintenance mode for them. Tracking, metrics and logging are important ways to obtain observable data of system status.

APM (application performance monitoring) is to monitor and diagnose the performance of the system by collecting, storing and analyzing the observable data of the system. Its main functions include performance index monitoring, call stack analysis, service topology, etc.

Apache ShardingSphere is not responsible for gathering, storing and demonstrating APM data, but provides the necessary information for the APM. In other words, Apache ShardingSphere is only responsible for generating valuable data and submitting it to relevant systems through standard protocols or plug-ins. Tracing is to obtain the tracking information of SQL parsing and SQL execution. Apache ShardingSphere provides support for SkyWalking, Zipkin, Jaeger and OpenTelemetry by default. It also supports users to develop customized components through plug-in.

- Use Zipkin or Jaeger Just provides correct Zipkin or Jaeger server information in the agent configuration file.

- Use OpenTelemetry OpenTelemetry was merged by OpenTracing and OpenCencus in 2019. In this way, you only need to fill in the appropriate configuration in the agent configuration file according to OpenTelemetry SDK Autoconfigure Guide.
- Use SkyWalking Enable the SkyWalking plug-in in configuration file and need to configure the SkyWalking apm-toolkit.
- Use SkyWalking's automatic monitor probe Cooperating with Apache SkyWalking team, Apache ShardingSphere team has realized ShardingSphere automatic monitor probe to automatically send performance data to SkyWalking. Note that automatic probe in this way cannot be used together with Apache ShardingSphere plug-in probe.

Metrics used to collect and display statistical indicator of cluster. Apache ShardingSphere supports Prometheus by default.

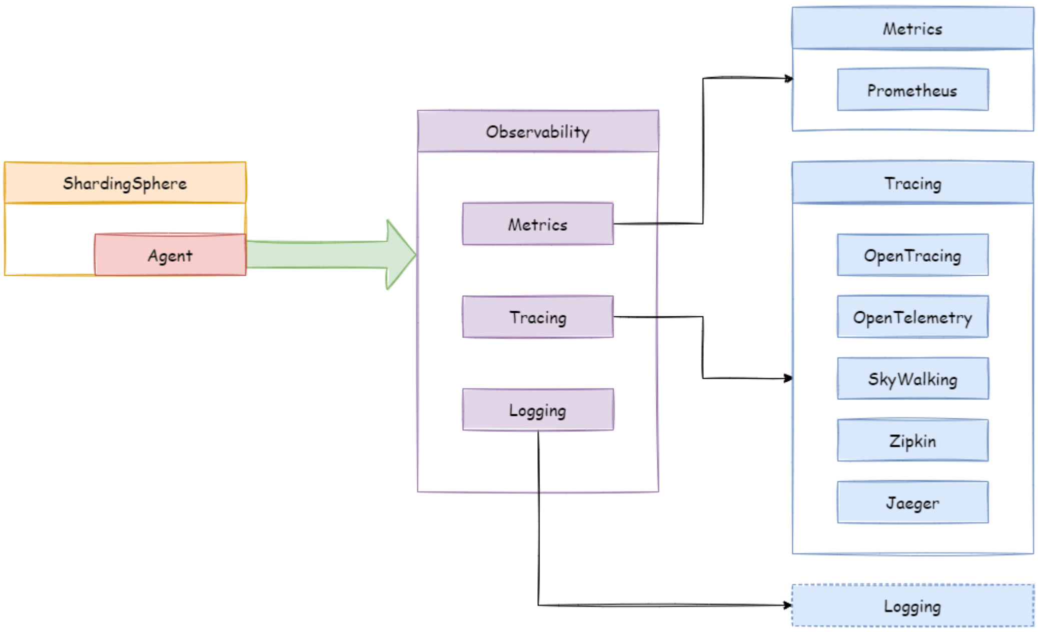


Figure7: Overview

5.10.2 Challenges

Tracing and metrics need to collect system information through event tracking. Lots of events tracking make kernel code mess, difficult to maintain, and difficult to customize extend.

5.10.3 Goal

The goal of Apache ShardingSphere observability module is providing as many performance and statistical indicators as possible and isolating kernel code and embedded code.

5.10.4 Application Scenarios

ShardingSphere provides observability for applications through the Agent module, and this feature applies to the following scenarios:

Monitoring panel

The system' s static information (such as application version) and dynamic information (such as the number of threads and SQL processing information) are exposed to a third-party application (such as Prometheus) using a standard interface. Administrators can visually monitor the real-time system status.

Monitoring application performance

In ShardingSphere, a SQL statement needs to go through the processes of parsing, routing, rewriting, execution, and result merging before it is finally executed and the response can be output. If a SQL statement is complex and the overall execution takes a long time, how do we know which procedure has room for optimization?

Through Agent plus Tracing, administrators can learn about the time consumption of each step of SQL execution. Thus, they can easily locate performance risks and formulate targeted SQL optimization schemes.

Tracing application links

In a distributed application plus data sharding scenario, it is tricky to figure out which node the SQL statement is issued from and which data source the statement is finally executed on. If an exception occurs during SQL execution, how do we locate the node where the exception occurred?

Agent + Tracing can help users solve the above problems.

Through tracing the full link of the SQL execution process, users can get complete information such as "where the SQL comes from and where it is sent to".

They can also visually observe the SQL routing situation through the generated topological graph, make timely responses, and quickly locate the root cause of problems.

5.10.5 Related References

- Usage of observability
- Dev guide: observability
- Implementation

5.10.6 Core Concept

Agent

Based on bytecode enhancement and plugin design to provide tracing, metrics and logging features.

Only after the plugin of the Agent is enabled, the monitoring indicator data can be output to the third-party APM for display.

APM

APM is an acronym for Application Performance Monitoring.

Focusing on the performance diagnosis of distributed systems, its main functions include call chain display, application topology analysis, etc.

Tracing

Tracing data between distributed services or internal processes will be collected by agent. It will then be sent to third-party APM systems.

Metrics

System statistical indicators are collected through probes and written to the time series database for display by third-party applications.

Logging

The log can be easily expanded through the agent to provide more information for analyzing the system running status.

6 User Manual

This chapter describes how to use projects of Apache ShardingSphere.

6.1 ShardingSphere-JDBC

Configuration is the only module in ShardingSphere-JDBC that interacts with application developers, through which developers can quickly and clearly understand the functions provided by ShardingSphere-JDBC.

This chapter is a configuration manual for ShardingSphere-JDBC, which can also be referred to as a dictionary if necessary.

ShardingSphere-JDBC has provided 4 kinds of configuration methods for different situations. By configuration, application developers can flexibly use data sharding, readwrite-splitting, data encryption, shadow database or the combination of them.

Mixed rule configurations are very similar to single rule configuration, except for the differences from single rule to multiple rules.

It should be noted that the superposition between rules are data source and table name related. If the previous rule is data source oriented aggregation, the next rule needs to use the aggregated logical data source name configured by the previous rule when configuring the data source; Similarly, if the previous rule is table oriented aggregation, the next rule needs to use the aggregated logical table name configured by the previous rule when configuring the table.

Please refer to Example for more details.

6.1.1 YAML Configuration

Overview

YAML configuration provides interaction with ShardingSphere JDBC through configuration files. When used with the governance module together, the configuration of persistence in the configuration center is YAML format.

YAML configuration is the most common configuration mode, which can omit the complexity of programming and simplify user configuration.

Usage

Import Maven Dependency

```
<dependency>
    <groupId>org.apache.shardingsphere</groupId>
    <artifactId>shardingsphere-jdbc-core</artifactId>
    <version>${shardingsphere.version}</version>
</dependency>
```

YAML Format

ShardingSphere-JDBC YAML file consists of database name, mode configuration, data source map, rule configurations and properties.

Note: The example connection pool is HikariCP, which can be replaced with other connection pools according to business scenarios.

```
# JDBC logic database name. Through this parameter to connect ShardingSphere-JDBC
and ShardingSphere-Proxy.
# Default value: logic_db
databaseName (?):

mode:

dataSources:

rules:
- !FOO_XXX
    ...
- !BAR_XXX
    ...

props:
  key_1: value_1
  key_2: value_2
```

Please refer to Mode Configuration for more mode details.

Please refer to Data Source Configuration for more data source details.

Please refer to Rules Configuration for more rule details.

Create Data Source

The ShardingSphereDataSource created by YamlShardingSphereDataSourceFactory implements the standard JDBC DataSource interface.

```
File yamlFile = // Indicate YAML file
DataSource dataSource = YamlShardingSphereDataSourceFactory.
createDataSource(yamlFile);
```

Use Data Source

Same with Java API.

YAML Syntax Explanation

!! means instantiation of that class

! means self-defined alias

- means one or multiple can be included

[] means array, can substitutable with - each other

Mode

Parameters

```
mode (?): # Default value is Standalone
  type: # Type of mode configuration. Values could be: Standalone, Cluster
  repository (?): # Persist repository configuration
  overwrite: # Whether overwrite persistent configuration with local configuration
```

Standalone Mode

```yaml
mode:
  type: Standalone
  repository:
    type: # Type of persist repository
    props: # Properties of persist repository
      foo_key: foo_value
      bar_key: bar_value
  overwrite: # Whether overwrite persistent configuration with local configuration
```

Cluster Mode (recommended)

```yaml
mode:
  type: Cluster
  repository:
    type: # Type of persist repository
    props: # Properties of persist repository
      namespace: # Namespace of registry center
      server-lists: # Server lists of registry center
      foo_key: foo_value
      bar_key: bar_value
  overwrite: # Whether overwrite persistent configuration with local configuration
```

Notes

1. Cluster mode deployment is recommended for production environment.
2. The ‘ZooKeeper’ registry center is recommended for cluster mode deployment.

Sample

Standalone Mode

```yaml
mode:
  type: Standalone
  repository:
    type: File
  overwrite: false
```

Cluster Mode (recommended)

```yaml
mode:
  type: Cluster
  repository:
    type: ZooKeeper
    props:
      namespace: governance
      server-lists: localhost:2181
      retryIntervalMilliseconds: 500
      timeToLiveSeconds: 60
  overwrite: false
```

Related References

- Installation and Usage of ZooKeeper Registry Center
- Please refer to Builtin Persist Repository List for more details about the type of repository.

Data Source

Background

ShardingSphere-JDBC Supports all JDBC drivers and database connection pools.

In this example, the database driver is MySQL, and the connection pool is HikariCP, which can be replaced with other database drivers and connection pools. When using ShardingSphere JDBC, the property name of the JDBC pool depends on the definition of the respective JDBC pool and is not defined by ShardingSphere. For related processing, please refer to the class org.apache.shardingsphere.infra.datasource.pool.creator.DataSourcePoolCreator. For example, with Alibaba Druid 1.2.9, using url instead of jdbcUrl in the example below is the expected behavior.

Parameters

```yaml
dataSources: # Data sources configuration, multiple <data-source-name> available
  <data-source-name>: # Data source name
    dataSourceClassName: # Data source class name
    driverClassName: # The database driver class name is subject to the configuration of the database connection pool itself
    jdbcUrl: # The database URL connection is subject to the configuration of the database connection pool itself
    username: # Database user name, subject to the configuration of the database connection pool itself
    password: # The database password is subject to the configuration of the database connection pool itself
    # ... Other properties of data source pool
```

```
```

Sample

```yaml
dataSources:
  ds_1:
    dataSourceClassName: com.zaxxer.hikari.HikariDataSource
    driverClassName: com.mysql.jdbc.Driver
    jdbcUrl: jdbc:mysql://localhost:3306/ds_1
    username: root
    password:
  ds_2:
    dataSourceClassName: com.zaxxer.hikari.HikariDataSource
    driverClassName: com.mysql.jdbc.Driver
    jdbcUrl: jdbc:mysql://localhost:3306/ds_2
    username: root
    password:

  # Configure other data sources
```

Rules

Rules are pluggable part of Apache ShardingSphere. This chapter is a YAML rule configuration manual for ShardingSphere-JDBC.

Sharding

Background

Data sharding YAML configuration is highly readable. The dependencies between sharding rules can be quickly understood through the YAML format. ShardingSphere automatically creates the ShardingSphereDataSource object according to YAML configuration, which can reduce unnecessary coding for users.

Parameters

```yaml
rules:
- !SHARDING
  tables: # Sharding table configuration
    <logic-table-name> (+): # Logic table name
      actualDataNodes (?): # Describe data source names and actual tables (refer to
Inline syntax rules)
      databaseStrategy (?): # Databases sharding strategy, use default databases
sharding strategy if absent. sharding strategy below can choose only one.
```

```yaml
        standard: # For single sharding column scenario
          shardingColumn: # Sharding column name
          shardingAlgorithmName: # Sharding algorithm name
        complex: # For multiple sharding columns scenario
          shardingColumns: # Sharding column names, multiple columns separated with comma
          shardingAlgorithmName: # Sharding algorithm name
        hint: # Sharding by hint
          shardingAlgorithmName: # Sharding algorithm name
        none: # Do not sharding
      tableStrategy: # Tables sharding strategy, same as database sharding strategy
      keyGenerateStrategy: # Key generator strategy
        column: # Column name of key generator
        keyGeneratorName: # Key generator name
  autoTables: # Auto Sharding table configuration
    t_order_auto: # Logic table name
      actualDataSources (?): # Data source names
      shardingStrategy: # Sharding strategy
        standard: # For single sharding column scenario
          shardingColumn: # Sharding column name
          shardingAlgorithmName: # Auto sharding algorithm name
  bindingTables (+): # Binding tables
    - <logic_table_name_1, logic_table_name_2, ...>
    - <logic_table_name_1, logic_table_name_2, ...>
  broadcastTables (+): # Broadcast tables
    - <table-name>
    - <table-name>
  defaultDatabaseStrategy: # Default strategy for database sharding
  defaultTableStrategy: # Default strategy for table sharding
  defaultKeyGenerateStrategy: # Default Key generator strategy
  defaultShardingColumn: # Default sharding column name

  # Sharding algorithm configuration
  shardingAlgorithms:
    <sharding-algorithm-name> (+): # Sharding algorithm name
      type: # Sharding algorithm type
      props: # Sharding algorithm properties
      # ...

  # Key generate algorithm configuration
  keyGenerators:
    <key-generate-algorithm-name> (+): # Key generate algorithm name
      type: # Key generate algorithm type
      props: # Key generate algorithm properties
      # ...
```

Procedure

1. Configure data sharding rules in YAML files, including data source, sharding rules, and global attributes and other configuration items.
2. Call createDataSource method of the object YamlShardingSphereDataSourceFactory. Create ShardingSphereDataSource according to the configuration information in YAML files.

Sample

The YAML configuration sample of data sharding is as follows:

```yaml
dataSources:
  ds_0:
    dataSourceClassName: com.zaxxer.hikari.HikariDataSource
    driverClassName: com.mysql.jdbc.Driver
    jdbcUrl: jdbc:mysql://localhost:3306/demo_ds_0?serverTimezone=UTC&useSSL=false&
useUnicode=true&characterEncoding=UTF-8
    username: root
    password:
  ds_1:
    dataSourceClassName: com.zaxxer.hikari.HikariDataSource
    driverClassName: com.mysql.jdbc.Driver
    jdbcUrl: jdbc:mysql://localhost:3306/demo_ds_1?serverTimezone=UTC&useSSL=false&
useUnicode=true&characterEncoding=UTF-8
    username: root
    password:

rules:
- !SHARDING
  tables:
    t_order:
      actualDataNodes: ds_${0..1}.t_order_${0..1}
      tableStrategy:
        standard:
          shardingColumn: order_id
          shardingAlgorithmName: t-order-inline
      keyGenerateStrategy:
        column: order_id
        keyGeneratorName: snowflake
    t_order_item:
      actualDataNodes: ds_${0..1}.t_order_item_${0..1}
      tableStrategy:
        standard:
          shardingColumn: order_id
          shardingAlgorithmName: t_order-item-inline
      keyGenerateStrategy:
        column: order_item_id
```

```yaml
        keyGeneratorName: snowflake
    t_account:
      actualDataNodes: ds_${0..1}.t_account_${0..1}
      tableStrategy:
        standard:
          shardingAlgorithmName: t-account-inline
      keyGenerateStrategy:
        column: account_id
        keyGeneratorName: snowflake
  defaultShardingColumn: account_id
  bindingTables:
    - t_order,t_order_item
  broadcastTables:
    - t_address
  defaultDatabaseStrategy:
    standard:
      shardingColumn: user_id
      shardingAlgorithmName: database-inline
  defaultTableStrategy:
    none:

  shardingAlgorithms:
    database-inline:
      type: INLINE
      props:
        algorithm-expression: ds_${user_id % 2}
    t-order-inline:
      type: INLINE
      props:
        algorithm-expression: t_order_${order_id % 2}
    t_order-item-inline:
      type: INLINE
      props:
        algorithm-expression: t_order_item_${order_id % 2}
    t-account-inline:
      type: INLINE
      props:
        algorithm-expression: t_account_${account_id % 2}
  keyGenerators:
    snowflake:
      type: SNOWFLAKE

props:
  sql-show: false
```

Read the YAML configuration to create a data source according to the createDataSource method of YamlShardingSphereDataSourceFactory.

```
YamlShardingSphereDataSourceFactory.createDataSource(getFile("/META-INF/sharding-
databases-tables.yaml"));
```

Related References

- Core Feature: Data Sharding
- Developer Guide: Data Sharding

Readwrite-splitting

Background

Read/write splitting YAML configuration is highly readable. The YAML format enables you to quickly understand the dependencies between read/write sharding rules. ShardingSphere automatically creates the ShardingSphereDataSource object according to the YAML configuration, which reduces unnecessary coding for users.

Parameters

Static Readwrite-splitting

```
rules:
- !READWRITE_SPLITTING
  dataSources:
    <data-source-name> (+): # Logic data source name of readwrite-splitting
      static-strategy: # Readwrite-splitting type
        write-data-source-name: # Write data source name
        read-data-source-names: # Read data source names, multiple data source
names separated with comma
      loadBalancerName: # Load balance algorithm name

  # Load balance algorithm configuration
  loadBalancers:
    <load-balancer-name> (+): # Load balance algorithm name
      type: # Load balance algorithm type
      props: # Load balance algorithm properties
        # ...
```

Dynamic Readwrite-splitting

```yaml
rules:
- !READWRITE_SPLITTING
  dataSources:
    <data-source-name> (+): # Logic data source name of readwrite-splitting
      dynamic-strategy: # Readwrite-splitting type
        auto-aware-data-source-name: # Database discovery logic data source name
        write-data-source-query-enabled: # All read data source are offline, write
data source whether the data source is responsible for read traffic
      loadBalancerName: # Load balance algorithm name

  # Load balance algorithm configuration
  loadBalancers:
    <load-balancer-name> (+): # Load balance algorithm name
      type: # Load balance algorithm type
      props: # Load balance algorithm properties
        # ...
```

Please refer to Built-in Load Balance Algorithm List for more details about type of algorithm. Please refer to Read-write splitting-Core features for more details about query consistent routing.

Procedure

1. Add read/write splitting data source.
2. Set the load balancer algorithm.
3. Use read/write data source.

Sample

```yaml
rules:
- !READWRITE_SPLITTING
  dataSources:
    readwrite_ds:
      staticStrategy:
        writeDataSourceName: write_ds
        readDataSourceNames:
          - read_ds_0
          - read_ds_1
      loadBalancerName: random
  loadBalancers:
    random:
      type: RANDOM
```

Related References

- Read-write splitting-Core features
- Java API: read-write splitting
- Spring Boot Starter: read-write splitting
- Spring namespace: read-write splitting

Distributed Transaction

Background

ShardingSphere provides three modes for distributed transactions `LOCAL`, `XA`, `BASE`.

Parameters

```yaml
rules:
  - !TRANSACTION
    defaultType: # Transaction mode, optional value LOCAL/XA/BASE
    providerType: # Specific implementation of the mode
```

Procedure

Use LOCAL Mode

The content of the server.yaml configuration file is as follows:

```yaml
rules:
  - !TRANSACTION
    defaultType: LOCAL
```

Use XA Mode

The content of the server.yaml configuration file is as follows:

```yaml
rules:
  - !TRANSACTION
    defaultType: XA
    providerType: Narayana/Atomikos
```

To manually add Narayana-related dependencies:

```
jta-5.12.4.Final.jar
arjuna-5.12.4.Final.jar
common-5.12.4.Final.jar
jboss-connector-api_1.7_spec-1.0.0.Final.jar
jboss-logging-3.2.1.Final.jar
jboss-transaction-api_1.2_spec-1.0.0.Alpha3.jar
jboss-transaction-spi-7.6.0.Final.jar
narayana-jts-integration-5.12.4.Final.jar
shardingsphere-transaction-xa-narayana-x.x.x-SNAPSHOT.jar
```

Use BASE Mode

The content of the server.yaml configuration file is as follows:

```yaml
rules:
  - !TRANSACTION
    defaultType: BASE
    providerType: Seata
```

Build a Seata Server, add relevant configuration files and Seata dependencies, see ShardingSphere Integrates Seata Flexible Transactions

HA

Background

Through `YAML` format, ShardingSphere will automatically create the `ShardingSphereDataSource` object according to the YAML configuration, reducing unnecessary coding work for users.

Parameters

```yaml
rules:
- !READWRITE_SPLITTING
  dataSources:
    replica_ds:
      dynamicStrategy:
        autoAwareDataSourceName: # High availability rule logical data source name

- !DB_DISCOVERY
  dataSources:
    <data-source-name> (+): # Logic data source name
      dataSourceNames: # Data source names
        - <data-source>
        - <data-source>
      discoveryHeartbeatName: # Detect heartbeat name
```

```yaml
      discoveryTypeName: # Database discovery type name

  # Heartbeat Configuration
  discoveryHeartbeats:
    <discovery-heartbeat-name> (+): # heartbeat name
      props:
        keep-alive-cron: # This is cron expression, such as: '0/5 * * * * ?'

  # Database Discovery Configuration
  discoveryTypes:
    <discovery-type-name> (+): # Database discovery type name
      type: # Database discovery type, such as: MySQL.MGR
      props (?):
        group-name: 92504d5b-6dec-11e8-91ea-246e9612aaf1 # Required parameters for
database discovery types, such as MGR's group-name
```

Sample

```yaml
databaseName: database_discovery_db

dataSources:
  ds_0:
    url: jdbc:mysql://127.0.0.1:33306/primary_demo_ds?serverTimezone=UTC&
useSSL=false
    username: root
    password:
    connectionTimeoutMilliseconds: 3000
    idleTimeoutMilliseconds: 60000
    maxLifetimeMilliseconds: 1800000
    maxPoolSize: 50
    minPoolSize: 1
  ds_1:
    url: jdbc:mysql://127.0.0.1:33307/primary_demo_ds?serverTimezone=UTC&
useSSL=false
    username: root
    password:
    connectionTimeoutMilliseconds: 3000
    idleTimeoutMilliseconds: 60000
    maxLifetimeMilliseconds: 1800000
    maxPoolSize: 50
    minPoolSize: 1
  ds_2:
    url: jdbc:mysql://127.0.0.1:33308/primary_demo_ds?serverTimezone=UTC&
useSSL=false
    username: root
    password:
```

```yaml
    connectionTimeoutMilliseconds: 3000
    idleTimeoutMilliseconds: 60000
    maxLifetimeMilliseconds: 1800000
    maxPoolSize: 50
    minPoolSize: 1

rules:
  - !READWRITE_SPLITTING
    dataSources:
      replica_ds:
        dynamicStrategy:
          autoAwareDataSourceName: readwrite_ds
  - !DB_DISCOVERY
    dataSources:
      readwrite_ds:
        dataSourceNames:
          - ds_0
          - ds_1
          - ds_2
        discoveryHeartbeatName: mgr-heartbeat
        discoveryTypeName: mgr
    discoveryHeartbeats:
      mgr-heartbeat:
        props:
          keep-alive-cron: '0/5 * * * * ?'
    discoveryTypes:
      mgr:
        type: MySQL.MGR
        props:
          group-name: 558edd3c-02ec-11ea-9bb3-080027e39bd2
```

Related References

- Feature Description of HA
- JAVA API: HA
- Spring Boot Starter: HA
- Spring Namespace: HA

Encryption

Background

The YAML configuration approach to data encryption is highly readable, with the YAML format enabling a quick understanding of dependencies between encryption rules. Based on the YAML configuration, ShardingSphere automatically completes the creation of ShardingSphereDataSource objects, reducing unnecessary coding efforts for users.

Parameters

```
rules:
- !ENCRYPT
  tables:
    <table-name> (+): # Encrypt table name
      columns:
        <column-name> (+): # Encrypt logic column name
          cipherColumn: # Cipher column name
          assistedQueryColumn (?):  # Assisted query column name
          plainColumn (?): # Plain column name
          encryptorName: # Encrypt algorithm name
      queryWithCipherColumn(?): # The current table whether query with cipher 
column for data encrypt.

  # Encrypt algorithm configuration
  encryptors:
    <encrypt-algorithm-name> (+): # Encrypt algorithm name
      type: # Encrypt algorithm type
      props: # Encrypt algorithm properties
        # ...

  queryWithCipherColumn: # Whether query with cipher column for data encrypt. User 
you can use plaintext to query if have
```

Please refer to Built-in Encrypt Algorithm List for more details about type of algorithm.

Procedure

1. Configure data encryption rules in the YAML file, including data sources, encryption rules, global attributes, and other configuration items.
2. Using the createDataSource of calling the YamlShardingSphereDataSourceFactory object to create ShardingSphereDataSource based on the configuration information in the YAML file.

Sample

The data encryption YAML configurations are as follows:

```yaml
dataSources:
  unique_ds:
    dataSourceClassName: com.zaxxer.hikari.HikariDataSource
    driverClassName: com.mysql.jdbc.Driver
    jdbcUrl: jdbc:mysql://localhost:3306/demo_ds?serverTimezone=UTC&useSSL=false&
useUnicode=true&characterEncoding=UTF-8
    username: root
    password:

rules:
- !ENCRYPT
  tables:
    t_user:
      columns:
        username:
          plainColumn: username_plain
          cipherColumn: username
          encryptorName: name-encryptor
        pwd:
          cipherColumn: pwd
          assistedQueryColumn: assisted_query_pwd
          encryptorName: pwd_encryptor
  encryptors:
    name-encryptor:
      type: AES
      props:
        aes-key-value: 123456abc
    pwd_encryptor:
      type: assistedTest
```

Read the YAML configuration to create a data source according to the createDataSource method of YamlShardingSphereDataSourceFactory.

```java
YamlShardingSphereDataSourceFactory.createDataSource(getFile());
```

Related References

- Core Feature: Data Encryption
- Developer Guide: Data Encryption

Shadow DB

Background

Please refer to the following configuration in order to use the ShardingSphere shadow DB feature in ShardingSphere-Proxy.

Parameters

```yaml
rules:
- !SHADOW
  dataSources:
    shadowDataSource:
      productionDataSourceName: # production data source name
      shadowDataSourceName: # shadow data source name
  tables:
    <table-name>:
      dataSourceNames: # shadow table associates shadow data source name list
        - <shadow-data-source>
      shadowAlgorithmNames: # shadow table associates shadow algorithm name list
        - <shadow-algorithm-name>
  defaultShadowAlgorithmName: # default shadow algorithm name (option)
  shadowAlgorithms:
    <shadow-algorithm-name> (+): # shadow algorithm name
      type: # shadow algorithm type
      props: # shadow algorithm attribute configuration
```

Please refer to Built-in shadow algorithm list for more details.

Procedure

1. Configure shadow DB rules in the YAML file, including data sources, shadow library rules, global properties and other configuration items;
2. Call the `createDataSource()` method of the `YamlShardingSphereDataSourceFactory` object to create a ShardingSphereDataSource based on the configuration information in the YAML file.

Sample

The YAML configuration sample of shadow DB is as follows:

```yaml
dataSources:
  ds:
    url: jdbc:mysql://127.0.0.1:3306/ds?serverTimezone=UTC&useSSL=false
    username: root
    password:
    connectionTimeoutMilliseconds: 30000
    idleTimeoutMilliseconds: 60000
    maxLifetimeMilliseconds: 1800000
    maxPoolSize: 50
    minPoolSize: 1
  shadow_ds:
    url: jdbc:mysql://127.0.0.1:3306/shadow_ds?serverTimezone=UTC&useSSL=false
    username: root
    password:
    connectionTimeoutMilliseconds: 30000
    idleTimeoutMilliseconds: 60000
    maxLifetimeMilliseconds: 1800000
    maxPoolSize: 50
    minPoolSize: 1

rules:
- !SHADOW
  dataSources:
    shadowDataSource:
      productionDataSourceName: ds
      shadowDataSourceName: shadow_ds
  tables:
    t_order:
      dataSourceNames:
        - shadowDataSource
      shadowAlgorithmNames:
        - user-id-insert-match-algorithm
        - simple-hint-algorithm
  shadowAlgorithms:
    user-id-insert-match-algorithm:
      type: REGEX_MATCH
      props:
        operation: insert
        column: user_id
        regex: "[1]"
    simple-hint-algorithm:
      type: SIMPLE_HINT
      props:
        foo: bar
```

Related References

- Core Features of Shadow DB
- JAVA API: Shadow DB Configuration
- Spring Boot Starter: Shadow DB Configuration
- Spring Namespace: Shadow DB Configuration

SQL-parser

Background

The SQL parser YAML configuration is readable and easy to use. The YAML files allow you to separate the code from the configuration, and easily modify the configuration file as needed.

Parameters

```yaml
rules:
- !SQL_PARSER
  sqlCommentParseEnabled: # Whether to parse SQL comments
  sqlStatementCache: # SQL statement local cache
    initialCapacity: # Initial capacity of local cache
    maximumSize: # Maximum capacity of local cache
  parseTreeCache: # Parse tree local cache
    initialCapacity: # Initial capacity of local cache
    maximumSize: # Maximum capacity of local cache
```

Procedure

1. Set local cache configuration.
2. Set parser configuration.
3. Use a parsing engine to parse SQL.

Sample

```yaml
rules:
  - !SQL_PARSER
    sqlCommentParseEnabled: true
    sqlStatementCache:
      initialCapacity: 2000
      maximumSize: 65535
    parseTreeCache:
```

```
    initialCapacity: 128
    maximumSize: 1024
```

Related References

- JAVA API: SQL Parsing
- Spring Boot Starter: SQL Parsing
- Spring namespace: SQL Parsing

SQL Translator

Configuration Item Explanation

```
rules:
- !SQL_TRANSLATOR
  type: # SQL translator type
  useOriginalSQLWhenTranslatingFailed: # Whether use original SQL when translating
failed
```

Mixed Rules

Background

ShardingSphere provides a variety of features, such as data sharding, read/write splitting, high availability, and data decryption. These features can be used independently or in combination. Below, you will find the parameters' explanation and configuration samples based on YAML.

Parameters

```
rules:
  - !SHARDING
    tables:
      <logic-table-name>: # Logical table name:
        actualDataNodes: # consists of logical data source name plus table name
(refer to Inline syntax rules)
        tableStrategy: # Table shards strategy. The same as database shards
strategy
          standard:
            shardingColumn: # Sharding column name
            shardingAlgorithmName: # Sharding algorithm name
        keyGenerateStrategy:
          column: # Auto-increment column name. By default, the auto-increment
```

```yaml
primary key generator is not used.
          keyGeneratorName: # Distributed sequence algorithm name
    defaultDatabaseStrategy:
      standard:
        shardingColumn: # Sharding column name
        shardingAlgorithmName: # Sharding algorithm name
    shardingAlgorithms:
      <sharding-algorithm-name>: # Sharding algorithm name
        type: INLINE
        props:
          algorithm-expression: # INLINE expression
      t_order_inline:
        type: INLINE
        props:
          algorithm-expression: # INLINE expression
    keyGenerators:
      <key-generate-algorithm-name> (+): # Distributed sequence algorithm name
        type: # Distributed sequence algorithm type
        props: # Property configuration of distributed sequence algorithm
  - !READWRITE_SPLITTING
    dataSources:
      <data-source-name>: # Read/write splitting logical data source name
        dynamicStrategy: # Read/write splitting type
          autoAwareDataSourceName: # Database discovery logical data source name
      <data-source-name>: # Read/write splitting logical data source name
        dynamicStrategy: # Read/write splitting type
          autoAwareDataSourceName: # Database discovery logical data source name
  - !DB_DISCOVERY
    dataSources:
      <data-source-name>:
        dataSourceNames: # Data source name list
          - ds_0
          - ds_1
          - ds_2
        discoveryHeartbeatName: # Detect heartbeat name
        discoveryTypeName: # Database discovery type name
      <data-source-name>:
        dataSourceNames: # Data source name list
          - ds_3
          - ds_4
          - ds_5
        discoveryHeartbeatName: # Detect heartbeat name
        discoveryTypeName: # Database discovery type name
    discoveryHeartbeats:
      <discovery-heartbeat-name>: # Heartbeat name
        props:
          keep-alive-cron: # cron expression, such as '0/5 * * * * ?'
    discoveryTypes:
```

```yaml
      <discovery-type-name>: # Database discovery type name
        type: # Database discovery type, such as MySQL.MGR.
        props:
          group-name:  # Required parameter of database discovery type, such as MGR
's group-name.
  - !ENCRYPT
    encryptors:
      <encrypt-algorithm-name> (+): # Encryption and decryption algorithm name
        type: # Encryption and decryption algorithm type
        props: # Encryption and decryption algorithm property configuration
      <encrypt-algorithm-name> (+): # Encryption and decryption algorithm name
        type: # Encryption and decryption algorithm type
    tables:
      <table-name>: # Encryption table name
        columns:
          <column-name>: # Encryption name
            plainColumn: # Plaincolumn name
            cipherColumn: # Ciphercolumn name
            encryptorName: # Encryption algorithm name
          <column-name>: # Encryption column name
            cipherColumn: # Ciphercolumn name
            encryptorName:  # Encryption algorithm name
```

Samples

```yaml
rules:
  - !SHARDING
    tables:
      t_order:
        actualDataNodes: replica_ds_${0..1}.t_order_${0..1}
        tableStrategy:
          standard:
            shardingColumn: order_id
            shardingAlgorithmName: t_order_inline
        keyGenerateStrategy:
          column: order_id
          keyGeneratorName: snowflake
    defaultDatabaseStrategy:
      standard:
        shardingColumn: user_id
        shardingAlgorithmName: database_inline
    shardingAlgorithms:
      database_inline:
        type: INLINE
        props:
          algorithm-expression: replica_ds_${user_id % 2}
```

```yaml
      t_order_inline:
        type: INLINE
        props:
          algorithm-expression: t_order_${order_id % 2}
      t_order_item_inline:
        type: INLINE
        props:
          algorithm-expression: t_order_item_${order_id % 2}
    keyGenerators:
      snowflake:
        type: SNOWFLAKE
  - !READWRITE_SPLITTING
    dataSources:
      replica_ds_0:
        dynamicStrategy:
          autoAwareDataSourceName: readwrite_ds_0
      replica_ds_1:
        dynamicStrategy:
          autoAwareDataSourceName: readwrite_ds_1
  - !DB_DISCOVERY
    dataSources:
      readwrite_ds_0:
        dataSourceNames:
          - ds_0
          - ds_1
          - ds_2
        discoveryHeartbeatName: mgr-heartbeat
        discoveryTypeName: mgr
      readwrite_ds_1:
        dataSourceNames:
          - ds_3
          - ds_4
          - ds_5
        discoveryHeartbeatName: mgr-heartbeat
        discoveryTypeName: mgr
    discoveryHeartbeats:
      mgr-heartbeat:
        props:
          keep-alive-cron: '0/5 * * * * ?'
    discoveryTypes:
      mgr:
        type: MySQL.MGR
        props:
          group-name: 558edd3c-02ec-11ea-9bb3-080027e39bd2
  - !ENCRYPT
    encryptors:
      aes_encryptor:
        type: AES
```

```yaml
        props:
          aes-key-value: 123456abc
      md5_encryptor:
        type: MD5
    tables:
      t_encrypt:
        columns:
          user_id:
            plainColumn: user_plain
            cipherColumn: user_cipher
            encryptorName: aes_encryptor
          order_id:
            cipherColumn: order_cipher
            encryptorName: md5_encryptor
```

Algorithm

Sharding

```yaml
shardingAlgorithms:
  # algorithmName is specified by users, and its property has to be consistent with that of shardingAlgorithmName in the sharding strategy.
  <algorithmName>:
    # type and props, please refer to the built-in sharding algorithm: https://shardingsphere.apache.org/document/current/en/user-manual/common-config/builtin-algorithm/sharding/
    type: xxx
    props:
      xxx: xxx
```

Encryption

```yaml
encryptors:
  # encryptorName is specified by users, and its property should be consistent with that of encryptorName in encryption rules.
  <encryptorName>:
    # type and props, please refer to the built-in encryption algorithm: https://shardingsphere.apache.org/document/current/en/user-manual/common-config/builtin-algorithm/encrypt/
    type: xxx
    props:
      xxx: xxx
```

Read/Write Splitting Load Balancer

```
loadBalancers:
  # loadBalancerName is specified by users, and its property has to be consistent
with that of loadBalancerName in read/write splitting rules.
    # type and props, please refer to the built-in read/write splitting algorithm
load balancer: https://shardingsphere.apache.org/document/current/en/user-manual/
common-config/builtin-algorithm/load-balance/
    type: xxx
    props:
      xxx: xxx
```

Shadow DB

```
loadBalancers:
  # shadowAlgorithmName is specified by users, and its property has to be
consistent with that of shadowAlgorithmNames in shadow DB rules.
  <shadowAlgorithmName>:
    # type and props, please refer to the built-in shadow DB algorithm: https://
shardingsphere.apache.org/document/current/en/user-manual/common-config/builtin-
algorithm/shadow/
    type: xxx
    props:
      xxx: xxx
```

High Availability

```
discoveryTypes:
  # discoveryTypeName is specified by users, and its property has to be consistent
with that of discoveryTypeName in the database discovery rules.
    type: xxx
    props:
      xxx: xxx
```

JDBC Driver

Background

ShardingSphere-JDBC provides a JDBC Driver, which can be used only through configuration changes without rewriting the code.

Parameters

Driver Class Name

`org.apache.shardingsphere.driver.ShardingSphereDriver`

URL Configuration

- Use jdbc:shardingsphere: as prefix
- Configuration file: xxx.yaml, keep consist format with YAML Configuration
- Configuration file loading rule:
  - No prefix means that the configuration file is loaded from the absolute path
  - `classpath:` prefix indicates that the configuration file is loaded from the classpath

Procedure

1. Import Maven Dependency

```
<dependency>
    <groupId>org.apache.shardingsphere</groupId>
    <artifactId>shardingsphere-jdbc-core</artifactId>
    <version>${shardingsphere.version}</version>
</dependency>
```

2. Use drive

- Use native drivers:

```
Class.forName("org.apache.shardingsphere.driver.ShardingSphereDriver");
String jdbcUrl = "jdbc:shardingsphere:classpath:config.yaml";

String sql = "SELECT i.* FROM t_order o JOIN t_order_item i ON o.order_id=i.order_
id WHERE o.user_id=? AND o.order_id=?";
try (
        Connection conn = DriverManager.getConnection(jdbcUrl);
        PreparedStatement ps = conn.prepareStatement(sql)) {
    ps.setInt(1, 10);
    ps.setInt(2, 1000);
    try (ResultSet rs = preparedStatement.executeQuery()) {
        while(rs.next()) {
            // ...
        }
    }
}
```

- Use database connection pool:

```java
String driverClassName = "org.apache.shardingsphere.driver.ShardingSphereDriver";
String jdbcUrl = "jdbc:shardingsphere:classpath:config.yaml";

// Take HikariCP as an example
HikariDataSource dataSource = new HikariDataSource();
dataSource.setDriverClassName(driverClassName);
dataSource.setJdbcUrl(jdbcUrl);

String sql = "SELECT i.* FROM t_order o JOIN t_order_item i ON o.order_id=i.order_
id WHERE o.user_id=? AND o.order_id=?";
try (
        Connection conn = dataSource.getConnection();
        PreparedStatement ps = conn.prepareStatement(sql)) {
    ps.setInt(1, 10);
    ps.setInt(2, 1000);
    try (ResultSet rs = preparedStatement.executeQuery()) {
        while(rs.next()) {
            // ...
        }
    }
}
```

Sample

Load JDBC URL of config.yaml profile in classpath:

```
jdbc:shardingsphere:classpath:config.yaml
```

Load JDBC URL of config.yaml profile in absolute path

```
jdbc:shardingsphere:/path/to/config.yaml
```

6.1.2 Java API

Overview

Java API is the basic configuration methods in ShardingSphere-JDBC, and other configurations will eventually be transformed into Java API configuration methods.

The Java API is the most complex and flexible configuration method, which is suitable for the scenarios requiring dynamic configuration through programming.

Usage

Import Maven Dependency

```
<dependency>
    <groupId>org.apache.shardingsphere</groupId>
    <artifactId>shardingsphere-jdbc-core</artifactId>
    <version>${shardingsphere.version}</version>
</dependency>
```

Create Data Source

ShardingSphere-JDBC Java API consists of database name, mode configuration, data source map, rule configurations and properties.

The ShardingSphereDataSource created by ShardingSphereDataSourceFactory implements the standard JDBC DataSource interface.

```
String databaseName = "foo_schema"; // Indicate logic database name
ModeConfiguration modeConfig = ... // Build mode configuration
Map<String, DataSource> dataSourceMap = ... // Build actual data sources
Collection<RuleConfiguration> ruleConfigs = ... // Build concentrate rule
configurations
Properties props = ... // Build properties
DataSource dataSource = ShardingSphereDataSourceFactory.
createDataSource(databaseName, modeConfig, dataSourceMap, ruleConfigs, props);
```

Please refer to Mode Configuration for more mode details.

Please refer to Data Source Confiugration for more data source details.

Please refer to Rules Confiugration for more rule details.

Use Data Source

Developer can choose to use native JDBC or ORM frameworks such as JPA, Hibernate or MyBatis through the DataSource.

Take native JDBC usage as an example:

```
// Create ShardingSphereDataSource
DataSource dataSource = ShardingSphereDataSourceFactory.
createDataSource(databaseName, modeConfig, dataSourceMap, ruleConfigs, props);

String sql = "SELECT i.* FROM t_order o JOIN t_order_item i ON o.order_id=i.order_
id WHERE o.user_id=? AND o.order_id=?";
try (
        Connection conn = dataSource.getConnection();
```

```
        PreparedStatement ps = conn.prepareStatement(sql)) {
    ps.setInt(1, 10);
    ps.setInt(2, 1000);
    try (ResultSet rs = preparedStatement.executeQuery()) {
        while(rs.next()) {
            // ...
        }
    }
}
```

Mode

Background

Build the running mode through Java API.

Parameters

Class name: org.apache.shardingsphere.infra.config.mode.ModeConfiguration

Attributes:

• Name*	Da taType	Description	D ef au lt Va lu e
type	String	Type of mode configurationValues could be: Standalone or Cluster	St an da lo ne
repository	Pe rsistRe positor yConfig uration	Persist repository configurationStandalone type uses StandalonePersistRepositoryConfigurationCluster type uses ClusterPersistRepositoryConfiguration	
overwrite	boolean	Whether overwrite persistent configuration with local configuration	f al se

Standalone Persist Configuration

Class name: org.apache.shardingsphere.mode.repository.standalone.StandalonePersistRepositoryConfiguration

Attributes:

Name	*DataType*	*Description*
type	String	Type of persist repository
props	Properties	Properties of persist repository

Cluster Persist Configuration

Class name: org.apache.shardingsphere.mode.repository.cluster.ClusterPersistRepositoryConfiguration

Attributes:

Name	*Data Type*	*Description*
type	String	Type of persist repository
namespace	String	Namespace of registry center
server-lists	String	Server lists of registry center
props	Properties	Properties of persist repository

Notes

1. Cluster mode deployment is recommended for production environment.
2. The ‘ZooKeeper’ registry center is recommended for cluster mode deployment.

Procedure

Introduce Maven Dependency

```
<dependency>
 <groupId>org.apache.shardingsphere</groupId>
 <artifactId>shardingsphere-jdbc-core-spring-boot-starter</artifactId>
 <version>${latest.release.version}</version>
</dependency>
```

Notice: Please change `${latest.release.version}` to the actual version.

Sample

Standalone Mode

```java
ModeConfiguration modeConfig = createModeConfiguration();
Map<String, DataSource> dataSourceMap = ... // Building real data sources
Collection<RuleConfiguration> ruleConfigs = ... // Build specific rules
Properties props = ... // Build property configuration
DataSource dataSource = ShardingSphereDataSourceFactory.
createDataSource(databaseName, modeConfig, dataSourceMap, ruleConfigs, props);

private ModeConfiguration createModeConfiguration() {
    return new ModeConfiguration("Standalone", new
StandalonePersistRepositoryConfiguration("H2", new Properties()), true);
}
```

Cluster Mode (Recommended)

```java
ModeConfiguration modeConfig = createModeConfiguration();
Map<String, DataSource> dataSourceMap = ... // Building real data sources
Collection<RuleConfiguration> ruleConfigs = ... // Build specific rules
Properties props = ... // Build property configuration
DataSource dataSource = ShardingSphereDataSourceFactory.
createDataSource(databaseName, modeConfig, dataSourceMap, ruleConfigs, props);

private ModeConfiguration createModeConfiguration() {
    return new ModeConfiguration("Cluster", new
ClusterPersistRepositoryConfiguration("ZooKeeper", "governance-sharding-db",
"localhost:2181", new Properties()), true);
}
```

Related References

- Installation and Usage of ZooKeeper Registry Center
- Please refer to Builtin Persist Repository List for more details about type of repository.

Data Source

Background

ShardingSphere-JDBC supports all database JDBC drivers and connection pools.

This section describes how to configure data sources through the JAVA API.

Procedure

1. Import Maven dependency.

```
<dependency>
    <groupId>org.apache.shardingsphere</groupId>
    <artifactId>shardingsphere-jdbc-core</artifactId>
    <version>${latest.release.version}</version>
</dependency>
```

> Notice: Please change ${latest.release.version} to the actual version.

Sample

```
ModeConfiguration modeConfig = // Build running mode
Map<String, DataSource> dataSourceMap = createDataSources();
Collection<RuleConfiguration> ruleConfigs = ... // Build specific rules
Properties props = ... // Build attribute configuration
DataSource dataSource = ShardingSphereDataSourceFactory.
createDataSource(databaseName, modeConfig, dataSourceMap, ruleConfigs, props);

private Map<String, DataSource> createDataSources() {
    Map<String, DataSource> dataSourceMap = new HashMap<>();
    // Configure the 1st data source
    HikariDataSource dataSource1 = new HikariDataSource();
    dataSource1.setDriverClassName("com.mysql.jdbc.Driver");
    dataSource1.setJdbcUrl("jdbc:mysql://localhost:3306/ds_1");
    dataSource1.setUsername("root");
    dataSource1.setPassword("");
    dataSourceMap.put("ds_1", dataSource1);

    // Configure the 2nd data source
    HikariDataSource dataSource2 = new HikariDataSource();
    dataSource2.setDriverClassName("com.mysql.jdbc.Driver");
    dataSource2.setJdbcUrl("jdbc:mysql://localhost:3306/ds_2");
    dataSource2.setUsername("root");
    dataSource2.setPassword("");
```

```
    dataSourceMap.put("ds_2", dataSource2);
}
```

Rules

Rules are pluggable part of Apache ShardingSphere. This chapter is a java rule configuration manual for ShardingSphere-JDBC.

Sharding

Background

The Java API rule configuration for data sharding, which allows users to create ShardingSphereDataSource objects directly by writing Java code, is flexible enough to integrate various types of business systems without relying on additional jar packages.

Parameters

Root Configuration

Class name: org.apache.shardingsphere.sharding.api.config.ShardingRuleConfiguration

Attributes:

Name	*DataType*	*Description*	*Default Value*
tables (+)	Collection<ShardingTableRuleConfiguration>	Sharding table rules	•
autoTables (+)	Collection<ShardingAutoTableRuleConfiguration>	Sharding auto table rules	•
bindingTableGroups (*)	Collection<String>	Binding table rules	Empty
broadcastTables (*)	Collection<String>	Broadcast table rules	Empty
defaultDatabaseShardingStrategy (?)	ShardingStrategyConfiguration	Default database sharding strategy	Not sharding
defaultTableShardingStrategy (?)	ShardingStrategyConfiguration	Default table sharding strategy	Not sharding
defaultKeyGenerateStrategy (?)	KeyGeneratorConfiguration	Default key generator	Snowflake
defaultShardingColumn (?)	String	Default sharding column name	None
shardingAlgorithms (+)	Map<String, AlgorithmConfiguration>	Sharding algorithm name and configurations	None
keyGenerators (?)	Map<String, AlgorithmConfiguration>	Key generate algorithm name and configurations	None

Sharding Table Configuration

Class name: org.apache.shardingsphere.sharding.api.config.ShardingTableRuleConfiguration

Attributes:

• Name*	*Dat aType*	*Description*	*Default Value*
logic Table	String	Name of sharding logic table	•
actua lData Nodes (?)	String	Describe data source names and actual tables, delimiter as point. Multiple data nodes split by comma, support inline expression	Broadcast table or databases sharding only
data baseS hardi ngStr ategy (?)	Shard ingStr ategyC onfigu ration	Databases sharding strategy	Use default databases sharding strategy
t ableS hardi ngStr ategy (?)	Shard ingStr ategyC onfigu ration	Tables sharding strategy	Use default tables sharding strategy
keyG enera teStr ategy (?)	K eyGene ratorC onfigu ration	Key generator configuration	Use default key generator

Sharding Auto Table Configuration

Class name: org.apache.shardingsphere.sharding.api.config.ShardingAutoTableRuleConfiguration

Attributes:

Name	*DataType*	*Description*	*Default Value*
lo gicTable	String	Name of sharding logic table	•
a ctualDat aSources (?)	String	Data source names. Multiple data nodes split by comma	Use all configured data sources
sharding Strategy (?)	Shardin gStrategyCo nfiguration	Sharding strategy	Use default sharding strategy
key Generate Strategy (?)	Key GeneratorCo nfiguration	Key generator configuration	Use default key generator

Sharding Strategy Configuration

Standard Sharding Strategy Configuration

Class name: org.apache.shardingsphere.sharding.api.config.strategy.sharding.StandardShardingStrategyConfiguration

Attributes:

Name	*DataType*	*Description*
shardingColumn	String	Sharding column name
shardingAlgorithmName	String	Sharding algorithm name

Complex Sharding Strategy Configuration

Class name: org.apache.shardingsphere.sharding.api.config.strategy.sharding.ComplexShardingStrategyConfiguration

Attributes:

Name	*DataType*	*Description*
shardingColumns	String	Sharding column name, separated by commas
shardingAlgorithmName	String	Sharding algorithm name

Hint Sharding Strategy Configuration

Class name: org.apache.shardingsphere.sharding.api.config.strategy.sharding.HintShardingStrategyConfiguration

Attributes:

Name	*DataType*	*Description*
shardingAlgorithmName	String	Sharding algorithm name

None Sharding Strategy Configuration

Class name: org.apache.shardingsphere.sharding.api.config.strategy.sharding.NoneShardingStrategyConfiguration

Attributes: None

Please refer to Built-in Sharding Algorithm List for more details about type of algorithm.

Distributed Key Strategy Configuration

Class name: org.apache.shardingsphere.sharding.api.config.strategy.keygen.KeyGenerateStrategyConfiguration

Attributes:

Name	*DataType*	*Description*
column	String	Column name of key generate
keyGeneratorName	String	key generate algorithm name

Please refer to Built-in Key Generate Algorithm List for more details about type of algorithm.

Procedure

1. Create an authentic data source mapping relationship, with key as the logical name of the data source and value as the DataSource object.
2. Create the sharding rule object ShardingRuleConfiguration, and initialize the sharding table objects—ShardingTableRuleConfiguration, the set of bound tables, the set of broadcast tables, and parameters like library sharding strategy and the database sharding strategy, on which the data sharding depends.
3. Using the ShardingSphereDataSource method of calling the ShardingSphereDataSourceFactory subject to create the ShardingSphereDataSource.

Sample

```
public final class ShardingDatabasesAndTablesConfigurationPrecise implements
ExampleConfiguration {

    @Override
    public DataSource getDataSource() throws SQLException {
        return ShardingSphereDataSourceFactory.
createDataSource(createDataSourceMap(), Collections.
singleton(createShardingRuleConfiguration()), new Properties());
    }

    private ShardingRuleConfiguration createShardingRuleConfiguration() {
        ShardingRuleConfiguration result = new ShardingRuleConfiguration();
        result.getTables().add(getOrderTableRuleConfiguration());
        result.getTables().add(getOrderItemTableRuleConfiguration());
        result.getBindingTableGroups().add("t_order, t_order_item");
        result.getBroadcastTables().add("t_address");
        result.setDefaultDatabaseShardingStrategy(new
StandardShardingStrategyConfiguration("user_id", "inline"));
        result.setDefaultTableShardingStrategy(new
StandardShardingStrategyConfiguration("order_id", "standard_test_tbl"));
```

```
        Properties props = new Properties();
        props.setProperty("algorithm-expression", "demo_ds_${user_id % 2}");
        result.getShardingAlgorithms().put("inline", new AlgorithmConfiguration(
"INLINE", props));
        result.getShardingAlgorithms().put("standard_test_tbl", new
AlgorithmConfiguration("STANDARD_TEST_TBL", new Properties()));
        result.getKeyGenerators().put("snowflake", new AlgorithmConfiguration(
"SNOWFLAKE", new Properties()));
        return result;
    }

    private ShardingTableRuleConfiguration getOrderTableRuleConfiguration() {
        ShardingTableRuleConfiguration result = new ShardingTableRuleConfiguration(
"t_order", "demo_ds_${0..1}.t_order_${[0, 1]}");
        result.setKeyGenerateStrategy(new KeyGenerateStrategyConfiguration("order_
id", "snowflake"));
        return result;
    }

    private ShardingTableRuleConfiguration getOrderItemTableRuleConfiguration() {
        ShardingTableRuleConfiguration result = new ShardingTableRuleConfiguration(
"t_order_item", "demo_ds_${0..1}.t_order_item_${[0, 1]}");
        result.setKeyGenerateStrategy(new KeyGenerateStrategyConfiguration("order_
item_id", "snowflake"));
        return result;
    }

    private Map<String, DataSource> createDataSourceMap() {
        Map<String, DataSource> result = new HashMap<>();
        result.put("demo_ds_0", DataSourceUtil.createDataSource("demo_ds_0"));
        result.put("demo_ds_1", DataSourceUtil.createDataSource("demo_ds_1"));
        return result;
    }
}
```

Related References

- Core Feature: Data Sharding
- Developer Guide: Data Sharding

Readwrite-splitting

Background

The read/write splitting configured in Java API form can be easily applied to various scenarios without relying on additional jar packages. Users only need to construct the read/write splitting data source through java code to be able to use the read/write splitting function.

Parameters Explained

Entry

Class name: org.apache.shardingsphere.readwritesplitting.api.ReadwriteSplittingRuleConfiguration

Configurable Properties:

• Name*	*DataType*	*Description*
d ataSo urces (+)	Colle ction<ReadwriteSplittingDa taSourceRuleConfiguration>	Data sources of write and reads
loa dBala ncers (*)	Map<String, AlgorithmConfig- uration>	Load balance algorithm name and configurations of replica data sources

Primary-secondary Data Source Configuration

Class name: org.apache.shardingsphere.readwritesplitting.api.rule.ReadwriteSplittingDataSourceRuleConfiguration

Configurable Properties:

Name	• Dat aTy pe*	*Description*	*Default Value*
name	Str ing	Readwrite-splitting data source name	•
stat icStrategy	Str ing	Static Readwrite-splitting configuration	•
dynam icStrategy	P rop ert ies	Dynamic Readwrite-splitting configuration	•
loadBa lancerName (?)	Str ing	Load balance algorithm name of replica sources	Round robin load balance algorithm

Class name：org.apache.shardingsphere.readwritesplitting.api.strategy.StaticReadwriteSplittingStrategyConfiguration

Configurable Properties:

Name	*DataType*	*Description*
writeDataSourceName	String	Write data source name
readDataSourceNames	List<String>	Read data sources list

Class name：org.apache.shardingsphere.readwritesplitting.api.strategy.DynamicReadwriteSplittingStrategyConfigurati

Configurable Properties:

Name	• DataType*	*Description*	*De fault V alue*
autoAwareDataSourceName	String	Database discovery logic data source name	•
writeDataSourceQueryEnabled (?)	String	All read data source are offline, write data source whether the data source is responsible for read traffic	true

Please refer to Built-in Load Balance Algorithm List for details on algorithm types. Please refer to Read-write splitting-Core features for more details about query consistent routing.

Operating Procedures

1. Add read-write splitting data source
2. Set load balancing algorithms
3. Use read-write splitting data source

Configuration Examples

```
public DataSource getDataSource() throws SQLException {
        ReadwriteSplittingDataSourceRuleConfiguration dataSourceConfig = new
ReadwriteSplittingDataSourceRuleConfiguration(
                "demo_read_query_ds", new
StaticReadwriteSplittingStrategyConfiguration("demo_write_ds",
                Arrays.asList("demo_read_ds_0", "demo_read_ds_1")), null,"demo_
weight_lb");
        Properties algorithmProps = new Properties();
        algorithmProps.setProperty("demo_read_ds_0", "2");
        algorithmProps.setProperty("demo_read_ds_1", "1");
```

```
        Map<String, AlgorithmConfiguration> algorithmConfigMap = new HashMap<>(1);
        algorithmConfigMap.put("demo_weight_lb", new AlgorithmConfiguration("WEIGHT
", algorithmProps));
        ReadwriteSplittingRuleConfiguration ruleConfig = new
ReadwriteSplittingRuleConfiguration(Collections.singleton(dataSourceConfig),
algorithmConfigMap);
        Properties props = new Properties();
        props.setProperty("sql-show", Boolean.TRUE.toString());
        return ShardingSphereDataSourceFactory.
createDataSource(createDataSourceMap(), Collections.singleton(ruleConfig), props);
    }

    private Map<String, DataSource> createDataSourceMap() {
        Map<String, DataSource> result = new HashMap<>(3, 1);
        result.put("demo_write_ds", DataSourceUtil.createDataSource("demo_write_ds
"));
        result.put("demo_read_ds_0", DataSourceUtil.createDataSource("demo_read_ds_
0"));
        result.put("demo_read_ds_1", DataSourceUtil.createDataSource("demo_read_ds_
1"));
        return result;
    }
```

References

- Read-write splitting-Core features
- YAML Configuration: read-write splitting
- Spring Boot Starter: read-write splitting
- Spring namespace: read-write splitting

Distributed Transaction

Root Configuration

org.apache.shardingsphere.transaction.config.TransactionRuleConfiguration

Attributes:

name	*DataType*	*Description*
defaultType	String	Default transaction type
providerType (?)	String	Transaction provider type
props (?)	Properties	Transaction properties

HA

Background

Build high availability rule configuration through `Java API`.

Parameters

Root Configuration

Class name: org.apache.shardingsphere.dbdiscovery.api.config.DatabaseDiscoveryRuleConfiguration
Attributes:

Name	*Data Type*	*Description*
dataSources (+)	Collection<DatabaseDisco veryDataSourceRuleConfiguration>	Data source configuration
discover yHeartbeats (+)	Map<String, Database DiscoveryHeartBeatConfiguration>	Detect heartbeat configuration
dis coveryTypes (+)	Map<String, AlgorithmConfiguration>	Database discovery type configuration

Data Source Configuration

Class name: org.apache.shardingsphere.dbdiscovery.api.config.rule.DatabaseDiscoveryDataSourceRuleConfiguration

Attributes:

Name	*Data Type*	*Description*
groupName (+)	String	Database discovery group name
data SourceNames (+)	Co llection <String>	Data source names, multiple data source names separated with comma. Such as: ds_0, ds_1
discoveryHe artbeatName (+)	String	Detect heartbeat name
discov eryTypeName (+)	String	Database discovery type name

Detect Heartbeat Configuration

Class name: org.apache.shardingsphere.dbdiscovery.api.config.rule.DatabaseDiscoveryHeartBeatConfiguration

Attributes:

• Name*	• DataType*	*Description*
props (+)	Properties	Detect heartbeat attribute configuration, keep-alive-cron configuration, cron expression. Such as: `0/5 * * * * ?`

Database Discovery Type Configuration

Class name: org.apache.shardingsphere.infra.config.algorithm.AlgorithmConfiguration

Attributes:

Name	*Data Type*	*Description*
type (+)	String	Database discovery type, such as: MySQL.MGR
props (?)	Properties	Required parameters for high-availability types, such as MGR's group-name

Procedure

1. Import Maven dependency.

```
<dependency>
    <groupId>org.apache.shardingsphere</groupId>
    <artifactId>shardingsphere-jdbc-core</artifactId>
    <version>${latest.release.version}</version>
</dependency>
```

Notice: Please change `${latest.release.version}` to the actual version.

Sample

```
// Build data source ds_0, ds_1, ds_2
Map<String, DataSource> dataSourceMap = new HashMap<>(3, 1);
dataSourceMap.put("ds_0", createDataSource1("primary_demo_ds"));
dataSourceMap.put("ds_1", createDataSource2("primary_demo_ds"));
dataSourceMap.put("ds_2", createDataSource3("primary_demo_ds"));

DataSource dataSource = ShardingSphereDataSourceFactory.createDataSource("database_
discovery_db", dataSourceMap, Arrays.asList(createDatabaseDiscoveryConfiguration(),
createReadwriteSplittingConfiguration()), null);

private static DatabaseDiscoveryRuleConfiguration
createDatabaseDiscoveryConfiguration() {
    DatabaseDiscoveryDataSourceRuleConfiguration dataSourceRuleConfiguration = new
DatabaseDiscoveryDataSourceRuleConfiguration("readwrite_ds", Arrays.asList("ds_0,
ds_1, ds_2"), "mgr-heartbeat", "mgr");
    return new DatabaseDiscoveryRuleConfiguration(Collections.
singleton(dataSourceRuleConfiguration), createDiscoveryHeartbeats(),
createDiscoveryTypes());
}

private static ReadwriteSplittingRuleConfiguration
createReadwriteSplittingConfiguration() {
    ReadwriteSplittingDataSourceRuleConfiguration dataSourceConfiguration1 = new
ReadwriteSplittingDataSourceRuleConfiguration("replica_ds", new
DynamicReadwriteSplittingStrategyConfiguration("readwrite_ds", true), "");
    return new ReadwriteSplittingRuleConfiguration(Arrays.
asList(dataSourceConfiguration1), Collections.emptyMap());
}

private static Map<String, AlgorithmConfiguration> createDiscoveryTypes() {
    Map<String, AlgorithmConfiguration> discoveryTypes = new HashMap<>(1,  1);
    Properties props = new Properties();
    props.put("group-name", "558edd3c-02ec-11ea-9bb3-080027e39bd2");
    discoveryTypes.put("mgr", new AlgorithmConfiguration("MGR", props));
    return discoveryTypes;
}

private static Map<String, DatabaseDiscoveryHeartBeatConfiguration>
createDiscoveryHeartbeats() {
    Map<String, DatabaseDiscoveryHeartBeatConfiguration>
discoveryHeartBeatConfiguration = new HashMap<>(1,  1);
    Properties props = new Properties();
    props.put("keep-alive-cron", "0/5 * * * * ?");
    discoveryHeartBeatConfiguration.put("mgr-heartbeat", new
DatabaseDiscoveryHeartBeatConfiguration(props));
    return discoveryHeartBeatConfiguration;
```

```
}
```

Related References

- Feature Description of HA
- YAML Configuration: HA
- Spring Boot Starter: HA
- Spring Namespace: HA

Encryption

Background

The data encryption Java API rule configuration allows users to directly create ShardingSphereDataSource objects by writing java code. The Java API configuration method is very flexible and can integrate various types of business systems without relying on additional jar packages.

Parameters

Root Configuration

Class name: org.apache.shardingsphere.encrypt.api.config.EncryptRuleConfiguration

Attributes:

Name	*DataType*	*Description*	*D efa ult Val ue*
tables (+)	Collection<En crypt-TableRule Configura-tion>	Encrypt table rule configurations	
enc ryptors (+)	Map<String, Algo-rithm Configuration>	Encrypt algorithm name and configurations	
queryWi thCi-phe rColumn (?)	boolean	Whether query with cipher column for data en-crypt. User you can use plaintext to query if have	t rue

Encrypt Table Rule Configuration

Class name: org.apache.shardingsphere.encrypt.api.config.rule.EncryptTableRuleConfiguration

Attributes:

Name	*DataType*	*Description*
name	String	Table name
columns (+)	Collection<EncryptColumnRuleConfiguration>	Encrypt column rule configurations
queryWithCipherColumn (?)	boolean	The current table whether query with cipher column for data encrypt

Encrypt Column Rule Configuration

Class name: org.apache.shardingsphere.encrypt.api.config.rule.EncryptColumnRuleConfiguration

Attributes:

Name	*DataType*	*Description*
logicColumn	String	Logic column name
cipherColumn	String	Cipher column name
assistedQueryColumn (?)	String	Assisted query column name
plainColumn (?)	String	Plain column name
encryptorName	String	Encrypt algorithm name
assistedQueryEncryptorName	String	Assisted query encrypt algorithm name
queryWithCipherColumn (?)	boolean	The current column whether query with cipher column for data encrypt

Encrypt Algorithm Configuration

Class name: org.apache.shardingsphere.infra.config.algorithm.AlgorithmConfiguration

Attributes:

Name	*DataType*	*Description*
name	String	Encrypt algorithm name
type	String	Encrypt algorithm type
properties	Properties	Encrypt algorithm properties

Please refer to Built-in Encrypt Algorithm List for more details about type of algorithm.

Procedure

1. Create a real data source mapping relationship, where key is the logical name of the data source and value is the datasource object.
2. Create the encryption rule object EncryptRuleConfiguration, and initialize the encryption table object EncryptTableRuleConfiguration, encryption algorithm and other parameters in the object.
3. Call createDataSource of ShardingSphereDataSourceFactory to create ShardingSphereDataSource.

Sample

```
public final class EncryptDatabasesConfiguration implements ExampleConfiguration {

    @Override
    public DataSource getDataSource() {
        Properties props = new Properties();
        props.setProperty("aes-key-value", "123456");
        EncryptColumnRuleConfiguration columnConfigAes = new
EncryptColumnRuleConfiguration("username", "username", "", "username_plain", "name_
encryptor", null);
        EncryptColumnRuleConfiguration columnConfigTest = new
EncryptColumnRuleConfiguration("pwd", "pwd", "assisted_query_pwd", "", "pwd_
encryptor", null);
        EncryptTableRuleConfiguration encryptTableRuleConfig = new
EncryptTableRuleConfiguration("t_user", Arrays.asList(columnConfigAes,
columnConfigTest), null);
        Map<String, AlgorithmConfiguration> encryptAlgorithmConfigs = new
LinkedHashMap<>(2, 1);
        encryptAlgorithmConfigs.put("name_encryptor", new AlgorithmConfiguration(
"AES", props));
        encryptAlgorithmConfigs.put("pwd_encryptor", new AlgorithmConfiguration(
"assistedTest", props));
        EncryptRuleConfiguration encryptRuleConfig = new
EncryptRuleConfiguration(Collections.singleton(encryptTableRuleConfig),
encryptAlgorithmConfigs);
        try {
            return ShardingSphereDataSourceFactory.createDataSource(DataSourceUtil.
createDataSource("demo_ds"), Collections.singleton(encryptRuleConfig), props);
        } catch (final SQLException ex) {
            ex.printStackTrace();
            return null;
        }
    }
}
```

Related References

- The feature description of Data Encryption
- Dev Guide of Data Encryption

Shadow DB

Background

In the distributed application architecture based on microservices, businesses require multiple services to be completed through a series of services and middleware, so the stress test of a single service can no longer meet the needs of real scenarios. If we reconstruct a stress test environment similar to the production environment, it is too expensive and often fails to simulate the complexity and traffic of the online environment. For this reason, the industry often chooses the full link stress test, which is performed in the production environment, so that the test results can accurately reflect the true capacity and performance of the system.

Parameters

Root Configuration

Class name: org.apache.shardingsphere.shadow.api.config.ShadowRuleConfiguration

Attributes:

Name	*Data Type*	*Description*
dataSources	Map<String, ShadowD ataSourceConfiguration>	shadow data source mapping name and configuration
tables	Map<String, Sh adowTableConfiguration>	shadow table name and configuration
shad owAlgorithms	Map<String, AlgorithmConfiguration>	shadow algorithm name and configuration
de faultShadowA lgorithmName	String	default shadow algorithm name

Shadow Data Source Configuration

Class name: org.apache.shardingsphere.shadow.api.config.datasource.ShadowDataSourceConfiguration

Attributes:

Name	*DataType*	*Description*
productionDataSourceName	String	Production data source name
shadowDataSourceName	String	Shadow data source name

Shadow Table Configuration

Class name: org.apache.shardingsphere.shadow.api.config.table.ShadowTableConfiguration

Attributes:

Name	*Data Type*	*Description*
dataSourceNames	Collection<String>	shadow table associates shadow data source mapping name list
shadowAlgorithmNames	Collection<String>	shadow table associates shadow algorithm name list

Shadow Algorithm Configuration

Class name： org.apache.shardingsphere.infra.config.algorithm.AlgorithmConfiguration

Attributes:

Name	*Data Type*	*Description*
type	String	shadow algorithm type
props	Properties	shadow algorithm configuration

Please refer to Built-in Shadow Algorithm List.

Procedure

1. Create production and shadow data source.
2. Configure shadow rule.

- Configure shadow data source
- Configure shadow table
- Configure shadow algorithm

Sample

```
public final class ShadowConfiguration {

    @Override
    public DataSource getDataSource() throws SQLException {
        Map<String, DataSource> dataSourceMap = createDataSourceMap();
        return ShardingSphereDataSourceFactory.createDataSource(dataSourceMap,
createRuleConfigurations(), createShardingSphereProps());
    }
```

```
    private Map<String, DataSource> createDataSourceMap() {
        Map<String, DataSource> result = new LinkedHashMap<>();
        result.put("ds", DataSourceUtil.createDataSource("demo_ds"));
        result.put("ds_shadow", DataSourceUtil.createDataSource("shadow_demo_ds"));
        return result;
    }

    private Collection<RuleConfiguration> createRuleConfigurations() {
        Collection<RuleConfiguration> result = new LinkedList<>();
        ShadowRuleConfiguration shadowRule = new ShadowRuleConfiguration();
        shadowRule.setDataSources(createShadowDataSources());
        shadowRule.setTables(createShadowTables());
        shadowRule.setShadowAlgorithms(createShadowAlgorithmConfigurations());
        result.add(shadowRule);
        return result;
    }

    private Map<String, ShadowDataSourceConfiguration> createShadowDataSources() {
        Map<String, ShadowDataSourceConfiguration> result = new LinkedHashMap<>();
        result.put("shadow-data-source", new ShadowDataSourceConfiguration("ds",
"ds_shadow"));
        return result;
    }

    private Map<String, ShadowTableConfiguration> createShadowTables() {
        Map<String, ShadowTableConfiguration> result = new LinkedHashMap<>();
        result.put("t_user", new ShadowTableConfiguration(Collections.
singletonList("shadow-data-source"), createShadowAlgorithmNames()));
        return result;
    }

    private Collection<String> createShadowAlgorithmNames() {
        Collection<String> result = new LinkedList<>();
        result.add("user-id-insert-match-algorithm");
        result.add("simple-hint-algorithm");
        return result;
    }

    private Map<String, AlgorithmConfiguration>
createShadowAlgorithmConfigurations() {
        Map<String, AlgorithmConfiguration> result = new LinkedHashMap<>();
        Properties userIdInsertProps = new Properties();
        userIdInsertProps.setProperty("operation", "insert");
        userIdInsertProps.setProperty("column", "user_type");
        userIdInsertProps.setProperty("value", "1");
        result.put("user-id-insert-match-algorithm", new AlgorithmConfiguration(
"VALUE_MATCH", userIdInsertProps));
        return result;
```

```
    }
}
```

Related References

Features Description of Shadow DB

SQL Parser

Background

SQL is the standard language for users to communicate with databases. The SQL parsing engine is responsible for parsing the SQL string into an abstract syntax tree for Apache ShardingSphere to understand and implement its incremental function. Currently, MySQL, PostgreSQL, SQLServer, Oracle, openGauss and SQL dialects conforming to SQL92 specifications are supported. Due to the complexity of SQL syntax, there are still a few unsupported SQLs. By using SQL parsing in the form of Java API, you can easily integrate into various systems and flexibly customize user requirements.

Parameters

Class: org.apache.shardingsphere.parser.config.SQLParserRuleConfiguration

Attributes:

name	*DataType*	*Description*
sqlCommentParseEnabled (?)	boolean	Whether to parse SQL comments
parseTreeCache (?)	CacheOption	Parse syntax tree local cache configuration
sqlStatementCache (?)	CacheOption	sql statement local cache configuration

Cache option Configuration

Class：org.apache.shardingsphere.sql.parser.api.CacheOption

Attributes:

na me	• DataType*	*Des cription*	*Default Value*
ini tia lCa pac ity	int	Initial capacity of local cache	parser syntax tree local cache default value 128, SQL statement cache default value 2000
ma xim umS ize (?)	long	Maximum capacity of local cache	The default value of local cache for parsing syntax tree is 1024, and the default value of sql statement cache is 65535

Procedure

1. Set local cache configuration.
2. Set resolution configuration.
3. Use the parsing engine to parse SQL.

Sample

```
CacheOption cacheOption = new CacheOption(128, 1024L);
SQLParserEngine parserEngine = new SQLParserEngine("MySQL", cacheOption);
ParseASTNode parseASTNode = parserEngine.parse("SELECT t.id, t.name, t.age FROM
table1 AS t ORDER BY t.id DESC;", false);
SQLVisitorEngine visitorEngine = new SQLVisitorEngine("MySQL", "STATEMENT", false,
new Properties());
MySQLStatement sqlStatement = visitorEngine.visit(parseASTNode);
System.out.println(sqlStatement.toString());
```

Related References

- YAML Configuration: SQL Parser
- Spring Boot Starter: SQL Parser
- Spring Namespace: SQL Parser

SQL Translator

Root Configuration

Class: org.apache.shardingsphere.sqltranslator.api.config.SQLTranslatorRuleConfiguration

Attributes:

name	*DataType*	*Description*
type	String	SQL translator type
useOriginalSQLWhenTranslatingFailed (?)	boolean	Whether use original SQL when translating failed

Mixed Rules

Background

ShardingSphere provides a variety of features, such as data sharding, read/write splitting, high availability, and data decryption. These features can be used independently or in combination. Below, you will find the configuration samples based on JAVA API.

Samples

```
// Sharding configuration
private ShardingRuleConfiguration createShardingRuleConfiguration() {
    ShardingRuleConfiguration result = new ShardingRuleConfiguration();
    result.getTables().add(getOrderTableRuleConfiguration());
    result.setDefaultDatabaseShardingStrategy(new
StandardShardingStrategyConfiguration("user_id", "inline"));
    result.setDefaultTableShardingStrategy(new
StandardShardingStrategyConfiguration("order_id", "standard_test_tbl"));
    Properties props = new Properties();
    props.setProperty("algorithm-expression", "demo_ds_${user_id % 2}");
    result.getShardingAlgorithms().put("inline", new AlgorithmConfiguration("INLINE
", props));
    result.getShardingAlgorithms().put("standard_test_tbl", new
AlgorithmConfiguration("STANDARD_TEST_TBL", new Properties()));
    result.getKeyGenerators().put("snowflake", new AlgorithmConfiguration(
"SNOWFLAKE", new Properties()));
    return result;
}

private ShardingTableRuleConfiguration getOrderTableRuleConfiguration() {
    ShardingTableRuleConfiguration result = new ShardingTableRuleConfiguration("t_
```

```
order", "demo_ds_${0..1}.t_order_${[0, 1]}");
    result.setKeyGenerateStrategy(new KeyGenerateStrategyConfiguration("order_id",
"snowflake"));
    return result;
}

// Dynamic read/write splitting configuration
private static ReadwriteSplittingRuleConfiguration
createReadwriteSplittingConfiguration() {
    ReadwriteSplittingDataSourceRuleConfiguration dataSourceConfiguration1 = new
ReadwriteSplittingDataSourceRuleConfiguration("replica_ds_0", new
DynamicReadwriteSplittingStrategyConfiguration("readwrite_ds_0", true), "");
    ReadwriteSplittingDataSourceRuleConfiguration dataSourceConfiguration2 = new
ReadwriteSplittingDataSourceRuleConfiguration("replica_ds_1", new
DynamicReadwriteSplittingStrategyConfiguration("readwrite_ds_1", true), "");
    Collection<ReadwriteSplittingDataSourceRuleConfiguration> dataSources = new
LinkedList<>();
    dataSources.add(dataSourceRuleConfiguration1);
    dataSources.add(dataSourceRuleConfiguration2);
    return new ReadwriteSplittingRuleConfiguration(dataSources, Collections.
emptyMap());
}

// Database discovery configuration
private static DatabaseDiscoveryRuleConfiguration
createDatabaseDiscoveryConfiguration() {
    DatabaseDiscoveryDataSourceRuleConfiguration dataSourceRuleConfiguration1 = new
DatabaseDiscoveryDataSourceRuleConfiguration("readwrite_ds_0", Arrays.asList("ds_0,
ds_1, ds_2"), "mgr-heartbeat", "mgr");
    DatabaseDiscoveryDataSourceRuleConfiguration dataSourceRuleConfiguration2 = new
DatabaseDiscoveryDataSourceRuleConfiguration("readwrite_ds_1", Arrays.asList("ds_3,
ds_4, ds_5"), "mgr-heartbeat", "mgr");
    Collection<DatabaseDiscoveryDataSourceRuleConfiguration> dataSources = new
LinkedList<>();
    dataSources.add(dataSourceRuleConfiguration1);
    dataSources.add(dataSourceRuleConfiguration2);
    return new DatabaseDiscoveryRuleConfiguration(configs,
createDiscoveryHeartbeats(), createDiscoveryTypes());
}

private static DatabaseDiscoveryRuleConfiguration
createDatabaseDiscoveryConfiguration() {
    DatabaseDiscoveryDataSourceRuleConfiguration dataSourceRuleConfiguration = new
DatabaseDiscoveryDataSourceRuleConfiguration("readwrite_ds_1", Arrays.asList("ds_3,
ds_4, ds_5"), "mgr-heartbeat", "mgr");
    return new DatabaseDiscoveryRuleConfiguration(Collections.
singleton(dataSourceRuleConfiguration), createDiscoveryHeartbeats(),
createDiscoveryTypes());
```

```
}

private static Map<String, AlgorithmConfiguration> createDiscoveryTypes() {
    Map<String, AlgorithmConfiguration> result = new HashMap<>(1,  1);
    Properties props = new Properties();
    props.put("group-name", "558edd3c-02ec-11ea-9bb3-080027e39bd2");
    discoveryTypes.put("mgr", new AlgorithmConfiguration("MGR", props));
    return result;
}

private static Map<String, DatabaseDiscoveryHeartBeatConfiguration>
createDiscoveryHeartbeats() {
    Map<String, DatabaseDiscoveryHeartBeatConfiguration> result = new HashMap<>(1,
1);
    Properties props = new Properties();
    props.put("keep-alive-cron", "0/5 * * * * ?");
    discoveryHeartBeatConfiguration.put("mgr-heartbeat", new
DatabaseDiscoveryHeartBeatConfiguration(props));
    return result;
}

// Data decryption configuration
public EncryptRuleConfiguration createEncryptRuleConfiguration() {
    Properties props = new Properties();
    props.setProperty("aes-key-value", "123456");
    EncryptColumnRuleConfiguration columnConfigAes = new
EncryptColumnRuleConfiguration("username", "username", "", "username_plain", "name_
encryptor", null);
    EncryptColumnRuleConfiguration columnConfigTest = new
EncryptColumnRuleConfiguration("pwd", "pwd", "assisted_query_pwd", "", "pwd_
encryptor", null);
    EncryptTableRuleConfiguration encryptTableRuleConfig = new
EncryptTableRuleConfiguration("t_user", Arrays.asList(columnConfigAes,
columnConfigTest), null);
    Map<String, AlgorithmConfiguration> encryptAlgorithmConfigs = new LinkedHashMap
<>(2, 1);
    encryptAlgorithmConfigs.put("name_encryptor", new AlgorithmConfiguration("AES",
props));
    encryptAlgorithmConfigs.put("pwd_encryptor", new AlgorithmConfiguration(
"assistedTest", props));
    EncryptRuleConfiguration result = new EncryptRuleConfiguration(Collections.
singleton(encryptTableRuleConfig), encryptAlgorithmConfigs);
    return result;
}
```

Algorithm

Sharding

```
ShardingRuleConfiguration ruleConfiguration = new ShardingRuleConfiguration();
// algorithmName is specified by users and should be consistent with the sharding algorithm in the sharding strategy.
// type and props, please refer to the built-in sharding algorithm: https://shardingsphere.apache.org/document/current/en/user-manual/common-config/builtin-algorithm/sharding/
ruleConfiguration.getShardingAlgorithms().put("algorithmName", new AlgorithmConfiguration("xxx", new Properties()));
```

Encryption

```
// encryptorName is specified by users, and its property should be consistent with that of encryptorName in encryption rules.
// type and props, please refer to the built-in encryption algorithm: https://shardingsphere.apache.org/document/current/en/user-manual/common-config/builtin-algorithm/encrypt/
Map<String, AlgorithmConfiguration> algorithmConfigs = new LinkedHashMap<>(1, 1);
algorithmConfigs.put("encryptorName", new AlgorithmConfiguration("xxx", new Properties()));
```

Read/Write Splitting Load Balancer

```
// loadBalancerName is specified by users, and its property has to be consistent with that of loadBalancerName in read/write splitting rules.
// type and props, please refer to the built-in read/write splitting algorithm load balancer: https://shardingsphere.apache.org/document/current/en/user-manual/common-config/builtin-algorithm/load-balance/
Map<String, AlgorithmConfiguration> algorithmConfigs = new HashMap<>(1, 1);
algorithmConfigs.put("loadBalancerName", new AlgorithmConfiguration("xxx", new Properties()));
```

Shadow DB

```
// shadowAlgorithmName is specified by users, and its property has to be consistent with that of shadowAlgorithmNames in shadow DB rules.
// type and props, please refer to the built-in shadow DB algorithm: https://shardingsphere.apache.org/document/current/en/user-manual/common-config/builtin-algorithm/shadow/
Map<String, AlgorithmConfiguration> algorithmConfigs = new HashMap<>(1, 1);
```

```
algorithmConfigs.put("shadowAlgorithmName", new AlgorithmConfiguration("xxx", new
Properties()));
```

High Availability

```
// discoveryTypeName is specified by users, and its property has to be consistent
with that of discoveryTypeName in database discovery rules.
Map<String, AlgorithmConfiguration> algorithmConfigs = new HashMap<>(1, 1);
algorithmConfigs.put("discoveryTypeName", new AlgorithmConfiguration("xxx", new
Properties()));
```

6.1.3 Spring Boot Starter

Overview

ShardingSphere-JDBC provides official Spring Boot Starter to make convenient for developers to integrate ShardingSphere-JDBC and Spring Boot.

Usage

Import Maven Dependency

```
<dependency>
    <groupId>org.apache.shardingsphere</groupId>
    <artifactId>shardingsphere-jdbc-core-spring-boot-starter</artifactId>
    <version>${shardingsphere.version}</version>
</dependency>
```

Configure Spring Boot Properties

ShardingSphere-JDBC spring boot properties consists of database name, mode configuration, data source map, rule configurations and properties.

```
# JDBC logic database name. Through this parameter to connect ShardingSphere-JDBC
and ShardingSphere-Proxy.
spring.shardingsphere.database.name= # logic database name, default value: logic_db
spring.shardingsphere.mode.xxx= # mode configuration
spring.shardingsphere.dataSource.xxx= # data source map
spring.shardingsphere.rules.xxx= # rule configurations
spring.shardingsphere.props= # properties
```

Please refer to Mode Confiugration for more mode details.

Please refer to Data Source Confiugration for more data source details.

Please refer to Rules Confiugration for more rule details.

Use Data Source

Developer can inject to use native JDBC or ORM frameworks such as JPA, Hibernate or MyBatis through the DataSource.

Take native JDBC usage as an example:

```
@Resource
private DataSource dataSource;
```

Mode

Parameters

```
mode (?): # Default value is Standalone
  type: # Type of mode configuration. Values could be: Standalone or Cluster
  repository (?): # Persist repository configuration
  overwrite: # Whether overwrite persistent configuration with local configuration
```

Standalone Mode

```
mode:
  type: Standalone
  repository:
    type: # Type of persist repository
    props: # Properties of persist repository
      foo_key: foo_value
      bar_key: bar_value
  overwrite: # Whether overwrite persistent configuration with local configuration
```

Cluster Mode (recommended)

```
mode:
  type: Cluster
  repository:
    type: # Type of persist repository
    props: # Properties of persist repository
      namespace: # Namespace of registry center
      server-lists: # Server lists of registry center
      foo_key: foo_value
```

```
    bar_key: bar_value
  overwrite: # Whether overwrite persistent configuration with local configuration
```

Notes

1. Cluster mode deployment is recommended for production environment.
2. The ‘ZooKeeper’ registry center is recommended for cluster mode deployment.

Sample

Standalone Mode

```
mode:
  type: Standalone
  repository:
    type: H2
  overwrite: false
```

Cluster Mode (recommended)

```
mode:
  type: Cluster
  repository:
    type: ZooKeeper
    props:
      namespace: governance
      server-lists: localhost:2181
      retryIntervalMilliseconds: 500
      timeToLiveSeconds: 60
  overwrite: false
```

Related References

- Installation and Usage of ZooKeeper Registry Center
- Please refer to Builtin Persist Repository List for more details about the type of repository.

Data Source

Background information

Use local datasource

The database driver showed in the example is MySQL and the connection pool is HikariCP, either of which can be replaced by other database drivers and connection pools. When using ShardingSphere JDBC, the property names of the JDBC pools depend on its own definition instead of being fixed by ShardingSphere. See relevant procedures at `org.apache.shardingsphere.infra.datasource.pool.creator.DataSourcePoolCreator`. For example, using `url` instead of `jdbc-url` for Alibaba Druid 1.2.9 is the expected behavior.

Use datasource JNDI

If you wish to use JNDI for database configuration, you can replace a series of datasource configurations with `spring.shardingsphere.datasource.${datasourceName}.jndiName` when you are using ShardingSphere-JDBC on application servers(e.g. Tomcat).

Parameters Explanation

Using local datasource

```
spring.shardingsphere.datasource.names= # Actual datasource names. Multiple
datasources are separated with comma

# <actual-data-source-name>  to show actual datasource name
spring.shardingsphere.datasource.<actual-data-source-name>.type= # Full class name
of the database connection pool
spring.shardingsphere.datasource.<actual-data-source-name>.driver-class-name= #
Database-driven class name, based on the database connection pool's own
configuration
spring.shardingsphere.datasource.<actual-data-source-name>.jdbc-url= # Database URL
connection, in ine with the connection pool's own configuration
spring.shardingsphere.datasource.<actual-data-source-name>.username= # database
user names, in line with the connection pool's own configuration
spring.shardingsphere.datasource.<actual-data-source-name>.password= # database
password , in line with the connection pool's own configuration
spring.shardingsphere.datasource.<actual-data-source-name>.<xxx>= # ... Other
properties of the database connection pool
```

Using JNDI datasource

```
spring.shardingsphere.datasource.names= # Authentic datasource names. Multiple
datasources are separated with comma
# <actual-data-source-name> to show actual datasource name
spring.shardingsphere.datasource.<actual-data-source-name>.jndi-name= # datasource
JNDI
```

Configuration Examples

Using local datasource

```
# configure actual datasource
spring.shardingsphere.datasource.names=ds1,ds2

# configure the first datasource
spring.shardingsphere.datasource.ds1.type=com.zaxxer.hikari.HikariDataSource
spring.shardingsphere.datasource.ds1.driver-class-name=com.mysql.jdbc.Driver
spring.shardingsphere.datasource.ds1.jdbc-url=jdbc:mysql://localhost:3306/ds1
spring.shardingsphere.datasource.ds1.username=root
spring.shardingsphere.datasource.ds1.password=

# configure the second datasource
spring.shardingsphere.datasource.ds2.type=com.zaxxer.hikari.HikariDataSource
spring.shardingsphere.datasource.ds2.driver-class-name=com.mysql.jdbc.Driver
spring.shardingsphere.datasource.ds2.jdbc-url=jdbc:mysql://localhost:3306/ds2
spring.shardingsphere.datasource.ds2.username=root
spring.shardingsphere.datasource.ds2.password=
```

Using JNDI datasource

```
# configure actual datasource
spring.shardingsphere.datasource.names=ds1,ds2
# configure the first datasource
spring.shardingsphere.datasource.ds1.jndi-name=java:comp/env/jdbc/ds1
# configure the second datasource
spring.shardingsphere.datasource.ds2.jndi-name=java:comp/env/jdbc/ds2
```

Rules

Rules are pluggable part of Apache ShardingSphere. This chapter is a Spring Boot Starter rule configuration manual for ShardingSphere-JDBC.

Sharding

Background

SpringBoot Starter's data sharding configuration applies to business scenarios that use SpringBoot, which can maximize SpringBoot's capabilities, such as configuration initialization and Bean management. It can complete the creation of the ShardingSphereDataSource object and reduce unnecessary coding.

Parameters

```
spring.shardingsphere.datasource.names= # Omit the data source configuration,
please refer to the usage

# Standard sharding table configuration
spring.shardingsphere.rules.sharding.tables.<table-name>.actual-data-nodes= #
Describe data source names and actual tables, delimiter as point, multiple data
nodes separated with comma, support inline expression. Absent means sharding
databases only.

# Databases sharding strategy, use default databases sharding strategy if absent.
sharding strategy below can choose only one.

# For single sharding column scenario
spring.shardingsphere.rules.sharding.tables.<table-name>.database-strategy.
standard.sharding-column= # Sharding column name
spring.shardingsphere.rules.sharding.tables.<table-name>.database-strategy.
standard.sharding-algorithm-name= # Sharding algorithm name

# For multiple sharding columns scenario
spring.shardingsphere.rules.sharding.tables.<table-name>.database-strategy.complex.
sharding-columns= # Sharding column names, multiple columns separated with comma
spring.shardingsphere.rules.sharding.tables.<table-name>.database-strategy.complex.
sharding-algorithm-name= # Sharding algorithm name

# Sharding by hint
spring.shardingsphere.rules.sharding.tables.<table-name>.database-strategy.hint.
sharding-algorithm-name= # Sharding algorithm name

# Tables sharding strategy, same as database sharding strategy
spring.shardingsphere.rules.sharding.tables.<table-name>.table-strategy.xxx= #
```

```
Omitted

# Auto sharding table configuraiton
spring.shardingsphere.rules.sharding.auto-tables.<auto-table-name>.actual-data-
sources= # data source names

spring.shardingsphere.rules.sharding.auto-tables.<auto-table-name>.sharding-
strategy.standard.sharding-column= # Sharding column name
spring.shardingsphere.rules.sharding.auto-tables.<auto-table-name>.sharding-
strategy.standard.sharding-algorithm-name= # Auto sharding algorithm name

# Key generator strategy configuration
spring.shardingsphere.rules.sharding.tables.<table-name>.key-generate-strategy.
column= # Column name of key generator
spring.shardingsphere.rules.sharding.tables.<table-name>.key-generate-strategy.key-
generator-name= # Key generator name

spring.shardingsphere.rules.sharding.binding-tables[0]= # Binding table name
spring.shardingsphere.rules.sharding.binding-tables[1]= # Binding table name
spring.shardingsphere.rules.sharding.binding-tables[x]= # Binding table name

spring.shardingsphere.rules.sharding.broadcast-tables[0]= # Broadcast tables
spring.shardingsphere.rules.sharding.broadcast-tables[1]= # Broadcast tables
spring.shardingsphere.rules.sharding.broadcast-tables[x]= # Broadcast tables

spring.shardingsphere.rules.sharding.default-database-strategy.xxx= # Default
strategy for database sharding
spring.shardingsphere.rules.sharding.default-table-strategy.xxx= # Default strategy
for table sharding
spring.shardingsphere.rules.sharding.default-key-generate-strategy.xxx= # Default
Key generator strategy
spring.shardingsphere.rules.sharding.default-sharding-column= # Default sharding
column name

# Sharding algorithm configuration
spring.shardingsphere.rules.sharding.sharding-algorithms.<sharding-algorithm-name>.
type= # Sharding algorithm type
spring.shardingsphere.rules.sharding.sharding-algorithms.<sharding-algorithm-name>.
props.xxx=# Sharding algorithm properties

# Key generate algorithm configuration
spring.shardingsphere.rules.sharding.key-generators.<key-generate-algorithm-name>.
type= # Key generate algorithm type
spring.shardingsphere.rules.sharding.key-generators.<key-generate-algorithm-name>.
props.xxx= # Key generate algorithm properties
```

Please refer to Built-in Sharding Algorithm List and Built-in Key Generate Algorithm List for more details about type of algorithm.

Attention: Inline expression identifier can use `${...}` or `$->{...}`, but `${...}` is conflict with spring placeholder of properties, so use `$->{...}` on spring environment is better.

Procedure

1. Configure data sharding rules in the SpringBoot file, including data sources, sharding rules, and global attributes.
2. Start the SpringBoot program. The configuration is automatically loaded and the ShardingSphere-DataSource is initialized.

Sample

```
spring.shardingsphere.mode.type=Standalone
spring.shardingsphere.mode.repository.type=File
spring.shardingsphere.mode.overwrite=true

spring.shardingsphere.datasource.names=ds-0,ds-1

spring.shardingsphere.datasource.ds-0.jdbc-url=jdbc:mysql://localhost:3306/demo_ds_
0?serverTimezone=UTC&useSSL=false&useUnicode=true&characterEncoding=UTF-8
spring.shardingsphere.datasource.ds-0.type=com.zaxxer.hikari.HikariDataSource
spring.shardingsphere.datasource.ds-0.driver-class-name=com.mysql.jdbc.Driver
spring.shardingsphere.datasource.ds-0.username=root
spring.shardingsphere.datasource.ds-0.password=

spring.shardingsphere.datasource.ds-1.jdbc-url=jdbc:mysql://localhost:3306/demo_ds_
1?serverTimezone=UTC&useSSL=false&useUnicode=true&characterEncoding=UTF-8
spring.shardingsphere.datasource.ds-1.type=com.zaxxer.hikari.HikariDataSource
spring.shardingsphere.datasource.ds-1.driver-class-name=com.mysql.jdbc.Driver
spring.shardingsphere.datasource.ds-1.username=root
spring.shardingsphere.datasource.ds-1.password=

spring.shardingsphere.rules.sharding.default-database-strategy.standard.sharding-
column=user_id
spring.shardingsphere.rules.sharding.default-database-strategy.standard.sharding-
algorithm-name=database-inline
spring.shardingsphere.rules.sharding.binding-tables[0]=t_order,t_order_item
spring.shardingsphere.rules.sharding.broadcast-tables=t_address

spring.shardingsphere.rules.sharding.tables.t_order.actual-data-nodes=ds-$->{0..1}.
t_order_$->{0..1}
spring.shardingsphere.rules.sharding.tables.t_order.table-strategy.standard.
sharding-column=order_id
spring.shardingsphere.rules.sharding.tables.t_order.table-strategy.standard.
sharding-algorithm-name=t-order-inline
```

```
spring.shardingsphere.rules.sharding.tables.t_order.key-generate-strategy.
column=order_id
spring.shardingsphere.rules.sharding.tables.t_order.key-generate-strategy.key-
generator-name=snowflake

spring.shardingsphere.rules.sharding.tables.t_order_item.actual-data-nodes=ds-$->
{0..1}.t_order_item_$->{0..1}
spring.shardingsphere.rules.sharding.tables.t_order_item.table-strategy.standard.
sharding-column=order_id
spring.shardingsphere.rules.sharding.tables.t_order_item.table-strategy.standard.
sharding-algorithm-name=t-order-item-inline

spring.shardingsphere.rules.sharding.tables.t_order_item.key-generate-strategy.
column=order_item_id
spring.shardingsphere.rules.sharding.tables.t_order_item.key-generate-strategy.key-
generator-name=snowflake

spring.shardingsphere.rules.sharding.sharding-algorithms.database-inline.
type=INLINE
spring.shardingsphere.rules.sharding.sharding-algorithms.database-inline.props.
algorithm-expression=ds-$->{user_id % 2}
spring.shardingsphere.rules.sharding.sharding-algorithms.t-order-inline.type=INLINE
spring.shardingsphere.rules.sharding.sharding-algorithms.t-order-inline.props.
algorithm-expression=t_order_$->{order_id % 2}
spring.shardingsphere.rules.sharding.sharding-algorithms.t-order-item-inline.
type=INLINE
spring.shardingsphere.rules.sharding.sharding-algorithms.t-order-item-inline.props.
algorithm-expression=t_order_item_$->{order_id % 2}

spring.shardingsphere.rules.sharding.key-generators.snowflake.type=SNOWFLAKE
```

Related References

- Core Feature: Data Sharding
- Developer Guide: Data Sharding

Readwrite splitting

Background

The read-write splitting configuration method of Spring Boot Starter is suitable for business scenarios using SpringBoot and can maximize the capabilities of initializing SringBoot configuration process and bean management to complete the creation of ShardingSphereDataSource object, reducing unnecessary coding work.

Parameters Explained

Static Readwrite-splitting

```
spring.shardingsphere.datasource.names= # Omit the data source configuration,
please refer to the usage

spring.shardingsphere.rules.readwrite-splitting.data-sources.<readwrite-splitting-
data-source-name>.static-strategy.write-data-source-name= # Write data source name
spring.shardingsphere.rules.readwrite-splitting.data-sources.<readwrite-splitting-
data-source-name>.static-strategy.read-data-source-names= # Read data source names,
multiple data source names separated with comma
spring.shardingsphere.rules.readwrite-splitting.data-sources.<readwrite-splitting-
data-source-name>.load-balancer-name= # Load balance algorithm name

# Load balance algorithm configuration
spring.shardingsphere.rules.readwrite-splitting.load-balancers.<load-balance-
algorithm-name>.type= # Load balance algorithm type
spring.shardingsphere.rules.readwrite-splitting.load-balancers.<load-balance-
algorithm-name>.props.xxx= # Load balance algorithm properties
```

Dynamic Readwrite-splitting

```
spring.shardingsphere.datasource.names= # Omit the data source configuration,
please refer to the usage

spring.shardingsphere.rules.readwrite-splitting.data-sources.<readwrite-splitting-
data-source-name>.dynamic-strategy.auto-aware-data-source-name= # Database
discovery logic data source name
spring.shardingsphere.rules.readwrite-splitting.data-sources.<readwrite-splitting-
data-source-name>.dynamic-strategy.write-data-source-query-enabled= # All read data
source are offline, write data source whether the data source is responsible for
read traffic
spring.shardingsphere.rules.readwrite-splitting.data-sources.<readwrite-splitting-
data-source-name>.load-balancer-name= # Load balance algorithm name

# Load balance algorithm configuration
spring.shardingsphere.rules.readwrite-splitting.load-balancers.<load-balance-
algorithm-name>.type= # Load balance algorithm type
spring.shardingsphere.rules.readwrite-splitting.load-balancers.<load-balance-
algorithm-name>.props.xxx= # Load balance algorithm properties
```

Please refer to Built-in Load Balance Algorithm List for more details about type of algorithm. Please refer to Read-write splitting-Core features for more details about query consistent routing.

Operating Procedure

1. Add read/write splitting data source.
2. Set load-balancing algorithm.
3. Use read/write splitting data source.

Configuration Examples

```
spring.shardingsphere.rules.readwrite-splitting.data-sources.readwrite_ds.static-strategy.write-data-source-name=write-ds
spring.shardingsphere.rules.readwrite-splitting.data-sources.readwrite_ds.static-strategy.read-data-source-names=read-ds-0,read-ds-1
spring.shardingsphere.rules.readwrite-splitting.data-sources.readwrite_ds.load-balancer-name=round_robin
spring.shardingsphere.rules.readwrite-splitting.load-balancers.round_robin.type=ROUND_ROBIN
```

References

- Read-write splitting-Core features
- Java API: read-write splitting
- YAML Configuration: read-write splitting
- Spring namespace: read-write splitting

HA

Background

The `Spring Boot Starter` configuration method is applicable to business scenarios using SpringBoot. It can make full use of the SpringBoot configuration initialization and bean management capabilities, to automatically complete the creation of `ShardingSphereDataSource` objects.

Parameters

```
spring.shardingsphere.datasource.names= # Omit data source configuration, please refer to the user manual

spring.shardingsphere.rules.readwrite-splitting.data-sources.<readwrite-splitting-data-source-name>.dynamic-strategy.auto-aware-data-source-name= # Logical data source name discovered by the database
```

```
spring.shardingsphere.rules.database-discovery.data-sources.<database-discovery-data-source-name>.data-source-names= # Data source name. Multiple data sources are separated by commas, for example: ds_0, ds_1
spring.shardingsphere.rules.database-discovery.data-sources.<database-discovery-data-source-name>.discovery-heartbeat-name= # Detect heartbeat name
spring.shardingsphere.rules.database-discovery.data-sources.<database-discovery-data-source-name>.discovery-type-name= # Database discovery type name
spring.shardingsphere.rules.database-discovery.discovery-heartbeats.<discovery-heartbeat-name>.props.keep-alive-cron= # Cron expression, such as: '0/5 * * * * ?'
spring.shardingsphere.rules.database-discovery.discovery-types.<discovery-type-name>.type= # Database discovery type, such as: MySQL.MGR
spring.shardingsphere.rules.database-discovery.discovery-types.<discovery-type-name>.props.group-name= # Necessary parameters of database discovery type, such as group-name of MGR
```

Procedure

1. Import MAVEN dependency.

```
<dependency>
    <groupId>org.apache.shardingsphere</groupId>
    <artifactId>shardingsphere-jdbc-core-spring-boot-starter</artifactId>
    <version>${latest.release.version}</version>
</dependency>
```

Note: please change `${latest.release.version}' to the actual version number.

Sample

```
spring.shardingsphere.datasource.names=ds-0,ds-1,ds-2
spring.shardingsphere.datasource.ds-0.jdbc-url = jdbc:mysql://127.0.0.1:13306/primary_demo_ds?serverTimezone=UTC&useSSL=false
spring.shardingsphere.datasource.ds-0.username=root
spring.shardingsphere.datasource.ds-0.password=
spring.shardingsphere.datasource.ds-0.type=com.zaxxer.hikari.HikariDataSource
spring.shardingsphere.datasource.ds-0.driver-class-name=com.mysql.cj.jdbc.Driver

spring.shardingsphere.datasource.ds-1.jdbc-url = jdbc:mysql://127.0.0.1:13307/primary_demo_ds?serverTimezone=UTC&useSSL=false
spring.shardingsphere.datasource.ds-1.username=root
spring.shardingsphere.datasource.ds-1.password=
spring.shardingsphere.datasource.ds-1.type=com.zaxxer.hikari.HikariDataSource
spring.shardingsphere.datasource.ds-1.driver-class-name=com.mysql.cj.jdbc.Driver

spring.shardingsphere.datasource.ds-2.jdbc-url = jdbc:mysql://127.0.0.1:13308/primary_demo_ds?serverTimezone=UTC&useSSL=false
```

```
spring.shardingsphere.datasource.ds-2.username=root
spring.shardingsphere.datasource.ds-2.password=
spring.shardingsphere.datasource.ds-2.type=com.zaxxer.hikari.HikariDataSource
spring.shardingsphere.datasource.ds-2.driver-class-name=com.mysql.cj.jdbc.Driver

spring.shardingsphere.rules.readwrite-splitting.data-sources.replica_ds.dynamic-strategy.auto-aware-data-source-name=readwrite_ds

spring.shardingsphere.rules.database-discovery.data-sources.readwrite_ds.data-source-names=ds-0, ds-1, ds-2
spring.shardingsphere.rules.database-discovery.data-sources.readwrite_ds.discovery-heartbeat-name=mgr-heartbeat
spring.shardingsphere.rules.database-discovery.data-sources.readwrite_ds.discovery-type-name=mgr
spring.shardingsphere.rules.database-discovery.discovery-heartbeats.mgr-heartbeat.props.keep-alive-cron=0/5 * * * * ?
spring.shardingsphere.rules.database-discovery.discovery-types.mgr.type=MGR
spring.shardingsphere.rules.database-discovery.discovery-types.mgr.props.groupName=b13df29e-90b6-11e8-8d1b-525400fc3996
```

Related References

- Feature Description of HA
- JAVA API: HA
- YAML Configuration: HA
- Spring Namespace: HA

Encryption

Background

The configuration method for Spring Boot Starter Data Encryption is suitable for business scenarios using SpringBoot and can make the most of SringBoot’s configuration initialization and Bean management capabilities to complete the creation of ShardingSphereDataSource objects, reducing unnecessary coding work.

Parameters

```
spring.shardingsphere.datasource.names= # Omit the data source configuration,
please refer to the usage

spring.shardingsphere.rules.encrypt.tables.<table-name>.query-with-cipher-column= #
Whether the table uses cipher columns for query
spring.shardingsphere.rules.encrypt.tables.<table-name>.columns.<column-name>.
cipher-column= # Cipher column name
spring.shardingsphere.rules.encrypt.tables.<table-name>.columns.<column-name>.
assisted-query-column= # Assisted query column name
spring.shardingsphere.rules.encrypt.tables.<table-name>.columns.<column-name>.
plain-column= # Plain column name
spring.shardingsphere.rules.encrypt.tables.<table-name>.columns.<column-name>.
encryptor-name= # Encrypt algorithm name

# Encrypt algorithm configuration
spring.shardingsphere.rules.encrypt.encryptors.<encrypt-algorithm-name>.type= #
Encrypt algorithm type
spring.shardingsphere.rules.encrypt.encryptors.<encrypt-algorithm-name>.props.xxx=
# Encrypt algorithm properties

spring.shardingsphere.rules.encrypt.queryWithCipherColumn= # Whether query with
cipher column for data encrypt. User you can use plaintext to query if have
```

Please refer to Built-in Encrypt Algorithm List for more details about type of algorithm.

Procedure

1. Configure the data encryption rules in the SpringBoot file, including the data source, encryption rules, global properties and other items.
2. Start the SpringBoot program, which will automatically load the configuration and initialize the ShardingSphereDataSource.

Sample

```
spring.shardingsphere.datasource.names=ds

spring.shardingsphere.datasource.ds.type=com.zaxxer.hikari.HikariDataSource
spring.shardingsphere.datasource.ds.driver-class-name=com.mysql.jdbc.Driver
spring.shardingsphere.datasource.ds.jdbc-url=jdbc:mysql://localhost:3306/demo_ds?
serverTimezone=UTC&useSSL=false&useUnicode=true&characterEncoding=UTF-8
spring.shardingsphere.datasource.ds.username=root
spring.shardingsphere.datasource.ds.password=
```

```
spring.shardingsphere.rules.encrypt.encryptors.name-encryptor.type=AES
spring.shardingsphere.rules.encrypt.encryptors.name-encryptor.props.aes-key-
value=123456abc
spring.shardingsphere.rules.encrypt.encryptors.pwd-encryptor.type=AES
spring.shardingsphere.rules.encrypt.encryptors.pwd-encryptor.props.aes-key-
value=123456abc

spring.shardingsphere.rules.encrypt.tables.t_user.columns.username.cipher-
column=username
spring.shardingsphere.rules.encrypt.tables.t_user.columns.username.encryptor-
name=name-encryptor
spring.shardingsphere.rules.encrypt.tables.t_user.columns.pwd.cipher-column=pwd
spring.shardingsphere.rules.encrypt.tables.t_user.columns.pwd.encryptor-name=pwd-
encryptor

spring.shardingsphere.props.query-with-cipher-column=true
spring.shardingsphere.props.sql-show=true
```

Related References

- Core Feature: Data Encryption
- Developer Guide: Data Encryption

Shadow DB

Background

If you want to use the ShardingSphere Shadow DB feature in the Spring Boot environment, please refer to the following configuration.

Parameters

```
spring.shardingsphere.datasource.names= # Omit the data source configuration,
please refer to the usage

spring.shardingsphere.rules.shadow.data-sources.shadow-data-source.production-data-
source-name= # Production data source name
spring.shardingsphere.rules.shadow.data-sources.shadow-data-source.shadow-data-
source-name= # Shadow data source name

spring.shardingsphere.rules.shadow.tables.<table-name>.data-source-names= # Shadow
table location shadow data source names (multiple values are separated by ",")
spring.shardingsphere.rules.shadow.tables.<table-name>.shadow-algorithm-names= #
Shadow table location shadow algorithm names (multiple values are separated by ",")
```

```
spring.shardingsphere.rules.shadow.defaultShadowAlgorithmName= # Default shadow
algorithm name, optional item.

spring.shardingsphere.rules.shadow.shadow-algorithms.<shadow-algorithm-name>.type=
# Shadow algorithm type
spring.shardingsphere.rules.shadow.shadow-algorithms.<shadow-algorithm-name>.props.
xxx= # Shadow algorithm property configuration
```

For details, see list of built-in shadow algorithms

Procedure

1. Configure the shadow library rules in the SpringBoot file, including configuration items such as data sources, shadow rules, and global properties.
2. Start the SpringBoot program, the configuration will be loaded automatically, and the ShardingSphereDataSource will be initialized.

Sample

```
spring.shardingsphere.datasource.names=ds,shadow-ds

spring.shardingsphere.datasource.ds.jdbc-url=jdbc:mysql://localhost:3306/ds?
serverTimezone=UTC&useSSL=false&useUnicode=true&characterEncoding=UTF-8
spring.shardingsphere.datasource.ds.type=com.zaxxer.hikari.HikariDataSource
spring.shardingsphere.datasource.ds.driver-class-name=com.mysql.jdbc.Driver
spring.shardingsphere.datasource.ds.username=root
spring.shardingsphere.datasource.ds.password=

spring.shardingsphere.datasource.shadow-ds.jdbc-url=jdbc:mysql://localhost:3306/
shadow_ds?serverTimezone=UTC&useSSL=false&useUnicode=true&characterEncoding=UTF-8
spring.shardingsphere.datasource.shadow-ds.type=com.zaxxer.hikari.HikariDataSource
spring.shardingsphere.datasource.shadow-ds.driver-class-name=com.mysql.jdbc.Driver
spring.shardingsphere.datasource.shadow-ds.username=root
spring.shardingsphere.datasource.shadow-ds.password=

spring.shardingsphere.rules.shadow.data-sources.shadow-data-source.production-data-
source-name=ds
spring.shardingsphere.rules.shadow.data-sources.shadow-data-source.shadow-data-
source-name=shadow-ds

spring.shardingsphere.rules.shadow.tables.t_user.data-source-names=shadow-data-
source
spring.shardingsphere.rules.shadow.tables.t_user.shadow-algorithm-names=user-id-
insert-match-algorithm,simple-hint-algorithm
```

```
spring.shardingsphere.rules.shadow.shadow-algorithms.user-id-insert-match-
algorithm.type=VALUE_MATCH
spring.shardingsphere.rules.shadow.shadow-algorithms.user-id-insert-match-
algorithm.props.operation=insert
spring.shardingsphere.rules.shadow.shadow-algorithms.user-id-insert-match-
algorithm.props.column=user_id
spring.shardingsphere.rules.shadow.shadow-algorithms.user-id-insert-match-
algorithm.props.value=1

spring.shardingsphere.rules.shadow.shadow-algorithms.simple-hint-algorithm.
type=SIMPLE_HINT
spring.shardingsphere.rules.shadow.shadow-algorithms.simple-hint-algorithm.props.
shadow=true
spring.shardingsphere.rules.shadow.shadow-algorithms.simple-hint-algorithm.props.
foo=bar
```

Related References

- Feature Description of Shadow DB
- JAVA API: Shadow DB
- YAML Configuration: Shadow DB
- Spring Namespace: Shadow DB
- Dev Guide: Shadow DB

SQL Parser

Background

The configuration method of Spring Boot Starter is applicable to business scenarios using SpringBoot. In this way, the SpringBoot configuration initialization and bean management capabilities can be used to the greatest extent, so as to simplify code development.

Parameters

```
spring.shardingsphere.rules.sql-parser.sql-comment-parse-enabled= # Whether to
parse SQL comments

spring.shardingsphere.rules.sql-parser.sql-statement-cache.initial-capacity= #
Initial capacity of SQL statement local cache
spring.shardingsphere.rules.sql-parser.sql-statement-cache.maximum-size= # Maximum
capacity of SQL statement local cache
```

```
spring.shardingsphere.rules.sql-parser.parse-tree-cache.initial-capacity= # Initial
capacity of parse tree local cache
spring.shardingsphere.rules.sql-parser.parse-tree-cache.maximum-size= # Maximum
local cache capacity of parse tree
```

Procedure

1. Set local cache configuration
2. Set parser configuration
3. use the parser engine to parse SQL

Sample

```
spring.shardingsphere.rules.sql-parser.sql-comment-parse-enabled=true

spring.shardingsphere.rules.sql-parser.sql-statement-cache.initial-capacity=2000
spring.shardingsphere.rules.sql-parser.sql-statement-cache.maximum-size=65535

spring.shardingsphere.rules.sql-parser.parse-tree-cache.initial-capacity=128
spring.shardingsphere.rules.sql-parser.parse-tree-cache.maximum-size=1024
```

Related References

- JAVA API: SQL Parser
- YAML Configuration: SQL Parser
- Spring Namespace: SQL Parser

Mixed Rules

Background

ShardingSphere provides a variety of features, such as data sharding, read/write splitting, high availability, and data decryption. These features can be used independently or in combination.

Below, you will find the parameters' explanation and configuration samples based on SpringBoot Starter.

Parameters

```
spring.shardingsphere.datasource.names= # Please refer to the user manual for the
data source configuration
# Standard sharding table configuration
spring.shardingsphere.rules.sharding.tables.<table-name>.actual-data-nodes= # It
consists of data source name plus table name, separated by decimal points. Multiple
tables are separated by commas, and inline expression is supported. By default, a
data node is generated with a known data source and logical table name, used for
broadcast tables (that is, each database needs the same table for associated
queries, mostly the dictionary table) or the situation when only database sharding
is needed and all databases have the same table structure.
# Standard sharding scenarios used for a single shard key
spring.shardingsphere.rules.sharding.tables.<table-name>.database-strategy.
standard.sharding-column= # Sharding column name
spring.shardingsphere.rules.sharding.tables.<table-name>.database-strategy.
standard.sharding-algorithm-name= # sharding algorithm name
# Table shards strategy. The same as database shards strategy
spring.shardingsphere.rules.sharding.tables.<table-name>.table-strategy.xxx= # Omit
# Distributed sequence strategy configuration
spring.shardingsphere.rules.sharding.tables.<table-name>.key-generate-strategy.
column= # Distributed sequence column name
spring.shardingsphere.rules.sharding.tables.<table-name>.key-generate-strategy.key-
generator-name= # Distributed sequence algorithm name
# Sharding algorithm configuration
spring.shardingsphere.rules.sharding.sharding-algorithms.<sharding-algorithm-name>.
type= # Sharding algorithm type
spring.shardingsphere.rules.sharding.sharding-algorithms.<sharding-algorithm-name>.
props.xxx= # Sharidng algorithm property configuration
# Distributed sequence algorithm configuration
spring.shardingsphere.rules.sharding.key-generators.<key-generate-algorithm-name>.
type= # Distributed sequence algorithm type
spring.shardingsphere.rules.sharding.key-generators.<key-generate-algorithm-name>.
props.xxx= # Property configuration of distributed sequence algorithm
# Dynamic read/write splitting configuration
spring.shardingsphere.rules.readwrite-splitting.data-sources.<readwrite-splitting-
data-source-name>.dynamic-strategy.auto-aware-data-source-name= # logical data
source name of database discovery
spring.shardingsphere.rules.readwrite-splitting.data-sources.<readwrite-splitting-
data-source-name>.dynamic-strategy.write-data-source-query-enabled= # All the read
databases went offline. Whether the primary database bears the read traffic.
spring.shardingsphere.rules.readwrite-splitting.data-sources.<readwrite-splitting-
data-source-name>.load-balancer-name= # Load balancer algorithm name
# Database discovery configuration
spring.shardingsphere.rules.database-discovery.data-sources.<database-discovery-
data-source-name>.data-source-names= # Data source name. Multiple data sources are
separated by commas, such as ds_0, ds_1.
spring.shardingsphere.rules.database-discovery.data-sources.<database-discovery-
```

```
data-source-name>.discovery-heartbeat-name= # Detect heartbeat name
spring.shardingsphere.rules.database-discovery.data-sources.<database-discovery-
data-source-name>.discovery-type-name= # Database discovery type name
spring.shardingsphere.rules.database-discovery.discovery-heartbeats.<discovery-
heartbeat-name>.props.keep-alive-cron= # cron expression, such as '0/5 * * * ?'.
spring.shardingsphere.rules.database-discovery.discovery-types.<discovery-type-
name>.type= # Database discovery type, such as MySQL.MGR.
spring.shardingsphere.rules.database-discovery.discovery-types.<discovery-type-
name>.props.group-name= # Required parameter of database discovery type, such as
MGR's group-name.
# Data desensitization configuration
spring.shardingsphere.rules.encrypt.tables.<table-name>.query-with-cipher-column= #
Whether the table uses ciphercolumn for queries.
spring.shardingsphere.rules.encrypt.tables.<table-name>.columns.<column-name>.
cipher-column= # Ciphercolumn name
spring.shardingsphere.rules.encrypt.tables.<table-name>.columns.<column-name>.
assisted-query-column= # Query column name
spring.shardingsphere.rules.encrypt.tables.<table-name>.columns.<column-name>.
plain-column= # Plaincolumn name
spring.shardingsphere.rules.encrypt.tables.<table-name>.columns.<column-name>.
encryptor-name= # Encryption algorithm name
# Encryption algorithm configuration
spring.shardingsphere.rules.encrypt.encryptors.<encrypt-algorithm-name>.type= #
Encryption algorithm type
spring.shardingsphere.rules.encrypt.encryptors.<encrypt-algorithm-name>.props.xxx=
# Encryption algorithm property configuration
spring.shardingsphere.rules.encrypt.queryWithCipherColumn= # Whether use
ciphercolumn for queries. You can use the plaincolumn for queries if it's
available.
```

Samples

```
# Sharding configuration
spring.shardingsphere.rules.sharding.tables.t_order.actual-data-nodes=replica-ds-$-
>{0..1}.t_order_$->{0..1}
spring.shardingsphere.rules.sharding.tables.t_order.table-strategy.standard.
sharding-column=order_id
spring.shardingsphere.rules.sharding.tables.t_order.table-strategy.standard.
sharding-algorithm-name=t-order-inline
spring.shardingsphere.rules.sharding.tables.t_order.key-generate-strategy.
column=order_id
spring.shardingsphere.rules.sharding.tables.t_order.key-generate-strategy.key-
generator-name=snowflake
spring.shardingsphere.rules.sharding.tables.t_order_item.actual-data-nodes=replica-
ds-$->{0..1}.t_order_item_$->{0..1}
spring.shardingsphere.rules.sharding.tables.t_order_item.table-strategy.standard.
```

```
sharding-column=order_id
spring.shardingsphere.rules.sharding.sharding-algorithms.database-inline.
type=INLINE
spring.shardingsphere.rules.sharding.sharding-algorithms.database-inline.props.
algorithm-expression=replica_ds-$->{user_id % 2}
spring.shardingsphere.rules.sharding.sharding-algorithms.t-order-inline.type=INLINE
spring.shardingsphere.rules.sharding.sharding-algorithms.t-order-inline.props.
algorithm-expression=t_order_$->{order_id % 2}
spring.shardingsphere.rules.sharding.key-generators.snowflake.type=SNOWFLAKE
# Dynamic read/write splitting configuration
spring.shardingsphere.rules.readwrite-splitting.data-sources.replica-ds-0.dynamic-
strategy.auto-aware-data-source-name=readwrite-ds-0
spring.shardingsphere.rules.readwrite-splitting.data-sources.replica-ds-1.dynamic-
strategy.auto-aware-data-source-name=readwrite-ds-1
# Database discovery configuration
spring.shardingsphere.rules.database-discovery.data-sources.readwrite-ds-0.data-
source-names=ds-0, ds-1, ds-2
spring.shardingsphere.rules.database-discovery.data-sources.readwrite-ds-0.
discovery-heartbeat-name=mgr-heartbeat
spring.shardingsphere.rules.database-discovery.data-sources.readwrite-ds-0.
discovery-type-name=mgr
spring.shardingsphere.rules.database-discovery.data-sources.readwrite-ds-1.data-
source-names=ds-3, ds-4, ds-5
spring.shardingsphere.rules.database-discovery.data-sources.readwrite-ds-1.
discovery-heartbeat-name=mgr-heartbeat
spring.shardingsphere.rules.database-discovery.data-sources.readwrite-ds-1.
discovery-type-name=mgr
spring.shardingsphere.rules.database-discovery.discovery-heartbeats.mgr-heartbeat.
props.keep-alive-cron=0/5 * * * * ?
spring.shardingsphere.rules.database-discovery.discovery-types.mgr.type=MGR
spring.shardingsphere.rules.database-discovery.discovery-types.mgr.props.
groupName=b13df29e-90b6-11e8-8d1b-525400fc3996
# Data decryption
spring.shardingsphere.rules.encrypt.encryptors.name-encryptor.type=AES
spring.shardingsphere.rules.encrypt.encryptors.name-encryptor.props.aes-key-
value=123456abc
spring.shardingsphere.rules.encrypt.encryptors.pwd-encryptor.type=AES
spring.shardingsphere.rules.encrypt.encryptors.pwd-encryptor.props.aes-key-
value=123456abc
spring.shardingsphere.rules.encrypt.tables.t_user.columns.username.cipher-
column=username
spring.shardingsphere.rules.encrypt.tables.t_user.columns.username.encryptor-
name=name-encryptor
spring.shardingsphere.rules.encrypt.tables.t_user.columns.pwd.cipher-column=pwd
spring.shardingsphere.rules.encrypt.tables.t_user.columns.pwd.encryptor-name=pwd-
encryptor
spring.shardingsphere.props.query-with-cipher-column=true
spring.shardingsphere.props.sql-show=true
```

Algorithm

Sharding

```
# sharding-algorithm-name is specified by users and its property should be
consistent with that of sharding-algorithm-name in the sharding strategy.
# type and props, please refer to the built-in sharding algorithm: https://
shardingsphere.apache.org/document/current/en/user-manual/common-config/builtin-
algorithm/sharding/
spring.shardingsphere.rules.sharding.sharding-algorithms.<sharding-algorithm-name>.
type=xxx
spring.shardingsphere.rules.sharding.sharding-algorithms.<sharding-algorithm-name>.
props.xxx=xxx
```

Encryption

```
# encrypt-algorithm-name is specified by users, and its property should be
consistent with that of encryptor-name in encryption rules.
# type and props, please refer to the built-in encryption algorithm: https://
shardingsphere.apache.org/document/current/en/user-manual/common-config/builtin-
algorithm/encrypt/
spring.shardingsphere.rules.encrypt.encryptors.<encrypt-algorithm-name>.type=xxx
spring.shardingsphere.rules.encrypt.encryptors.<encrypt-algorithm-name>.props.
xxx=xxx
```

Read/Write Splitting Load Balancer

```
# load-balance-algorithm-name is specified by users, and its property has to be
consistent with that of load-balancer-name in read/write splitting rules.
# type and props, please refer to the built-in read/write splitting algorithm load
balancer: https://shardingsphere.apache.org/document/current/en/user-manual/common-
config/builtin-algorithm/load-balance/
spring.shardingsphere.rules.readwrite-splitting.load-balancers.<load-balance-
algorithm-name>.type=xxx
spring.shardingsphere.rules.readwrite-splitting.load-balancers.<load-balance-
algorithm-name>.props.xxx=xxx
```

Shadow DB

```
# shadow-algorithm-name is specified by users, and its property has to be
consistent with that of shadow-algorithm-names in shadow DB rules.
# type and props, please refer to the built-in shadow DB algorithm: https://
shardingsphere.apache.org/document/current/en/user-manual/common-config/builtin-
algorithm/shadow/
spring.shardingsphere.rules.shadow.shadow-algorithms.<shadow-algorithm-name>.
type=xxx
spring.shardingsphere.rules.shadow.shadow-algorithms.<shadow-algorithm-name>.props.
xxx=xxx
```

High Availability

```
# discovery-type-name is specified by users, and its property has to be consistent
with that of discovery-type-name in database discovery rules.
spring.shardingsphere.rules.database-discovery.discovery-types.<discovery-type-
name>.type=xxx
spring.shardingsphere.rules.database-discovery.discovery-types.<discovery-type-
name>.props.xxx=xxx
```

6.1.4 Spring Namespace

Overview

ShardingSphere-JDBC provides official Spring Namespace to make convenient for developers to integrate ShardingSphere-JDBC and Spring.

Usage

Import Maven Dependency

```
<dependency>
    <groupId>org.apache.shardingsphere</groupId>
    <artifactId>shardingsphere-jdbc-core-spring-namespace</artifactId>
    <version>${shardingsphere.version}</version>
</dependency>
```

Configure Spring Bean

Configuration Item Explanation

Namespace: http://shardingsphere.apache.org/schema/shardingsphere/datasource/datasource-5.1.2.xsd

<shardingsphere:data-source />

N ame	• Type*	*Description*
id	Attribute	Spring Bean Id
d atab ase- name (?)	Attribute	JDBC data source alias
d ata- sour ce-n ames	Attribute	Data source name, multiple data source names are separated by commas
r ule- refs	Attribute	Rule name, multiple rule names are separated by commas
mode (?)	Tag	Mode configuration
p rops (?)	Tag	Properties configuration, Please refer to Properties Configuration for more details

Example

```
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:shardingsphere="http://shardingsphere.apache.org/schema/
shardingsphere/datasource"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
                           http://www.springframework.org/schema/beans/spring-
beans.xsd
                           http://shardingsphere.apache.org/schema/shardingsphere/
datasource
                           http://shardingsphere.apache.org/schema/shardingsphere/
datasource/datasource.xsd
                           ">
    <shardingsphere:data-source id="ds" database-name="foo_schema" data-source-
names="..." rule-refs="...">
        <shardingsphere:mode type="..." />
        <props>
            <prop key="xxx.xxx">${xxx.xxx}</prop>
        </props>
    </shardingsphere:data-source>
```

```
</beans>
```

Use Data Source

Same with Spring Boot Starter.

Mode

Background

The default configuration uses standalone mode.

Parameters Explained

Standalone Mode

Namespace:http://shardingsphere.apache.org/schema/shardingsphere/mode-repository/standalone/repository-5.1.1.xsd

Name	*Type*	*Description*
id	Property	Persistent repository Bean name
type	Property	Persistent repository Type
props (?)	Tag	Properties required for persistent repository

Cluster Mode(Recommended)

Namespace: http://shardingsphere.apache.org/schema/shardingsphere/mode-repository/cluster/repository-5.1.1.xsd

Name	*Type*	*Description*
id	Property	Persistent repository Bean name
type	Property	Persistent repository Type
namespace	Property	Registry Center namespace
server-lists	Property	Registry Center Link
props (?)	Tag	Properties required for persistent repository

Tips:

1. For production environments, it is recommended to use cluster mode deployment.
2. For cluster mode deployment, it is recommended to use `ZooKeeper` registry.

Operating Procedures

Introduce MAVEN dependency

```
<dependency>
    <groupId>org.apache.shardingsphere</groupId>
    <artifactId>shardingsphere-jdbc-core-spring-namespace</artifactId>
    <version>${latest.release.version}</version>
</dependency>
```

Note: Please change `${latest.release.version}` to the actual version number.

Configuration Example

Standalone Mode

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:shardingsphere="http://shardingsphere.apache.org/schema/
shardingsphere/datasource"
       xmlns:standalone="http://shardingsphere.apache.org/schema/shardingsphere/
mode-repository/standalone"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
                           http://www.springframework.org/schema/beans/spring-
beans.xsd
                           http://shardingsphere.apache.org/schema/shardingsphere/
datasource
                  http://shardingsphere.apache.org/schema/shardingsphere/
datasource/datasource.xsd
                           http://shardingsphere.apache.org/schema/shardingsphere/
mode-repository/standalone
                           http://shardingsphere.apache.org/schema/shardingsphere/
mode-repository/standalone/repository.xsd">
    <standalone:repository id="standaloneRepository" type="File">
        <props>
            <prop key="path">.shardingsphere</prop>
        </props>
    </standalone:repository>

    <shardingsphere:data-source id="ds" database-name="foo_db" data-source-names=".
```

```
.." rule-refs="..." >
        <shardingsphere:mode type="Standalone" repository-ref="standaloneRepository
" overwrite="false" />
    </shardingsphere:data-source>
</beans>
```

Cluster Mode

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:shardingsphere="http://shardingsphere.apache.org/schema/
shardingsphere/datasource"
       xmlns:cluster="http://shardingsphere.apache.org/schema/shardingsphere/mode-
repository/cluster"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
                           http://www.springframework.org/schema/beans/spring-
beans.xsd
                           http://shardingsphere.apache.org/schema/shardingsphere/
datasource
                           http://shardingsphere.apache.org/schema/shardingsphere/
datasource/datasource.xsd
                           http://shardingsphere.apache.org/schema/shardingsphere/
mode-repository/cluster
                           http://shardingsphere.apache.org/schema/shardingsphere/
mode-repository/cluster/repository.xsd">
    <cluster:repository id="clusterRepository" type="Zookeeper" namespace=
"regCenter" server-lists="localhost:3182">
        <props>
            <prop key="max-retries">3</prop>
            <prop key="operation-timeout-milliseconds">1000</prop>
        </props>
    </cluster:repository>

    <shardingsphere:data-source id="ds" database-name="foo_db" data-source-names=".
.." rule-refs="...">
        <shardingsphere:mode type="Cluster" repository-ref="clusterRepository"
overwrite="false" />
    </shardingsphere:data-source>
</beans>
```

Relevant References

- Installation and use of ZooKeeper Registry Center
- For details about persistent repository, please refer to List of Built-in repository types

Data Source

Background

Any data source object configured as Spring bean can be used with the Spring namespace of ShardingSphere-JDBC Data Planning.

The database driver in the example is MySQL and the connection pool is HikariCP, both of which can be replaced by other database drivers and connection pools. When using ShardingSphere JDBC, the property names of the JDBC pools depend on the definition of JDBC pools themselves respectively, rather than being rigidly defined by ShardingSphere. For relevant processing, you can see reference class `org.apache.shardingsphere.infra.datasource.pool.creator.DataSourcePoolCreator`. As for Alibaba Druid 1.2.9, using `url` instead of `jdbcUrl` as in the following example is the expected behavior.

Configuration Examples

```
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:shardingsphere="http://shardingsphere.apache.org/schema/
shardingsphere/datasource"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
                           http://www.springframework.org/schema/beans/spring-
beans.xsd
                           http://shardingsphere.apache.org/schema/shardingsphere/
datasource
                           http://shardingsphere.apache.org/schema/shardingsphere/
datasource/datasource.xsd
                           ">
    <bean id="ds1" class="com.zaxxer.hikari.HikariDataSource" destroy-method="close
">
        <property name="driverClassName" value="com.mysql.jdbc.Driver" />
        <property name="jdbcUrl" value="jdbc:mysql://localhost:3306/ds1" />
        <property name="username" value="root" />
        <property name="password" value="" />
    </bean>

    <bean id="ds2" class="com.zaxxer.hikari.HikariDataSource" destroy-method="close
">
        <property name="driverClassName" value="com.mysql.jdbc.Driver" />
        <property name="jdbcUrl" value="jdbc:mysql://localhost:3306/ds2" />
```

```xml
        <property name="username" value="root" />
        <property name="password" value="" />
    </bean>

    <shardingsphere:data-source id="ds" database-name="foo_schema" data-source-
names="ds1,ds2" rule-refs="..." />
</beans>
```

Rules

Rules are pluggable part of Apache ShardingSphere. This chapter is a Spring namespace rule configuration manual for ShardingSphere-JDBC.

Sharding

Background

The configuration method of data sharding Spring Namespace is applicable to traditional Spring projects. The sharding rules and attributes are configured through the namespace xml configuration file. Spring completes the creation and management of ShardingSphereDataSource objects to avoid additional coding work.

Parameters

Namespace: http://shardingsphere.apache.org/schema/shardingsphere/sharding/sharding-5.1.2.xsd

<sharding:rule />

Name	*T ype*	*Description*
id	A ttri bute	Spring Bean Id
table-rules (?)	Tag	Sharding table rule configuration
auto-table-rules (?)	Tag	Automatic sharding table rule configuration
binding-table-rules (?)	Tag	Binding table rule configuration
broadcast-table-rules (?)	Tag	Broadcast table rule configuration
def ault-database-strategy-ref (?)	A ttri bute	Default database strategy name
default-table-strategy-ref (?)	A ttri bute	Default table strategy name
default -key-generate-strategy-ref (?)	A ttri bute	Default key generate strategy name
default-sharding-column (?)	A ttri bute	Default sharding column name

<sharding:table-rule />

Name	*Type*	*Description*
logic-table	At trib- ute	Logic table name
actual- data-nodes	At trib- ute	Describe data source names and actual tables, delimiter as point, multiple data nodes separated with comma, support inline expression. Absent means shard- ing databases only.
actual- data- sources	At trib- ute	Data source names for auto sharding table
database- strategy- ref	At trib- ute	Database strategy name for standard sharding table
table- strategy- ref	At trib- ute	Table strategy name for standard sharding table
sharding- strategy- ref	At trib- ute	sharding strategy name for auto sharding table
key- generate- strategy- ref	At trib- ute	Key generate strategy name

<sharding:binding-table-rules />

Name	*Type*	*Description*
binding-table-rule (+)	Tag	Binding table rule configuration

<sharding:binding-table-rule />

Name	• Type*	*Description*
logi c-tables	Attr ibute	Binding table name, multiple tables separated with comma

<sharding:broadcast-table-rules />

Name	*Type*	*Description*
broadcast-table-rule (+)	Tag	Broadcast table rule configuration

<sharding:broadcast-table-rule />

Name	*Type*	*Description*
table	Attribute	Broadcast table name

<sharding:standard-strategy />

Name	*Type*	*Description*
id	Attribute	Standard sharding strategy name
sharding-column	Attribute	Sharding column name
algorithm-ref	Attribute	Sharding algorithm name

<sharding:complex-strategy />

Name	*T ype*	*Description*
id	A ttri bute	Complex sharding strategy name
shardi ng-columns	A ttri bute	Sharding column names, multiple columns separated with comma
alg orithm-ref	A ttri bute	Sharding algorithm name

<sharding:hint-strategy />

Name	*Type*	*Description*
id	Attribute	Hint sharding strategy name
algorithm-ref	Attribute	Sharding algorithm name

<sharding:none-strategy />

Name	*Type*	*Description*
id	Attribute	Sharding strategy name

<sharding:key-generate-strategy />

Name	*Type*	*Description*
id	Attribute	Key generate strategy name
column	Attribute	Key generate column name
algorithm-ref	Attribute	Key generate algorithm name

<sharding:sharding-algorithm />

Name	*Type*	*Description*
id	Attribute	Sharding algorithm name
type	Attribute	Sharding algorithm type
props (?)	Tag	Sharding algorithm properties

<sharding:key-generate-algorithm />

Name	*Type*	*Description*
id	Attribute	Key generate algorithm name
type	Attribute	Key generate algorithm type
props (?)	Tag	Key generate algorithm properties

Please refer to Built-in Sharding Algorithm List and Built-in Key Generate Algorithm List for more details about type of algorithm.

> Attention: Inline expression identifier can use `${...}` or `$->{...}`, but `${...}` is conflict with spring placeholder of properties, so use `$->{...}` on spring environment is better.

Procedure

1. Configure data sharding rules in the Spring Namespace configuration file, including data source, sharding rules, global attributes and other configuration items.
2. Start the Spring program, the configuration will be loaded automatically, and the ShardingSphere-DataSource will be initialized.

Sample

```
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:tx="http://www.springframework.org/schema/tx"
       xmlns:shardingsphere="http://shardingsphere.apache.org/schema/
shardingsphere/datasource"
       xmlns:sharding="http://shardingsphere.apache.org/schema/shardingsphere/
sharding"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
                           http://www.springframework.org/schema/beans/spring-
beans.xsd
                           http://www.springframework.org/schema/tx
                           http://www.springframework.org/schema/tx/spring-tx.xsd
                           http://www.springframework.org/schema/context
                           http://www.springframework.org/schema/context/spring-
context.xsd
                           http://shardingsphere.apache.org/schema/shardingsphere/
datasource
                           http://shardingsphere.apache.org/schema/shardingsphere/
datasource/datasource.xsd
                           http://shardingsphere.apache.org/schema/shardingsphere/
sharding
                           http://shardingsphere.apache.org/schema/shardingsphere/
```

```
sharding/sharding.xsd
                           ">
    <context:component-scan base-package="org.apache.shardingsphere.example.core.
mybatis" />

    <bean id="demo_ds_0" class="com.zaxxer.hikari.HikariDataSource" destroy-method=
"close">
        <property name="driverClassName" value="com.mysql.jdbc.Driver"/>
        <property name="jdbcUrl" value="jdbc:mysql://localhost:3306/demo_ds_0?
serverTimezone=UTC&amp;useSSL=false&amp;useUnicode=true&amp;characterEncoding=UTF-8
"/>
        <property name="username" value="root"/>
        <property name="password" value=""/>
    </bean>

    <bean id="demo_ds_1" class="com.zaxxer.hikari.HikariDataSource" destroy-method=
"close">
        <property name="driverClassName" value="com.mysql.jdbc.Driver"/>
        <property name="jdbcUrl" value="jdbc:mysql://localhost:3306/demo_ds_1?
serverTimezone=UTC&amp;useSSL=false&amp;useUnicode=true&amp;characterEncoding=UTF-8
"/>
        <property name="username" value="root"/>
        <property name="password" value=""/>
    </bean>

    <sharding:standard-strategy id="databaseStrategy" sharding-column="user_id"
algorithm-ref="inlineStrategyShardingAlgorithm" />

    <sharding:sharding-algorithm id="inlineStrategyShardingAlgorithm" type="INLINE
">
        <props>
            <prop key="algorithm-expression">demo_ds_${user_id % 2}</prop>
        </props>
    </sharding:sharding-algorithm>

    <sharding:key-generate-algorithm id="snowflakeAlgorithm" type="SNOWFLAKE">
    </sharding:key-generate-algorithm>

    <sharding:key-generate-strategy id="orderKeyGenerator" column="order_id"
algorithm-ref="snowflakeAlgorithm" />
    <sharding:key-generate-strategy id="itemKeyGenerator" column="order_item_id"
algorithm-ref="snowflakeAlgorithm" />

    <sharding:rule id="shardingRule">
        <sharding:table-rules>
            <sharding:table-rule logic-table="t_order" database-strategy-ref=
"databaseStrategy" key-generate-strategy-ref="orderKeyGenerator" />
            <sharding:table-rule logic-table="t_order_item" database-strategy-ref=
```

```
"databaseStrategy" key-generate-strategy-ref="itemKeyGenerator" />
        </sharding:table-rules>
        <sharding:binding-table-rules>
            <sharding:binding-table-rule logic-tables="t_order,t_order_item"/>
        </sharding:binding-table-rules>
        <sharding:broadcast-table-rules>
            <sharding:broadcast-table-rule table="t_address"/>
        </sharding:broadcast-table-rules>
    </sharding:rule>

    <shardingsphere:data-source id="shardingDataSource" database-name="sharding-
databases" data-source-names="demo_ds_0, demo_ds_1" rule-refs="shardingRule" />

    <bean id="transactionManager" class="org.springframework.jdbc.datasource.
DataSourceTransactionManager">
        <property name="dataSource" ref="shardingDataSource" />
    </bean>
    <tx:annotation-driven />

    <bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">
        <property name="dataSource" ref="shardingDataSource"/>
        <property name="mapperLocations" value="classpath*:META-INF/mappers/*.xml"/
>
    </bean>

    <bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">
        <property name="basePackage" value="org.apache.shardingsphere.example.core.
mybatis.repository"/>
        <property name="sqlSessionFactoryBeanName" value="sqlSessionFactory"/>
    </bean>
</beans>
```

Related References

- Core Feature: Data Sharding
- Developer Guide: Data Sharding

Readwrite-splitting

Background

Spring namespace read/write splitting configuration method is suitable for conventional Spring projects, determine sharding rules and properties through namespace XML configuration files, and let Spring do the creation and management of ShardingSphereDataSource objects, avoiding additional coding work.

Parameters Explained

Namespace: http://shardingsphere.apache.org/schema/shardingsphere/readwrite-splitting/readwrite-splitting-5.1.2.xsd

<readwrite-splitting:rule />

Name	• Type*	*Description*
id	Attr ibute	Spring Bean Id
data-source-rule (+)	Tag	Readwrite-splitting data source rule configuration

<readwrite-splitting:data-source-rule />

Name	*Type*	*Description*
id	Attribute	Readwrite-splitting data source rule name
static-strategy	Tag	Static Readwrite-splitting type
dynamic-strategy	Tag	Dynamic Readwrite-splitting type
l oad-balance-algorithm-ref	Attribute	Load balance algorithm name

<readwrite-splitting:static-strategy />

Name	• Type*	*Description*
id	Attribute	Static readwrite-splitting name
write-d ata-source-name	Attribute	Write data source name
read-da ta-source-names	Attribute	Read data source names, multiple data source names separated with comma
load-balanc e-algorithm-ref	Attribute	Load balance algorithm name

<readwrite-splitting:dynamic-strategy />

Name	• Type*	*Description*
id	Attribute	Dynamic readwrite-splitting name
aut o-aware-data -source-name	Attribute	Database discovery logic data source name
write-d ata-source-q uery-enabled	Attribute	All read data source are offline, write data source whether the data source is responsible for read traffic
lo ad-balance-a lgorithm-ref	Attribute	Load balance algorithm name

<readwrite-splitting:load-balance-algorithm />

Name	*Type*	*Description*
id	Attribute	Load balance algorithm name
type	Attribute	Load balance algorithm type
props (?)	Tag	Load balance algorithm properties

Please refer to Built-in Load Balance Algorithm List for more details about type of algorithm. Please refer to Read-write splitting-Core features for more details about query consistent routing.

Operating Procedures

1. Add read/write splitting data source.
2. Set the load balancing algorithm.
3. Using read/write splitting data sources.

Configuration Example

```
<readwrite-splitting:load-balance-algorithm id="randomStrategy" type="RANDOM" />

<readwrite-splitting:rule id="readWriteSplittingRule">
    <readwrite-splitting:data-source-rule id="demo_ds" load-balance-algorithm-ref=
"randomStrategy">
        <readwrite-splitting:static-strategy id="staticStrategy" write-data-source-
name="demo_write_ds" read-data-source-names="demo_read_ds_0, demo_read_ds_1"/>
    </readwrite-splitting:data-source-rule>
</readwrite-splitting:rule>

<shardingsphere:data-source id="readWriteSplittingDataSource" data-source-names=
"demo_write_ds, demo_read_ds_0, demo_read_ds_1" rule-refs="readWriteSplittingRule"
/>
```

Related References

- Read-write splitting-Core features
- Java API: read-write splitting
- YAML Configuration: read-write splitting
- Spring Boot Starter: read-write splitting

HA

Background

The Spring namespace configuration method, applicable to traditional Spring projects, configures highly availability rules by means of namespace XML configuration files, and Spring completes the creation and management of ShardingSphereDataSource objects.

Parameters Explained

Namespace: http://shardingsphere.apache.org/schema/shardingsphere/database-discovery/database-discovery-5.1.1.xsd

<database-discovery:rule />

Name	*Type*	*Description*
id	Property	Spring Bean Id
data-source-rule (+)	Tag	Configuration of data source rules
discovery-heartbeat (+)	Tag	Configuration of heartbeat rules detection

<database-discovery:data-source-rule />

Name	• Type*	*Description*
id	Property	Data source rules name
data-source-names	Property	Data source name, multiple datasources are divided by comma,such as：ds_0, ds_1
d iscovery-heartbeat-name	Property	Detect heartbeat name
discovery-type-name	Property	type name of database discovery

<database-discovery:discovery-heartbeat />

Name	• Type*	*Description*
id	Property	heartbeat listen name
props	标 | property configuration of heartbeat 签 | listen, cron expression of keep-alive-cron property configuration, such as: ‘0/5 * * * * ?’	

<database-discovery:discovery-type />

Name	*Ty pe*	*Description*
id	Pr ope rty	Type name of database discovery
type	Pr ope rty	Database discovery type, such as: MySQL.MGR
props (?)	Tag	Configuration of database discovery type, such as group-name property configuration of MGR

Operating Procedures

1. Introduce Maven dependency

```
<dependency>
    <groupId>org.apache.shardingsphere</groupId>
    <artifactId>shardingsphere-jdbc-core-spring-namespace</artifactId>
    <version>${latest.release.version}</version>
</dependency>
```

Configuration Example

```
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:cluster="http://shardingsphere.apache.org/schema/shardingsphere/mode-
repository/cluster"
       xmlns:shardingsphere="http://shardingsphere.apache.org/schema/
shardingsphere/datasource"
       xmlns:database-discovery="http://shardingsphere.apache.org/schema/
shardingsphere/database-discovery"
       xmlns:readwrite-splitting="http://shardingsphere.apache.org/schema/
shardingsphere/readwrite-splitting"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
```

```
                           http://www.springframework.org/schema/beans/spring-
beans.xsd
                           http://shardingsphere.apache.org/schema/shardingsphere/
database-discovery
                           http://shardingsphere.apache.org/schema/shardingsphere/
database-discovery/database-discovery.xsd
                           http://shardingsphere.apache.org/schema/shardingsphere/
readwrite-splitting
                           http://shardingsphere.apache.org/schema/shardingsphere/
readwrite-splitting/readwrite-splitting.xsd
                           http://shardingsphere.apache.org/schema/shardingsphere/
mode-repository/cluster
                           http://shardingsphere.apache.org/schema/shardingsphere/
mode-repository/cluster/repository.xsd
                           http://shardingsphere.apache.org/schema/shardingsphere/
datasource
                           http://shardingsphere.apache.org/schema/shardingsphere/
datasource/datasource.xsd
                           ">
    <bean id="ds_0" class="com.zaxxer.hikari.HikariDataSource" destroy-method=
"close">
        <property name="driverClassName" value="com.mysql.jdbc.Driver" />
        <property name="jdbcUrl" value="jdbc:mysql://127.0.0.1:33306/primary_demo_
ds?serverTimezone=UTC&amp;useSSL=false&amp;useUnicode=true&amp;
characterEncoding=UTF-8" />
        <property name="username" value="root" />
        <property name="password" value="" />
    </bean>
    <bean id="ds_1" class="com.zaxxer.hikari.HikariDataSource" destroy-method=
"close">
        <property name="driverClassName" value="com.mysql.jdbc.Driver"/>
        <property name="jdbcUrl" value="jdbc:mysql://127.0.0.1:33307/primary_demo_
ds?serverTimezone=UTC&amp;useSSL=false&amp;useUnicode=true&amp;
characterEncoding=UTF-8" />
        <property name="username" value="root" />
        <property name="password" value="" />
    </bean>
    <bean id="ds_2" class="com.zaxxer.hikari.HikariDataSource" destroy-method=
"close">
        <property name="driverClassName" value="com.mysql.jdbc.Driver"/>
        <property name="jdbcUrl" value="jdbc:mysql://127.0.0.1:33308/primary_demo_
ds?useSSL=false"/>
        <property name="username" value="root"/>
        <property name="password" value=""/>
    </bean>
    <cluster:repository id="clusterRepository" type="ZooKeeper" namespace=
"governance" server-lists="localhost:2181">
        <props>
```

```
            <prop key="max-retries">3</prop>
            <prop key="operation-timeout-milliseconds">3000</prop>
        </props>
    </cluster:repository>
    <readwrite-splitting:rule id="readWriteSplittingRule">
        <readwrite-splitting:data-source-rule id="replica_ds">
            <readwrite-splitting:dynamic-strategy id="dynamicStrategy" auto-aware-
data-source-name="readwrite_ds" />
        </readwrite-splitting:data-source-rule>
    </readwrite-splitting:rule>
    <database-discovery:rule id="mgrDatabaseDiscoveryRule">
        <database-discovery:data-source-rule id="readwrite_ds" data-source-names=
"ds_0,ds_1,ds_2" discovery-heartbeat-name="mgr-heartbeat" discovery-type-name="mgr"
/>
        <database-discovery:discovery-heartbeat id="mgr-heartbeat">
            <props>
                <prop key="keep-alive-cron" >0/5 * * * * ?</prop>
            </props>
        </database-discovery:discovery-heartbeat>
    </database-discovery:rule>
    <database-discovery:discovery-type id="mgr" type="MySQL.MGR">
        <props>
            <prop key="group-name">558edd3c-02ec-11ea-9bb3-080027e39bd2</prop>
        </props>
    </database-discovery:discovery-type>
    <shardingsphere:data-source id="databaseDiscoveryDataSource" schema-name=
"database-discovery-db" data-source-names="ds_0, ds_1, ds_2" rule-refs=
"readWriteSplittingRule, mgrDatabaseDiscoveryRule">
    <shardingsphere:mode repository-ref="clusterRepository" type="Cluster" />
    </shardingsphere:data-source>
</beans>
```

Related References

- Feature Description of HA
- JAVA API: HA
- YAML Configuration: HA
- Spring Boot Starter: HA

Encryption

Background

Spring Namespace' s data encryption configuration applies to the traditional Spring projects. Sharding rules and attributes are configured through the XML configuration file of the namespace. Spring creates and manages the ShardingSphereDataSource object, reducing unnecessary coding.

Parameters

Namespace: http://shardingsphere.apache.org/schema/shardingsphere/encrypt/encrypt-5.1.2.xsd

<encrypt:rule />

Name	• *Type**	*Description*	*Def ault Va lue*
id	Attribute	Spring Bean Id	
que ryWithCip herColumn (?)	Attribute	Whether query with cipher column for data encrypt. User you can use plaintext to query if have	true
table (+)	Tag	Encrypt table configuration	

<encrypt:table />

Name	• *Type**	*Description*
name	Attribute	Encrypt table name
column (+)	Tag	Encrypt column configuration
que ry-with-ciph er-column(?) (?)	Attribute	Whether the table query with cipher column for data encrypt. User you can use plaintext to query if have

<encrypt:column />

Name	*Type*	*Description*
logic-column	Attribute	Column logic name
cipher-column	Attribute	Cipher column name
assisted-query-column (?)	Attribute	Assisted query column name
plain-column (?)	Attribute	Plain column name
encrypt-algorithm-ref	Attribute	Encrypt algorithm name

<encrypt:encrypt-algorithm />

Name	*Type*	*Description*
id	Attribute	Encrypt algorithm name
type	Attribute	Encrypt algorithm type
props (?)	Tag	Encrypt algorithm properties

Please refer to Built-in Encrypt Algorithm List for more details about type of algorithm.

Procedure

1. Configure data encryption rules in the Spring namespace configuration file, including data sources, encryption rules, and global attributes.
2. Start the Spring program, and it will automatically load the configuration and initialize the ShardingSphereDataSource.

Sample

```
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:shardingsphere="http://shardingsphere.apache.org/schema/
shardingsphere/datasource"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:tx="http://www.springframework.org/schema/tx"
       xmlns:encrypt="http://shardingsphere.apache.org/schema/shardingsphere/
encrypt"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
                           http://www.springframework.org/schema/beans/spring-
beans.xsd
                           http://www.springframework.org/schema/tx
                           http://www.springframework.org/schema/tx/spring-tx.xsd
                           http://www.springframework.org/schema/context
                           http://www.springframework.org/schema/context/spring-
context.xsd
                           http://shardingsphere.apache.org/schema/shardingsphere/
datasource
                           http://shardingsphere.apache.org/schema/shardingsphere/
datasource/datasource.xsd
                           http://shardingsphere.apache.org/schema/shardingsphere/
encrypt
                           http://shardingsphere.apache.org/schema/shardingsphere/
encrypt/encrypt.xsd
                           ">
    <context:component-scan base-package="org.apache.shardingsphere.example.core.
mybatis" />
```

```xml
    <bean id="ds" class="com.zaxxer.hikari.HikariDataSource" destroy-method="close
">
        <property name="driverClassName" value="com.mysql.jdbc.Driver"/>
        <property name="jdbcUrl" value="jdbc:mysql://localhost:3306/demo_ds?
serverTimezone=UTC&amp;useSSL=false&amp;useUnicode=true&amp;characterEncoding=UTF-8
"/>
        <property name="username" value="root"/>
        <property name="password" value=""/>
    </bean>

    <encrypt:encrypt-algorithm id="name_encryptor" type="AES">
        <props>
            <prop key="aes-key-value">123456</prop>
        </props>
    </encrypt:encrypt-algorithm>
    <encrypt:encrypt-algorithm id="pwd_encryptor" type="assistedTest" />

    <encrypt:rule id="encryptRule">
        <encrypt:table name="t_user">
            <encrypt:column logic-column="username" cipher-column="username" plain-
column="username_plain" encrypt-algorithm-ref="name_encryptor" />
            <encrypt:column logic-column="pwd" cipher-column="pwd" assisted-query-
column="assisted_query_pwd" encrypt-algorithm-ref="pwd_encryptor" />
        </encrypt:table>
    </encrypt:rule>

    <shardingsphere:data-source id="encryptDataSource" data-source-names="ds" rule-
refs="encryptRule" />

    <bean id="transactionManager" class="org.springframework.jdbc.datasource.
DataSourceTransactionManager">
        <property name="dataSource" ref="encryptDataSource" />
    </bean>
    <tx:annotation-driven />

    <bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">
        <property name="dataSource" ref="encryptDataSource"/>
        <property name="mapperLocations" value="classpath*:META-INF/mappers/*.xml"/
>
    </bean>

    <bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">
        <property name="basePackage" value="org.apache.shardingsphere.example.core.
mybatis.repository"/>
        <property name="sqlSessionFactoryBeanName" value="sqlSessionFactory"/>
    </bean>
</beans>
```

Related References

- Core Feature: Data Encryption
- Developer Guide: Data Encryption

Shadow DB

Background

Under the distributed application architecture based on microservices, the business needs multiple services to be completed through a series of service and middleware calls, so the stress test of a single service can no longer represent the real scenario. In the test environment, rebuilding a complete set of pressure test environments similar to the production environment would mean an excessively high cost, and often an inability to simulate the complexity and flow of the online environment. Therefore, enterprises usually select the full link voltage test method, i.e. a pressure test in the production environment, so that the test results can accurately reflect the system's real capacity and performance level.

Parameters

Configuration Entry

```
<shadow:rule />
```

Configurable Properties:

Name	*Type*	*Description*
id	Attribute	Spring Bean Id
data-source(?)	Tag	Shadow data source configuration
shadow-table(?)	Tag	Shadow table configuration
shadow-algorithm(?)	Tag	Shadow table configuration
default-shado w-algorithm-name(?)	Tag	Default shadow algorithm configuration

Shadow data source configuration:

```
<shadow:data-source />
```

Name	*Type*	*Description*
id	Attribute	Spring Bean Id
production-data-source-name	Attribute	Production data source name
shadow-data-source-name	Attribute	Shadow data source name

Shadow table configuration:

```
<shadow:shadow-table />
```

Name	*Type*	*Description*
name	Attribute	Shadow table name
data-sources	Attribute	Shadow table associated shadow data source name list (multiple values are separated by ",")
algorithm (?)	Tag	Shadow table association shadow algorithm configuration

```
<shadow:algorithm />
```

Name	*Type*	*Description*
s hadow-algorithm-ref	Attribute	Shadow table association shadow algorithm name

Shadow algorithm configuration:

```
<shadow:shadow-algorithm />
```

Name	*Type*	*Description*
id	Attribute	Shadow algorithm name
type	Attribute	Shadow algorithm type
props (?)	Tag	Shadow algorithm attribute configuration

Refer to Builin Shadow Algorithm for details

Procedure

1. Create production and shadow data sources.
2. Configure shadow rules.
   - Configure shadow data sources.
   - Configure shadow table.
   - Configure shadow algorithm.

Sample

```xml
<beans xmlns="http://www.springframework.org/schema/beans" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:shadow="http://shardingsphere.apache.org/schema/shardingsphere/shadow" xsi:schemaLocation="http://www.springframework.org/schema/beans
                           http://www.springframework.org/schema/beans/spring-beans.xsd
                           http://shardingsphere.apache.org/schema/shardingsphere/shadow
                           http://shardingsphere.apache.org/schema/shardingsphere/shadow/shadow.xsd
                           ">
    <shadow:shadow-algorithm id="user-id-insert-match-algorithm" type="VALUE_MATCH">
        <props>
            <prop key="operation">insert</prop>
            <prop key="column">user_id</prop>
            <prop key="value">1</prop>
        </props>
    </shadow:shadow-algorithm>

    <shadow:rule id="shadowRule">
        <shadow:data-source id="shadow-data-source" production-data-source-name="ds" shadow-data-source-name="ds_shadow"/>
        <shadow:shadow-table name="t_user" data-sources="shadow-data-source">
            <shadow:algorithm shadow-algorithm-ref="user-id-insert-match-algorithm" />
        </shadow:shadow-table>
    </shadow:rule>
</beans>
```

Related References

- Feature Description of Shadow DB
- JAVA API: Shadow DB
- YAML Configuration: Shadow DB
- Spring Namespace: Shadow DB
- Dev Guide: Shadow DB

SQL Parser

Background

Spring namespace's SQL parser configuration applies to traditional Spring projects. SQL parsing rules and attributes can be configured through the XML configuration files of the namespace.

Parameters

Namespace: http://shardingsphere.apache.org/schema/shardingsphere/sql-parser/sql-parser-5.1.2.xsd

<sql-parser:rule />

Name	*Type*	*Description*
id	Attribute	Spring Bean Id
sql-comment-parse-enable	Attribute	Whether to parse SQL comments
parse-tree-cache-ref	Attribute	Parse tree local cache name
sql-statement-cache-ref	Attribute	SQL statement local cache name

<sql-parser:cache-option />

Name	*Type*	*Description*
id	Attribute	Local cache configuration item name
initial-capacity	Attribute	Initial capacity of local cache
maximum-size	Attribute	Maximum capacity of local cache

Procedure

1. Set local cache configuration.
2. Set parser configuration.
3. Parse SQL with a parsing engine.

Sample

```
<sql-parser:rule id="sqlParseRule" sql-comment-parse-enable="true" parse-tree-
cache-ref="parseTreeCache" sql-statement-cache-ref="sqlStatementCache" />
<sql-parser:cache-option id="sqlStatementCache" initial-capacity="1024" maximum-
size="1024"/>
<sql-parser:cache-option id="parseTreeCache" initial-capacity="1024" maximum-size=
"1024"/>
```

Related References

- JAVA API: SQL Parser
- YAML Configuration: SQL Parser
- Spring Boot Starter: SQL Parser

Mixed Rules

Background

ShardingSphere provides a variety of features, such as data sharding, read/write splitting, high availability, and data decryption. These features can be used independently or in combination.

Below, you will find the configuration samples based on Spring Namespace.

Samples

```
<!-- Sharding configuration -->
<sharding:standard-strategy id="databaseStrategy" sharding-column="user_id"
algorithm-ref="inlineStrategyShardingAlgorithm" />
<sharding:sharding-algorithm id="inlineStrategyShardingAlgorithm" type="INLINE">
    <props>
        <prop key="algorithm-expression">replica_ds_${user_id % 2}</prop>
    </props>
</sharding:sharding-algorithm>
<sharding:key-generate-algorithm id="snowflakeAlgorithm" type="SNOWFLAKE">
</sharding:key-generate-algorithm>
```

```xml
<sharding:key-generate-strategy id="orderKeyGenerator" column="order_id" algorithm-
ref="snowflakeAlgorithm" />
<sharding:rule id="shardingRule">
    <sharding:table-rules>
        <sharding:table-rule logic-table="t_order" database-strategy-ref=
"databaseStrategy" key-generate-strategy-ref="orderKeyGenerator" />
    </sharding:table-rules>
</sharding:rule>

<!-- Dynamic read/write splitting configuration -->
<readwrite-splitting:rule id="readWriteSplittingRule">
    <readwrite-splitting:data-source-rule id="replica_ds_0">
        <readwrite-splitting:dynamic-strategy id="dynamicStrategy" auto-aware-data-
source-name="readwrite_ds_0" />
    </readwrite-splitting:data-source-rule>
    <readwrite-splitting:data-source-rule id="replica_ds_1">
        <readwrite-splitting:dynamic-strategy id="dynamicStrategy" auto-aware-data-
source-name="readwrite_ds_1" />
    </readwrite-splitting:data-source-rule>
</readwrite-splitting:rule>

<!-- Database discovery configuration -->
<database-discovery:rule id="mgrDatabaseDiscoveryRule">
    <database-discovery:data-source-rule id="readwrite_ds_0" data-source-names="ds_
0,ds_1,ds_2" discovery-heartbeat-name="mgr-heartbeat" discovery-type-name="mgr" />
    <database-discovery:data-source-rule id="readwrite_ds_1" data-source-names="ds_
3,ds_4,ds_5" discovery-heartbeat-name="mgr-heartbeat" discovery-type-name="mgr" />
    <database-discovery:discovery-heartbeat id="mgr-heartbeat">
        <props>
            <prop key="keep-alive-cron" >0/5 * * * * ?</prop>
        </props>
    </database-discovery:discovery-heartbeat>
</database-discovery:rule>
<database-discovery:discovery-type id="mgr" type="MySQL.MGR">
    <props>
        <prop key="group-name">558edd3c-02ec-11ea-9bb3-080027e39bd2</prop>
    </props>
</database-discovery:discovery-type>

<!-- Data decryption configuration -->
<encrypt:encrypt-algorithm id="name_encryptor" type="AES">
    <props>
        <prop key="aes-key-value">123456</prop>
    </props>
</encrypt:encrypt-algorithm>
<encrypt:encrypt-algorithm id="pwd_encryptor" type="assistedTest" />

<encrypt:rule id="encryptRule">
```

```xml
    <encrypt:table name="t_user">
        <encrypt:column logic-column="username" cipher-column="username" plain-column="username_plain" encrypt-algorithm-ref="name_encryptor" />
        <encrypt:column logic-column="pwd" cipher-column="pwd" assisted-query-column="assisted_query_pwd" encrypt-algorithm-ref="pwd_encryptor" />
    </encrypt:table>
</encrypt:rule>
```

Algorithm

Sharding

```xml
<!-- algorithmName is specified by users and its property should be consistent with that of algorithm-ref in the sharding strategy. -->
<!-- type and props, please refer to the built-in sharding algorithm: https://shardingsphere.apache.org/document/current/en/user-manual/common-config/builtin-algorithm/sharding/ -->
<sharding:sharding-algorithm id="algorithmName" type="xxx">
    <props>
        <prop key="xxx">xxx</prop>
    </props>
</sharding:sharding-algorithm>
```

Encryption

```xml
<!-- encryptorName is specified by users, and its property should be consistent with that of encrypt-algorithm-ref in encryption rules. -->
<!-- type and props, please refer to the built-in encryption algorithm: https://shardingsphere.apache.org/document/current/en/user-manual/common-config/builtin-algorithm/encrypt/ -->
<encrypt:encrypt-algorithm id="encryptorName" type="xxx">
    <props>
        <prop key="xxx">xxx</prop>
    </props>
</encrypt:encrypt-algorithm>
```

Read/Write Splitting Load Balancer

```
<!-- loadBalancerName is specified by users, and its property has to be consistent
with that of load-balance-algorithm-ref in read/write splitting rules. -->
<!-- type and props, please refer to the built-in read/write splitting algorithm
load balancer: https://shardingsphere.apache.org/document/current/en/user-manual/
common-config/builtin-algorithm/load-balance/ -->
<readwrite-splitting:load-balance-algorithm id="loadBalancerName" type="xxx">
    <props>
        <prop key="xxx">xxx</prop>
    </props>
</readwrite-splitting:load-balance-algorithm>
```

Shadow DB

```
<!-- shadowAlgorithmName is specified by users, and its property has to be
consistent with that of shadow-algorithm-ref in shadow DB rules. -->
<!-- type and props, please refer to the built-in shadow DB algorithm: https://
shardingsphere.apache.org/document/current/en/user-manual/common-config/builtin-
algorithm/shadow/ -->
<shadow:shadow-algorithm id="shadowAlgorithmName" type="xxx">
    <props>
        <prop key="xxx">xxx</prop>
    </props>
</shadow:shadow-algorithm>
```

High Availability

```
<!-- discoveryTypeName is specified by users, and its property has to be consistent
with that of discovery-type-name in database discovery rules. -->
<database-discovery:discovery-type id="discoveryTypeName" type="xxx">
    <props>
        <prop key="xxx">xxx</prop>
    </props>
</database-discovery:discovery-type>
```

6.1.5 Special API

This chapter will introduce the special API of ShardingSphere-JDBC.

Sharding

This chapter will introduce the Sharding API of ShardingSphere-JDBC.

Hint

Background

Apache ShardingSphere uses ThreadLocal to manage sharding key values for mandatory routing. A sharding value can be added by programming to the HintManager that takes effect only within the current thread. Apache ShardingSphere can also do mandatory routing by adding comments to SQL.

Main application scenarios for Hint: - The sharding fields do not exist in the SQL and database table structure but in the external business logic. - Certain data operations are forced to be performed in given databases.

Procedure

1. Call HintManager.getInstance() to obtain an instance of HintManager.
2. Use HintManager.addDatabaseShardingValue, HintManager.addTableShardingValue to set the sharding key value.
3. Execute SQL statements to complete routing and execution.
4. Call HintManager.close to clean up the contents of ThreadLocal.

Sample

Sharding with Hint

Hint Configuration

Hint algorithms require users to implement the interface of `org.apache.shardingsphere.api.sharding.hint.HintShardingAlgorithm`. Apache ShardingSphere will acquire sharding values from HintManager to route.

Take the following configurations for reference:

```
rules:
- !SHARDING
  tables:
    t_order:
```

```
      actualDataNodes: demo_ds_${0..1}.t_order_${0..1}
      databaseStrategy:
        hint:
          algorithmClassName: xxx.xxx.xxx.HintXXXAlgorithm
      tableStrategy:
        hint:
          algorithmClassName: xxx.xxx.xxx.HintXXXAlgorithm
  defaultTableStrategy:
    none:
  defaultKeyGenerateStrategy:
    type: SNOWFLAKE
    column: order_id

props:
    sql-show: true
```

Get HintManager

```
HintManager hintManager = HintManager.getInstance();
```

Add Sharding Value

- Use `hintManager.addDatabaseShardingValue` to add sharding key value of data source.
- Use `hintManager.addTableShardingValue` to add sharding key value of table.

  Users can use `hintManager.setDatabaseShardingValue` to add sharding in hint route to some certain sharding database without sharding tables.

Clean Hint Values

Sharding values are saved in `ThreadLocal`, so it is necessary to use `hintManager.close()` to clean `ThreadLocal`.

``HintManager`` has implemented ``AutoCloseable``. We recommend to close it automatically with ``try with resource``.

Codes:

```
// Sharding database and table with HintManager
String sql = "SELECT * FROM t_order";
try (HintManager hintManager = HintManager.getInstance();
     Connection conn = dataSource.getConnection();
     PreparedStatement preparedStatement = conn.prepareStatement(sql)) {
    hintManager.addDatabaseShardingValue("t_order", 1);
    hintManager.addTableShardingValue("t_order", 2);
    try (ResultSet rs = preparedStatement.executeQuery()) {
        while (rs.next()) {
            // ...
        }
    }
}

// Sharding database and one database route with HintManager
String sql = "SELECT * FROM t_order";
try (HintManager hintManager = HintManager.getInstance();
     Connection conn = dataSource.getConnection();
     PreparedStatement preparedStatement = conn.prepareStatement(sql)) {
    hintManager.setDatabaseShardingValue(3);
    try (ResultSet rs = preparedStatement.executeQuery()) {
        while (rs.next()) {
            // ...
        }
    }
}
```

Related References

- Core Feature: Data Sharding
- Developer Guide: Data Sharding

Readwrite Splitting

This chapter will introduce the Readwrite Splitting API of ShardingSphere-JDBC.

Hint

Background

Apache ShardingSphere uses ThreadLocal to manage primary database routing marks for mandatory routing. A primary database routing mark can be added to HintManager through programming, and this value is valid only in the current thread. Apache ShardingSphere can also route the primary database by adding comments to SQL.

Hint is mainly used to perform mandatory data operations in the primary database under the read/write splitting scenarios.

Procedure

1. Call `HintManager.getInstance()` to obtain HintManager instance.
2. Call `HintManager.setWriteRouteOnly()` method to set the primary database routing marks.
3. Execute SQL statements to complete routing and execution.
4. Call `HintManager.close()` to clear the content of ThreadLocal.

Sample

Primary Route with Hint

Use manual programming

Get HintManager

Be the same as sharding based on hint.

Configure Primary Database Route

- Use `hintManager.setWriteRouteOnly` to configure primary database route.

Clean Hint Value

Be the same as data sharding based on hint.

Codes:

```
String sql = "SELECT * FROM t_order";
try (HintManager hintManager = HintManager.getInstance();
     Connection conn = dataSource.getConnection();
     PreparedStatement preparedStatement = conn.prepareStatement(sql)) {
    hintManager.setWriteRouteOnly();
    try (ResultSet rs = preparedStatement.executeQuery()) {
        while (rs.next()) {
            // ...
        }
    }
}
```

Use special SQL comments

Terms of Use

To use SQL Hint function, users need to set `sqlCommentParseEnabled` to `true`. The comment format only supports `/* */` for now. The content needs to start with `ShardingSphere hint:`, and the attribute name needs to be `writeRouteOnly`.

Codes:

```
/* ShardingSphere hint: writeRouteOnly=true */
SELECT * FROM t_order;
```

Related References

- Core Feature: Readwrite Splitting
- Developer Guide: Readwrite Splitting

Transaction

Using distributed transaction through Apache ShardingSphere is no different from local transaction. In addition to transparent use of distributed transaction, Apache ShardingSphere can switch distributed transaction types every time the database accesses.

Supported transaction types include local, XA and BASE. It can be set before creating a database connection, and default value can be set when Apache ShardingSphere startup.

Use Java API

Background

With ShardingSphere-JDBC, XA and BASE mode transactions can be used through the API.

Prerequisites

Introducing Maven dependency

```
<dependency>
    <groupId>org.apache.shardingsphere</groupId>
    <artifactId>shardingsphere-jdbc-core</artifactId>
    <version>${shardingsphere.version}</version>
</dependency>

<!-- This module is required when using XA transactions -->
<dependency>
    <groupId>org.apache.shardingsphere</groupId>
    <artifactId>shardingsphere-transaction-xa-core</artifactId>
    <version>${shardingsphere.version}</version>
</dependency>

<!-- This module is required when using XA's Narayana mode -->
<dependency>
    <groupId>org.apache.shardingsphere</groupId>
    <artifactId>shardingsphere-transaction-xa-narayana</artifactId>
    <version>${project.version}</version>
</dependency>

<!-- This module is required when using BASE transactions -->
<dependency>
    <groupId>org.apache.shardingsphere</groupId>
    <artifactId>shardingsphere-transaction-base-seata-at</artifactId>
    <version>${shardingsphere.version}</version>
</dependency>
```

Procedure

1. Set the transaction type
2. Perform the business logic

Sample

```
TransactionTypeHolder.set(TransactionType.XA); // support TransactionType.LOCAL,
TransactionType.XA, TransactionType.BASE
        try (Connection conn = dataSource.getConnection()) { // use
ShardingSphereDataSource
        conn.setAutoCommit(false);
        PreparedStatement ps = conn.prepareStatement("INSERT INTO t_order (user_id,
status) VALUES (?, ?)");
        ps.setObject(1, 1000);
        ps.setObject(2, "init");
        ps.executeUpdate();
        conn.commit();
        }
```

Use Spring Boot Starter

Background

ShardingSphere-JDBC can be used through spring boot starter. ## Prerequisites

Introducing Maven dependency

```
<dependency>
    <groupId>org.apache.shardingsphere</groupId>
    <artifactId>shardingsphere-jdbc-core-spring-boot-starter</artifactId>
    <version>${shardingsphere.version}</version>
</dependency>

<!-- This module is required when using XA transactions -->
<dependency>
    <groupId>org.apache.shardingsphere</groupId>
    <artifactId>shardingsphere-transaction-xa-core</artifactId>
    <version>${shardingsphere.version}</version>
</dependency>

<!-- This module is required when using XA's Narayana mode -->
<dependency>
    <groupId>org.apache.shardingsphere</groupId>
    <artifactId>shardingsphere-transaction-xa-narayana</artifactId>
    <version>${project.version}</version>
```

```xml
</dependency>

<!-- This module is required when using BASE transactions -->
<dependency>
    <groupId>org.apache.shardingsphere</groupId>
    <artifactId>shardingsphere-transaction-base-seata-at</artifactId>
    <version>${shardingsphere.version}</version>
</dependency>
```

Procedure

1. Configure the transaction Type
2. Use distributed transactions

Sample

Configure the transaction Type

```java
@Configuration
@EnableTransactionManagement
public class TransactionConfiguration {

    @Bean
    public PlatformTransactionManager txManager(final DataSource dataSource) {
        return new DataSourceTransactionManager(dataSource);
    }

    @Bean
    public JdbcTemplate jdbcTemplate(final DataSource dataSource) {
        return new JdbcTemplate(dataSource);
    }
}
```

Use distributed transactions

```java
@Transactional
@ShardingSphereTransactionType(TransactionType.XA)  // 支持 TransactionType.LOCAL,
TransactionType.XA, TransactionType.BASE
public void insert() {
        jdbcTemplate.execute("INSERT INTO t_order (user_id, status) VALUES (?, ?)",
(PreparedStatementCallback<Object>) ps -> {
        ps.setObject(1, i);
        ps.setObject(2, "init");
```

```
        ps.executeUpdate();
        });
        }
```

Use Spring Namespace

Background

ShardingSphere-JDBC can be used through spring namespace.

Prerequisites

Introducing Maven denpendency

```xml
<dependency>
    <groupId>org.apache.shardingsphere</groupId>
    <artifactId>shardingsphere-jdbc-core-spring-namespace</artifactId>
    <version>${shardingsphere.version}</version>
</dependency>

<!-- This module is required when using XA transactions -->
<dependency>
    <groupId>org.apache.shardingsphere</groupId>
    <artifactId>shardingsphere-transaction-xa-core</artifactId>
    <version>${shardingsphere.version}</version>
</dependency>

<!-- This module is required when using XA's Narayana mode -->
<dependency>
    <groupId>org.apache.shardingsphere</groupId>
    <artifactId>shardingsphere-transaction-xa-narayana</artifactId>
    <version>${project.version}</version>
</dependency>

<!-- This module is required when using BASE transactions -->
<dependency>
    <groupId>org.apache.shardingsphere</groupId>
    <artifactId>shardingsphere-transaction-base-seata-at</artifactId>
    <version>${shardingsphere.version}</version>
</dependency>
```

Procedure

1. Configure the transaction manager
2. Use distributed transactions

Sample

Configure the transaction manager

```
<!-- Configuration of ShardingDataSource -->
<!-- ...  -->

<bean id="transactionManager" class="org.springframework.jdbc.datasource.
DataSourceTransactionManager">
    <property name="dataSource" ref="shardingDataSource" />
</bean>
<bean id="jdbcTemplate" class="org.springframework.jdbc.core.JdbcTemplate">
    <property name="dataSource" ref="shardingDataSource" />
</bean>
<tx:annotation-driven />

<!-- Enable automatic scanning of @ShardingSphereTransactionType annotation and use
Spring's native AOP for class and method enhancements -->
<sharding:tx-type-annotation-driven />
```

Use distributed transactions

```
@Transactional
@ShardingSphereTransactionType(TransactionType.XA)  // support TransactionType.
LOCAL, TransactionType.XA, TransactionType.BASE
public void insert() {
        jdbcTemplate.execute("INSERT INTO t_order (user_id, status) VALUES (?, ?)",
(PreparedStatementCallback<Object>) ps -> {
        ps.setObject(1, i);
        ps.setObject(2, "init");
        ps.executeUpdate();
        });
        }
```

Atomikos Transaction

Background

Apache ShardingSphere provides XA transactions, and the default XA transaction manager is Atomikos.

Procedure

1. Configure the transaction type
2. Configure Atomikos

Sample

Configure the transaction type

Yaml:

```yaml
- !TRANSACTION
  defaultType: XA
  providerType: Atomikos
```

SpringBoot:

```yaml
spring:
  shardingsphere:
    props:
      xa-transaction-manager-type: Atomikos
```

Spring Namespace:

```xml
<shardingsphere:data-source id="xxx" data-source-names="xxx" rule-refs="xxx">
    <props>
        <prop key="xa-transaction-manager-type">Atomikos</prop>
    </props>
</shardingsphere:data-source>
```

Configure Atomikos

Atomikos configuration items can be customized by adding `jta.properties` to the project' s classpath.

See Atomikos' s official documentation for more details.

Data Recovery

`xa_tx.log` is generated in the `logs` directory of the project. This is the log required for recovering XA crash. Do not delete it.

Bitronix Transaction

background

Apache ShardingSphere provides XA transactions that integrate with the Bitronix implementation.

Prerequisites

Introducing Maven dependency

```
<properties>
    <btm.version>2.1.3</btm.version>
</properties>

<dependency>
    <groupId>org.apache.shardingsphere</groupId>
    <artifactId>shardingsphere-jdbc-core</artifactId>
    <version>${shardingsphere.version}</version>
</dependency>

<!-- This module is required when using XA transactions -->
<dependency>
    <groupId>org.apache.shardingsphere</groupId>
    <artifactId>shardingsphere-transaction-xa-core</artifactId>
    <version>${shardingsphere.version}</version>
</dependency>

<dependency>
    <groupId>org.apache.shardingsphere</groupId>
    <artifactId>shardingsphere-transaction-xa-bitronix</artifactId>
    <version>${shardingsphere.version}</version>
</dependency>

<dependency>
    <groupId>org.codehaus.btm</groupId>
    <artifactId>btm</artifactId>
    <version>${btm.version}</version>
</dependency>
```

Procedure

1. Configure the XA transaction type
2. Configure Bitronix

Sample

Configure the XA transaction type

Yaml:

```
- !TRANSACTION
  defaultType: XA
  providerType: Bitronix
```

SpringBoot:

```
spring:
  shardingsphere:
    props:
      xa-transaction-manager-type: Bitronix
```

Spring Namespace:

```
<shardingsphere:data-source id="xxx" data-source-names="xxx" rule-refs="xxx">
    <props>
        <prop key="xa-transaction-manager-type">Bitronix</prop>
    </props>
</shardingsphere:data-source>
```

Configure Bitronix (Deletable)

See Bitronix' s Official Documentation for more details.

Narayana Transaction

Background

Apache ShardingSphere provides XA transactions that integrate with the Narayana implementation.

Prerequisites

Introducing Maven dependency

```
<properties>
    <narayana.version>5.12.4.Final</narayana.version>
    <jboss-transaction-spi.version>7.6.0.Final</jboss-transaction-spi.version>
    <jboss-logging.version>3.2.1.Final</jboss-logging.version>
</properties>

<dependency>
    <groupId>org.apache.shardingsphere</groupId>
    <artifactId>shardingsphere-jdbc-core</artifactId>
    <version>${shardingsphere.version}</version>
</dependency>

<!-- This module is required when using XA transactions -->
<dependency>
    <groupId>org.apache.shardingsphere</groupId>
    <artifactId>shardingsphere-transaction-xa-core</artifactId>
    <version>${shardingsphere.version}</version>
</dependency>

<dependency>
      <groupId>org.apache.shardingsphere</groupId>
      <artifactId>shardingsphere-transaction-xa-narayana</artifactId>
      <version>${shardingsphere.version}</version>
</dependency>
<dependency>
      <groupId>org.jboss.narayana.jta</groupId>
      <artifactId>jta</artifactId>
      <version>${narayana.version}</version>
</dependency>
<dependency>
       <groupId>org.jboss.narayana.jts</groupId>
       <artifactId>narayana-jts-integration</artifactId>
       <version>${narayana.version}</version>
</dependency>
<dependency>
       <groupId>org.jboss</groupId>
       <artifactId>jboss-transaction-spi</artifactId>
       <version>${jboss-transaction-spi.version}</version>
</dependency>
<dependency>
       <groupId>org.jboss.logging</groupId>
       <artifactId>jboss-logging</artifactId>
       <version>${jboss-logging.version}</version>
</dependency>
```

Procedure

1. Configure Narayana
2. Set the XA transaction type

Sample

Configure Narayana

Narayana configuration items can be customized by adding `jbossts-properties.xml` to the project's classpath.

See Narayana's Official Documentation for more details.

Set the XA transaction type

Yaml:

```yaml
- !TRANSACTION
  defaultType: XA
  providerType: Narayana
```

SpringBoot:

```yaml
spring:
  shardingsphere:
    props:
      xa-transaction-manager-type: Narayana
```

Spring Namespace:

```xml
<shardingsphere:data-source id="xxx" data-source-names="xxx" rule-refs="xxx">
    <props>
        <prop key="xa-transaction-manager-type">Narayana</prop>
    </props>
</shardingsphere:data-source>
```

Seata Transaction

Background

Apache ShardingSphere provides BASE transactions that integrate the Seata implementation.

Procedure

1. Start Seata Server
2. Create the log table
3. Add the Seata configuration

Sample

Start Seata Server

Refer to seata-work-shop to download and start the Seata server.

Create undo_log table

Create the undo_log table in each shard database instance (take MySQL as an example).

```
CREATE TABLE IF NOT EXISTS `undo_log`
(
  `id`            BIGINT(20)   NOT NULL AUTO_INCREMENT COMMENT 'increment id',
  `branch_id`     BIGINT(20)   NOT NULL COMMENT 'branch transaction id',
  `xid`           VARCHAR(100) NOT NULL COMMENT 'global transaction id',
  `context`       VARCHAR(128) NOT NULL COMMENT 'undo_log context,such as
serialization',
  `rollback_info` LONGBLOB     NOT NULL COMMENT 'rollback info',
  `log_status`    INT(11)      NOT NULL COMMENT '0:normal status,1:defense status',
  `log_created`   DATETIME     NOT NULL COMMENT 'create datetime',
  `log_modified`  DATETIME     NOT NULL COMMENT 'modify datetime',
  PRIMARY KEY (`id`),
  UNIQUE KEY `ux_undo_log` (`xid`, `branch_id`)
) ENGINE = InnoDB
  AUTO_INCREMENT = 1
  DEFAULT CHARSET = utf8 COMMENT ='AT transaction mode undo table';
```

Modify configuration

Add the seata.conf file to the classpath.

```
client {
    application.id = example    ## Apply the only primary key
    transaction.service.group = my_test_tx_group   ## The transaction group it
belongs to.
}
```

Modify the file.conf and registry.conf files of Seata as required.

6.1.6 Unsupported Items

DataSource Interface

- Do not support timeout related operations

Connection Interface

- Do not support operations of stored procedure, function and cursor
- Do not support native SQL
- Do not support savepoint related operations
- Do not support Schema/Catalog operation
- Do not support self-defined type mapping

Statement and PreparedStatement Interface

- Do not support statements that return multiple result sets (stored procedures, multiple pieces of non-SELECT data)
- Do not support the operation of international characters

ResultSet Interface

- Do not support getting result set pointer position
- Do not support changing result pointer position through none-next method
- Do not support revising the content of result set
- Do not support acquiring international characters
- Do not support getting Array

JDBC 4.1

- Do not support new functions of JDBC 4.1 interface

For all the unsupported methods, please read `org.apache.shardingsphere.driver.jdbc.unsupported` package.

6.2 ShardingSphere-Proxy

Configuration is the only module in ShardingSphere-Proxy that interacts with application developers, through which developer can quickly and clearly understand the functions provided by ShardingSphere-Proxy.

This chapter is a configuration manual for ShardingSphere-Proxy, which can also be referred to as a dictionary if necessary.

ShardingSphere-Proxy provided YAML configuration, and used DistSQL to communicate. By configuration, application developers can flexibly use data sharding, readwrite-splitting, data encryption, shadow database or the combination of them.

Rule configuration keeps consist with YAML configuration of ShardingSphere-JDBC. DistSQL and YAML can be replaced each other.

Please refer to Example for more details.

6.2.1 Startup

This chapter will introduce the deployment and startup of ShardingSphere-Proxy.

Use Binary Tar

Background

This section describes how to start ShardingSphere-Proxy by binary release packages

Premise

Start the Proxy with a binary package requires an environment with Java JRE 8 or later.

Steps

1. Obtain the binary release package of ShardingSphere-Proxy

Obtain the binary release package of ShardingSphere-Proxy on the download page.

2. Configure `conf/server.yaml`

ShardingSphere-Proxy's operational mode is configured on `server.yaml`, and its configuration mode is the same with that of ShardingSphere-JDBC. Refer to mode of configuration.

Please refer to the following links for other configuration items: * Permission configuration * Property configuration

3. Configure `conf/config-*.yaml`

Modify files named with the prefix `config-` in the `conf` directory, such as `conf/config-sharding.yaml` file and configure sharding rules and read/write splitting rules. See Confuguration Mannual for configuration methods. The `*` part of the `config-*.yaml` file can be named whatever you want.

ShardingSphere-Proxy supports multiple logical data sources. Each YAML configuration file named with the prefix `config-` is a logical data source.

4. Introduce database driver (Optional)

If the backend is connected to a PostgreSQL or openGauss database, no additional dependencies need to be introduced.

If the backend is connected to a MySQL database, please download mysql-connector-java-5.1.47.jar or mysql-connector-java-8.0.11.jar, and put it into the `ext-lib` directory.

5. Introduce dependencies required by the cluster mode (Optional)

ShardingSphere-Proxy integrates the ZooKeeper Curator client by default. ZooKeeper is used in cluster mode without introducing other dependencies.

If the cluster mode uses Etcd, the client drivers of Etcd jetcd-core 0.5.0 need to be copied into the `ext-lib` directory.

6. Introduce dependencies required by distributed transactions (Optional)

It is the same with ShardingSphere-JDBC. Please refer to Distributed Transaction for more details.

7. Introduce custom algorithm (Optional)

If you need to use a user-defined algorithm class, you can configure custom algorithm in the following ways:

```
1. Implement the algorithm implementation class defined by `ShardingAlgorithm`.
2. Create a `META-INF/services` directory under the project `resources` directory.
3. Create file `org.apache.shardingsphere.sharding.spi.ShardingAlgorithm` under the
directory `META-INF/services`.
4. Writes the fully qualified class name of the implementation class to a file
`org.apache.shardingsphere.sharding.spi.ShardingAlgorithm`
5. Package the above Java files into jar packages.
6. Copy the above jar package to the `ext-lib` directory.
7. Configure the Java file reference of the above custom algorithm implementation
class in a YAML file, see [Configuration rule](https://shardingsphere.apache.org/
document/current/en/user-manual/shardingsphere-proxy/yaml-config/) for more
details.
```

8. Start ShardingSphere-Proxy

In Linux or macOS, run `bin/start.sh`. In Windows, run `bin/start.bat` to start ShardingSphere-Proxy. The default listening port is `3307` and the default configuration directory is the `conf` directory in Proxy. The startup script can specify the listening port and the configuration file directory by running the following command:

```
bin/start.sh [port] [/path/to/conf]
```

9. Connect ShardingSphere-Proxy with client

Run the MySQL/PostgreSQL/openGauss client command to directly operate ShardingSphere-Proxy.

Connect ShardingSphere-Proxy with MySQL client:

```
mysql -h${proxy_host} -P${proxy_port} -u${proxy_username} -p${proxy_password}
```

Connect ShardingSphere-Proxy with PostgreSQL:

```
psql -h ${proxy_host} -p ${proxy_port} -U ${proxy_username}
```

Connect ShardingSphere-Proxy with openGauss client:

```
gsql -r -h ${proxy_host} -p ${proxy_port} -U ${proxy_username} -W ${proxy_password}
```

Sample

Please refer to samples on ShardingSphere repository for complete configuration: https://github.com/apache/shardingsphere/tree/master/examples/shardingsphere-proxy-example

Use Docker

Background

This chapter is an introduction about how to start ShardingSphere-Proxy via Docker

Notice

Using Docker to start ShardingSphere-Proxy does not require additional package supoort.

Steps

1. Acquire Docker Image

- Method 1 (Recommended): Pull from DockerHub

```
docker pull apache/shardingsphere-proxy
```

- Method 2: Acquire latest master branch image master: https://github.com/apache/shardingsphere/pkgs/container/shardingsphere-proxy

- Method 3: Build your own image

```
git clone https://github.com/apache/shardingsphere
mvn clean install
cd shardingsphere-distribution/shardingsphere-proxy-distribution
mvn clean package -Prelease,docker
```

If the following problems emerge, please make sure Docker daemon Process is running.

```
I/O exception (java.io.IOException) caught when processing request to {}->unix://
localhost:80: Connection refused?
```

2. Configure `conf/server.yaml` and `conf/config-*.yaml`

Configuration file template can be attained from the Docker container and can be copied to any directory on the host:

```
docker run -d --name tmp --entrypoint=bash apache/shardingsphere-proxy
docker cp tmp:/opt/shardingsphere-proxy/conf /host/path/to/conf
docker rm tmp
```

Since the network conditions inside the container may differ from those of the host, if errors such as "cannot connect to the database" occurs, please make sure that the IP of the database specified in the `conf/config-*.yaml` configuration file can be accessed from inside the Docker container.

For details, please refer to ShardingSphere-Proxy quick start manual - binary distribution packages.

3. (Optional) Introduce third-party dependencies or customized algorithms

If you have any of the following requirements: * ShardingSphere-Proxy Backend use MySQL Database; * Implement customized algorithms; * Use Etcd as Registry Center in cluster mode.

Please create `ext-lib` directory anywhere inside the host and refer to the steps in ShardingSphere-Proxy quick start manual - binary distribution packages.

4. Start ShardingSphere-Proxy container

Mount the `conf` and `ext-lib` directories from the host to the container. Start the container:

```
docker run -d \
    -v /host/path/to/conf:/opt/shardingsphere-proxy/conf \
    -v /host/path/to/ext-lib:/opt/shardingsphere-proxy/ext-lib \
    -e PORT=3308 -p13308:3308 apache/shardingsphere-proxy:latest
```

`ext-lib` is not necessary during the process. Users can mount it at will. ShardingSphere-Proxy default portal `3307` can be designated according to environment variable `-e PORT` Customized JVM related parameters can be set according to environment variable `JVM_OPTS`

5. Use Client to connect to ShardingSphere-Proxy

Please refer to ShardingSphere-Proxy quick start manual - binary distribution packages.

Configuration Example

For full configuration, please refer to the examples given in ShardingSphere library: https://github.com/apache/shardingsphere/tree/master/examples/shardingsphere-proxy-example

Use Helm

Background

Use Helm to provide guidance for the installation of ShardingSphere-Proxy instance in a Kubernetes cluster.

Requirements

- Kubernetes 1.18+
- kubectl
- Helm 3.2.0+
- StorageClass of PV (Persistent Volumes) can be dynamically applied for persistent data (Optional)
  .

Procedure

Online installation

1. Add ShardingSphere-Proxy to the local helm repo:

```
helm repo add shardingsphere https://shardingsphere.apache.org/charts
```

1. Install ShardingSphere-Proxy charts:

```
helm install shardingsphere-proxy shardingsphere/shardingsphere-proxy
```

Source installation

1. Charts will be installed with default configuration if the following commands are executed:

```
cd shardingsphere-proxy/charts/governance
helm dependency build
cd ../..
helm dependency build
cd ..
helm install shardingsphere-proxy shardingsphere-proxy
```

1. Please refer to the configuration items description below for more details:
2. Execute `helm list` to acquire all installed releases.

Uninstall

1. Delete all release records by default, add `--keep-history` to keep them.

```
helm uninstall shardingsphere-proxy
```

Parameters

Governance-Node parameters

Name	Description	V al ue
`gover nance.enabled`	Switch to enable or disable the governance helm chart	`` tr ue ``

Governance-Node ZooKeeper parameters

Name	Description	Value
`gove rnance.zookeeper.enabled`	Switch to enable or disable the ZooKeeper helm chart	`true`
`governanc e.zookeeper. replicaCount`	Number of ZooKeeper nodes	`1`
`governance.zooke eper. persistence.enabled`	Enable persistence on ZooKeeper using PVC(s)	` *false*`
`governance.zookeeper. persis- tence.storageClass`	Persistent Volume storage class	`""`
`governance.zookeeper . persistence.accessModes`	Persistent Volume access modes	`["R eadWrite Once"]`
`governance.zo okeeper. persistence.size`	Persistent Volume size	`8Gi`
`governance.zo okeeper. resources.limits`	The resources limits for the ZooKeeper containers	`{}`
`governance.zookeeper.r es- ources.requests.memory`	The requested memory for the ZooKeeper containers	` *256Mi*`
`governance.zookeepe r. resources.requests.cpu`	The requested cpu for the ZooKeeper containers	`250m`

Compute-Node ShardingSphere-Proxy parameters

Name	Description	Value
compute.i mage. repository	Image name of ShardingSphere-Proxy.	a pache/sharding sphere-proxy
compute.i mage. pullPolicy	The policy for pulling ShardingSphere-Proxy image	`` IfNotPresent``
co mpute.image.tag	ShardingSphere-Proxy image tag	5.1.2
compute.i magePullSe-crets	Specify docker-registry secret names as an array	[]
compute.r esources. limits	The resources limits for the ShardingSphere-Proxy containers	{}
c ompute.resources. requests.memory	The requested memory for the ShardingSphere-Proxy containers	2Gi
compute.resourc es. requests.cpu	The requested cpu for the ShardingSphere-Proxy containers	200m
c ompute.replicas	Number of cluster replicas	3
compu te.service.type	ShardingSphere-Proxy network mode	ClusterIP
compu te.service.port	ShardingSphere-Proxy expose port	3307
compute.mysqlCo nnec-tor.version	MySQL connector version	5.1.49
co mpute.startPort	ShardingSphere-Proxy start port	3307
compu te.serverConfig	Server Configuration file for ShardingSphere-Proxy	""

Sample

values.yaml

```
#
#  Licensed to the Apache Software Foundation (ASF) under one or more
#  contributor license agreements.  See the NOTICE file distributed with
#  this work for additional information regarding copyright ownership.
#  The ASF licenses this file to You under the Apache License, Version 2.0
#  (the "License"); you may not use this file except in compliance with
#  the License.  You may obtain a copy of the License at
#
#      http://www.apache.org/licenses/LICENSE-2.0
#
#  Unless required by applicable law or agreed to in writing, software
#  distributed under the License is distributed on an "AS IS" BASIS,
#  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
#  See the License for the specific language governing permissions and
#  limitations under the License.
```

```
#

## @section Governance-Node parameters
## @param governance.enabled Switch to enable or disable the governance helm chart
##
governance:
  enabled: true
  ## @section Governance-Node ZooKeeper parameters
  zookeeper:
    ## @param governance.zookeeper.enabled Switch to enable or disable the
ZooKeeper helm chart
    ##
    enabled: true
    ## @param governance.zookeeper.replicaCount Number of ZooKeeper nodes
    ##
    replicaCount: 1
    ## ZooKeeper Persistence parameters
    ## ref: https://kubernetes.io/docs/user-guide/persistent-volumes/
    ## @param governance.zookeeper.persistence.enabled Enable persistence on
ZooKeeper using PVC(s)
    ## @param governance.zookeeper.persistence.storageClass Persistent Volume
storage class
    ## @param governance.zookeeper.persistence.accessModes Persistent Volume access
modes
    ## @param governance.zookeeper.persistence.size Persistent Volume size
    ##
    persistence:
      enabled: false
      storageClass: ""
      accessModes:
        - ReadWriteOnce
      size: 8Gi
    ## ZooKeeper's resource requests and limits
    ## ref: https://kubernetes.io/docs/user-guide/compute-resources/
    ## @param governance.zookeeper.resources.limits The resources limits for the
ZooKeeper containers
    ## @param governance.zookeeper.resources.requests.memory The requested memory
for the ZooKeeper containers
    ## @param governance.zookeeper.resources.requests.cpu The requested cpu for the
ZooKeeper containers
    ##
    resources:
      limits: {}
      requests:
        memory: 256Mi
        cpu: 250m

## @section Compute-Node parameters
```

```
##
compute:
  ## @section Compute-Node ShardingSphere-Proxy parameters
  ## ref: https://kubernetes.io/docs/concepts/containers/images/
  ## @param compute.image.repository Image name of ShardingSphere-Proxy.
  ## @param compute.image.pullPolicy The policy for pulling ShardingSphere-Proxy
image
  ## @param compute.image.tag ShardingSphere-Proxy image tag
  ##
  image:
    repository: "apache/shardingsphere-proxy"
    pullPolicy: IfNotPresent
    ## Overrides the image tag whose default is the chart appVersion.
    ##
    tag: "5.1.2"
  ## @param compute.imagePullSecrets Specify docker-registry secret names as an
array
  ## e.g:
  ## imagePullSecrets:
  ##   - name: myRegistryKeySecretName
  ##
  imagePullSecrets: []
  ## ShardingSphere-Proxy resource requests and limits
  ## ref: https://kubernetes.io/docs/concepts/configuration/manage-resources-
containers/
  ## @param compute.resources.limits The resources limits for the ShardingSphere-
Proxy containers
  ## @param compute.resources.requests.memory The requested memory for the
ShardingSphere-Proxy containers
  ## @param compute.resources.requests.cpu The requested cpu for the
ShardingSphere-Proxy containers
  ##
  resources:
    limits: {}
    requests:
      memory: 2Gi
      cpu: 200m
  ## ShardingSphere-Proxy Deployment Configuration
  ## ref: https://kubernetes.io/docs/concepts/workloads/controllers/deployment/
  ## ref: https://kubernetes.io/docs/concepts/services-networking/service/
  ## @param compute.replicas Number of cluster replicas
  ##
  replicas: 3
  ## @param compute.service.type ShardingSphere-Proxy network mode
  ## @param compute.service.port ShardingSphere-Proxy expose port
  ##
  service:
    type: ClusterIP
```

```
    port: 3307
  ## MySQL connector Configuration
  ## ref: https://shardingsphere.apache.org/document/current/en/quick-start/
shardingsphere-proxy-quick-start/
  ## @param compute.mysqlConnector.version MySQL connector version
  ##
  mysqlConnector:
    version: "5.1.49"
  ## @param compute.startPort ShardingSphere-Proxy start port
  ## ShardingSphere-Proxy start port
  ## ref: https://shardingsphere.apache.org/document/current/en/user-manual/
shardingsphere-proxy/startup/docker/
  ##
  startPort: 3307
  ## @section Compute-Node ShardingSphere-Proxy ServerConfiguration parameters
  ## NOTE: If you use the sub-charts to deploy Zookeeper, the server-lists field
must be "{{ printf \"%s-zookeeper.%s:2181\" .Release.Name .Release.Namespace }}",
  ## otherwise please fill in the correct zookeeper address
  ## The server.yaml is auto-generated based on this parameter.
  ## If it is empty, the server.yaml is also empty.
  ## ref: https://shardingsphere.apache.org/document/current/en/user-manual/
shardingsphere-jdbc/yaml-config/mode/
  ## ref: https://shardingsphere.apache.org/document/current/en/user-manual/common-
config/builtin-algorithm/metadata-repository/
  ##
  serverConfig:
    ## @section Compute-Node ShardingSphere-Proxy ServerConfiguration authority
parameters
    ## NOTE: It is used to set up initial user to login compute node, and authority
data of storage node.
    ## ref: https://shardingsphere.apache.org/document/current/en/user-manual/
shardingsphere-proxy/yaml-config/authentication/
    ## @param compute.serverConfig.authority.privilege.type authority provider for
storage node, the default value is ALL_PERMITTED
    ## @param compute.serverConfig.authority.users[0].password Password for compute
node.
    ## @param compute.serverConfig.authority.users[0].user Username,authorized host
for compute node. Format: <username>@<hostname> hostname is % or empty string means
do not care about authorized host
    ##
    authority:
      privilege:
        type: ALL_PRIVILEGES_PERMITTED
      users:
      - password: root
        user: root@%
    ## @section Compute-Node ShardingSphere-Proxy ServerConfiguration mode
Configuration parameters
```

```
    ## @param compute.serverConfig.mode.type Type of mode configuration. Now only
support Cluster mode
    ## @param compute.serverConfig.mode.repository.props.namespace Namespace of
registry center
    ## @param compute.serverConfig.mode.repository.props.server-lists Server lists
of registry center
    ## @param compute.serverConfig.mode.repository.props.maxRetries Max retries of
client connection
    ## @param compute.serverConfig.mode.repository.props.
operationTimeoutMilliseconds Milliseconds of operation timeout
    ## @param compute.serverConfig.mode.repository.props.retryIntervalMilliseconds
Milliseconds of retry interval
    ## @param compute.serverConfig.mode.repository.props.timeToLiveSeconds Seconds
of ephemeral data live
    ## @param compute.serverConfig.mode.repository.type Type of persist repository.
Now only support ZooKeeper
    ## @param compute.serverConfig.mode.overwrite Whether overwrite persistent
configuration with local configuration
    ##
    mode:
      type: Cluster
      repository:
        type: ZooKeeper
        props:
          maxRetries: 3
          namespace: governance_ds
          operationTimeoutMilliseconds: 5000
          retryIntervalMilliseconds: 500
          server-lists: "{{ printf \"%s-zookeeper.%s:2181\" .Release.Name .Release.
Namespace }}"
          timeToLiveSeconds: 60
      overwrite: true
```

6.2.2 Yaml Configuration

The YAML configuration of ShardingSphere-JDBC is the subset of ShardingSphere-Proxy. In `server.yaml` file, ShardingSphere-Proxy can configure authority feature and more properties for Proxy only.

This chapter will introduce the extra YAML configuration of ShardingSphere-Proxy.

Authorization

Authorization configuration provided for users who can connect to ShardingSphere-Proxy. Users can be granted different authorities.

Background

ShardingSphere-Proxy uses the global rule, Authority Rule (identified as !AUTHORITY), to configure user and authorization information.

Thanks to ShardingSphere's pluggable architecture, Proxy provides two levels of authority providers, namely:

- `ALL_PERMITTED`: grant all authorities by default without authentication.
- `DATABASE_PERMITTED`: grant users the authority to specify a logical database, mapped through `user-database-mappings`.

The administrator can choose which authority provider to use as needed when configuring the Authority Rule.

Parameter

```
rules:
  - !AUTHORITY
    users:
      - # Specify the username, authorized host, and password for logging in to the compute node. Format: <username>@<hostname>:<password>. When the hostname is % or an empty string, it indicates that the authorized host is not limited.
    provider:
      type: # The authority provider type for storage node. The default value is ALL_PERMITTED.
```

Sample

ALL_PERMITTED

```
rules:
  - !AUTHORITY
    users:
      - root@localhost:root
      - my_user@:pwd
    provider:
      type: ALL_PERMITTED
```

The above configuration indicates: - The user `root` can connect to Proxy only through localhost, and the password is `root`. - The user `my_user` can connect to Proxy through any host, and the password is pwd. - The provider type is ALL_PERMITTED, which indicates that users are granted all authorities by default without authentication.

DATABASE_PERMITTED

```
rules:
  - !AUTHORITY
    users:
      - root@localhost:root
      - my_user@:pwd
    provider:
      type: DATABASE_PERMITTED
      props:
        user-database-mappings: root@localhost=sharding_db, root@localhost=test_db,
my_user@=sharding_db
```

The above configuration indicates: - The provider type is DATABASE_PERMITTED, which indicates that users are granted database-level authority and configuration is needed. - The user root can connect to Proxy only through localhost and can access sharding_db and test_db. - The user my_user can connect to Proxy through any host and can access sharding_db.

Related References

Please refer to Authority Provider for specific implementation of authority provider.

Properties

Background

Apache ShardingSphere can configure system-level configuration through property configuration. This section describes the configuration items in server.yaml.

Parameters

Name	• DataType*	*Description*	• Default*	• DynamicUpdate*
sql-show (?)	boolean	Whether to print SQL in logs. Printing SQL can help developers quickly locate system problems. Logs contain the following contents: logical SQL, authentic SQL and SQL parsing result. If configuration is enabled, logs will use Topic `ShardingSphere-SQL`, and log level is INFO。	false	True
sql-simple (?)	boolean	Whether to print simple SQL in logs.	false	True
kernel-exe cutor-size (?)	int	Set the size of the thread pool for task processing. Each ShardingSphereDataSource uses an independent thread pool, and different data sources on the same JVM do not share thread pools.	infinite	False
max-connec tions-size -per-query (?)	int	The maximum number of connections that a query request can use in each database instance.	1	True

Property configuration can be modified according to DistSQL#RAL. Properties that support dynamic change can take effect immediately. Properties that do not support dynamic change take effect after a restart.

Sample

For a complete sample, please refer to `server.yaml` in ShardingSphere's repository: https://github.com/apache/shardingsphere/blob/aac0d3026e00575114701be603ec189a02a45747/shardingsphere-proxy/shardingsphere-proxy-bootstrap/src/main/resources/conf/server.yaml#L71-L93

Rules

Background

This section describes how to configure the rules for ShardingSphere-Proxy.

Parameters Explained

Rules configuration of ShardingSphere-Proxy is the same as that of ShardingSphere-JDBC. For details, please refer to ShardingSphere-JDBC Rules Configuration.

Notice

Unlike ShardingSphere-JDBC, the following rules need to be configured in `server.yaml` of ShardingSphere-Proxy:

- SQL Parsing
- Distributed Operations
- SQL Translator

6.2.3 DistSQL

This chapter will introduce the detailed syntax of DistSQL.

Definition

DistSQL (Distributed SQL) is Apache ShardingSphere's specific SQL, providing additional operation capabilities compared to standard SQL.

Flexible rule configuration and resource management & control capabilities are one of the characteristics of Apache ShardingSphere.

When using 4.x and earlier versions, developers can operate data just like using a database, but they need to configure resources and rules through YAML file (or registry center). However, the YAML file format and the changes brought by using the registry center made it unfriendly to DBAs.

Starting from version 5.x, DistSQL enables users to operate Apache ShardingSphere just like a database, transforming it from a framework and middleware for developers to a database product for DBAs.

Related Concepts

DistSQL is divided into RDL, RQL, RAL and RUL.

RDL

Resource & Rule Definition Language, is responsible for the definition of resources and rules.

RQL

Resource & Rule Query Language, is responsible for the query of resources and rules.

RAL

Resource & Rule Administration Language, is responsible for hint, circuit breaker, configuration import and export, scaling control and other management functions.

RUL

Resource & Rule Utility Language, is responsible for SQL parsing, SQL formatting, preview execution plan, etc.

Impact on the System

Before

Before having DistSQL, users used SQL to operate data while using YAML configuration files to manage ShardingSphere, as shown below:

At that time, users faced the following problems: - Different types of clients are required to operate data and manage ShardingSphere configuration. - Multiple logical databases require multiple YAML files. - Editing a YAML file requires writing permissions. - Need to restart ShardingSphere after editing YAML.

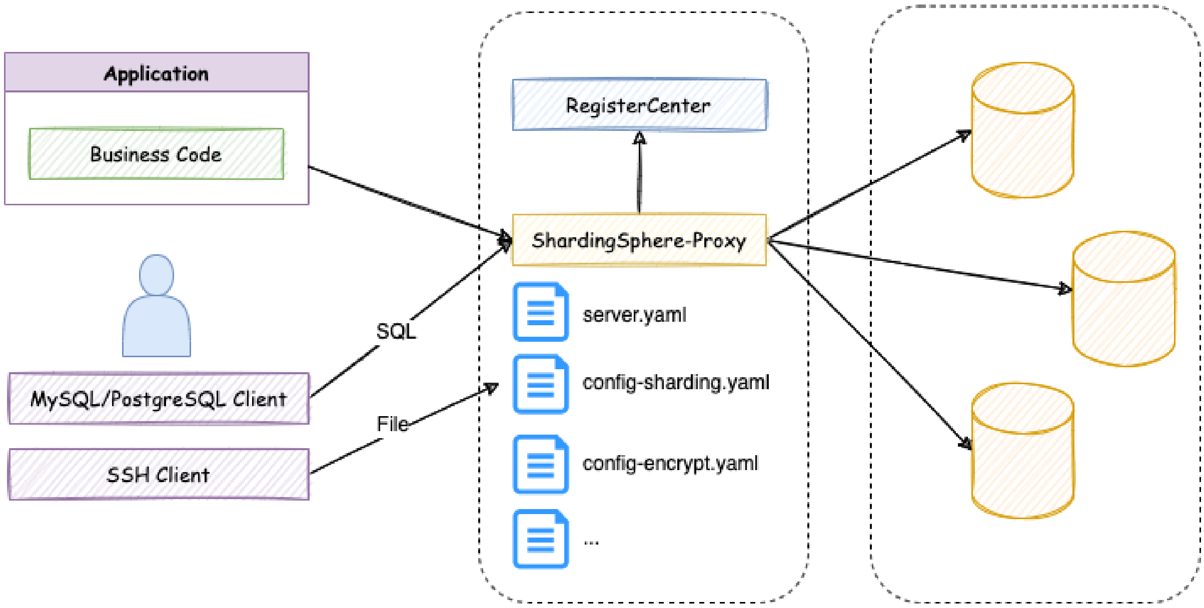


Figure1: Before

After

With the advent of DistSQL, the operation of ShardingSphere has also changed:

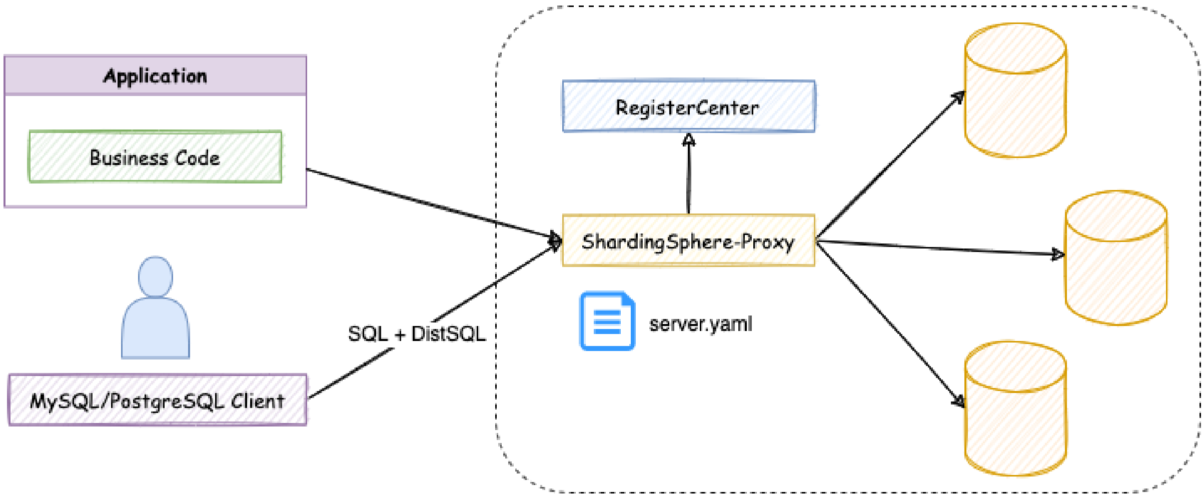


Figure2: After

Now, the user experience has been greatly improved: - Uses the same client to operate data and ShardingSphere configuration. - No need for additional YAML files, and the logical databases are managed through DistSQL. - Editing permissions for files are no longer required, and configuration is managed through DistSQL. - Configuration changes take effect in real-time without restarting ShardingSphere.

Limitations

DistSQL can be used only with ShardingSphere-Proxy, not with ShardingSphere-JDBC for now.

How it works

Like standard SQL, DistSQL is recognized by the parsing engine of ShardingSphere. It converts the input statement into an abstract syntax tree and then generates the `Statement` corresponding to each grammar, which is processed by the appropriate `Handler`.

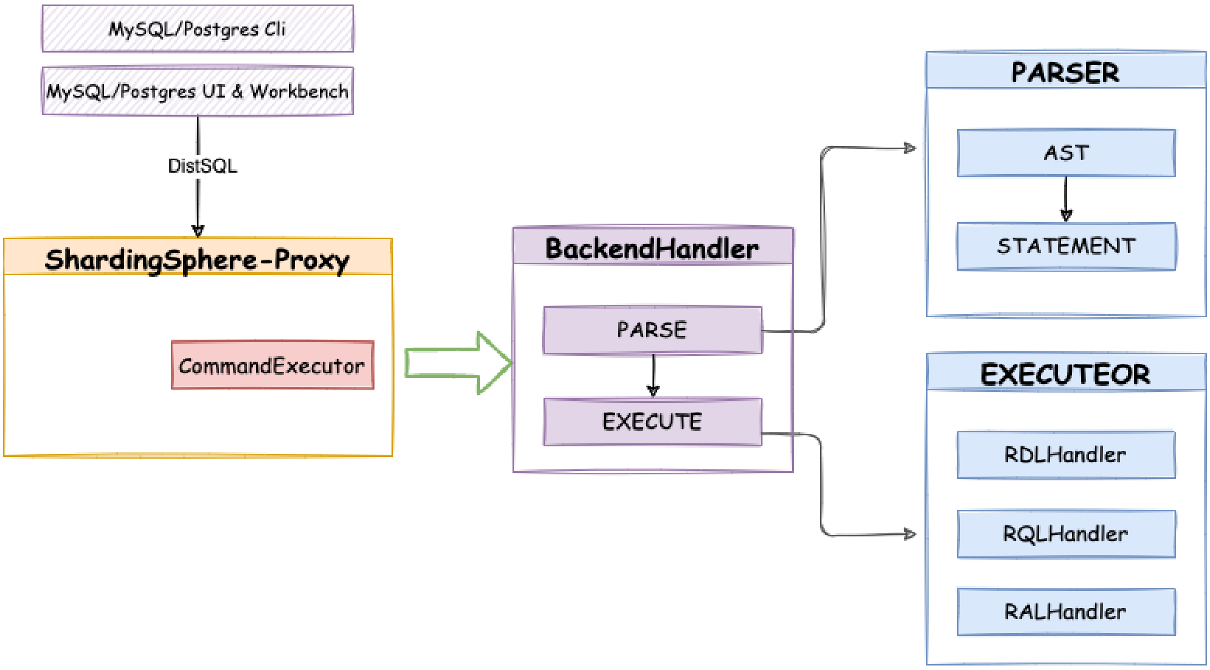


Figure3: Overview

Related References

User Manual: DistSQL

Syntax

This chapter describes the syntax of DistSQL in detail, and introduces use of DistSQL with practical examples.

Syntax Rule

In DistSQL statement, except for keywords, the input format of other elements shall conform to the following rules.

Identifier

1. The identifier represents an object in the SQL statement, including:

- database name
- table name
- column name
- index name
- resource name
- rule name
- algorithm name

2. The allowed characters in the identifier are: `[A-Z, A-Z, 0-9, _]` (letters, numbers, underscores) and should start with a letter.
3. When keywords or special characters appear in the identifier, use the backticks (`).

Literal

Types of literals include:

- string: enclosed in single quotes (') or double quotes (")
- int: it is generally a positive integer, such as 0-9;

Note: some DistSQL syntax allows negative values. In this case, a negative sign (-) can be added before the number, such as -1.

- boolean, containing only true & false. Case insensitive.

RDL Syntax

RDL (Resource & Rule Definition Language) responsible for definition of resources/rules.

Resource Definition

Syntax

```
ADD RESOURCE resourceDefinition [, resourceDefinition] ...

ALTER RESOURCE resourceDefinition [, resourceDefinition] ...

DROP RESOURCE resourceName [, resourceName] ... [ignore single tables]

resourceDefinition:
    simpleSource | urlSource

simpleSource:
    resourceName(HOST=hostname,PORT=port,DB=dbName,USER=user [,PASSWORD=password]
[,PROPERTIES(property [,property]) ...])

urlSource:
    resourceName(URL=url,USER=user [,PASSWORD=password] [,PROPERTIES(property [,
property]) ...])

property:
    key=value
```

Parameters Explained

Name	DataType	Description
resourceName	IDENTIFIER	Resource name
hostname	STRING	Host or IP
port	INT	Port
dbName	STRING	DB name
url	STRING	URL
user	STRING	username
password	STRING	password

Notes

- Before adding resources, please confirm that a distributed database has been created, and execute the `use` command to successfully select a database;
- Confirm that the resource to be added or altered can be connected, otherwise the operation will not be successful;
- Duplicate `resourceName` is not allowed;

- `PROPERTIES` is used to customize connection pool parameters, `key` and `value` are both STRING types;
- `ALTER RESOURCE` is not allowed to change the real data source associated with this resource;
- `ALTER RESOURCE` will switch the connection pool. This operation may affect the ongoing business, please use it with caution;
- `DROP RESOURCE` will only delete logical resources, not real data sources;
- Resources referenced by rules cannot be deleted;
- If the resource is only referenced by `single table rule`, and the user confirms that the restriction can be ignored, the optional parameter `ignore single tables` can be added to perform forced deletion.

Example

```
ADD RESOURCE resource_0 (
    HOST="127.0.0.1",
    PORT=3306,
    DB="db0",
    USER="root",
    PASSWORD="root"
),resource_1 (
    HOST="127.0.0.1",
    PORT=3306,
    DB="db1",
    USER="root"
),resource_2 (
    HOST="127.0.0.1",
    PORT=3306,
    DB="db2",
    USER="root",
    PROPERTIES("maximumPoolSize"="10")
),resource_3 (
    URL="jdbc:mysql://127.0.0.1:3306/db3?serverTimezone=UTC&useSSL=false",
    USER="root",
    PASSWORD="root",
    PROPERTIES("maximumPoolSize"="10","idleTimeout"="30000")
);

ALTER RESOURCE resource_0 (
    HOST="127.0.0.1",
    PORT=3309,
    DB="db0",
    USER="root",
    PASSWORD="root"
),resource_1 (
    URL="jdbc:mysql://127.0.0.1:3309/db1?serverTimezone=UTC&useSSL=false",
```

```
    USER="root",
    PASSWORD="root",
    PROPERTIES("maximumPoolSize"="10","idleTimeout"="30000")
);

DROP RESOURCE resource_0, resource_1;
DROP RESOURCE resource_2, resource_3 ignore single tables;
```

Rule Definition

This chapter describes the syntax of rule definition.

Sharding

Syntax

Sharding Table Rule

```
CREATE SHARDING TABLE RULE shardingTableRuleDefinition [,
shardingTableRuleDefinition] ...

ALTER SHARDING TABLE RULE shardingTableRuleDefinition [,
shardingTableRuleDefinition] ...

DROP SHARDING TABLE RULE tableName [, tableName] ...

CREATE DEFAULT SHARDING shardingScope STRATEGY (shardingStrategy)

ALTER DEFAULT SHARDING shardingScope STRATEGY (shardingStrategy)

DROP DEFAULT SHARDING shardingScope STRATEGY;

CREATE SHARDING ALGORITHM shardingAlgorithmDefinition [,
shardingAlgorithmDefinition] ...

ALTER SHARDING ALGORITHM shardingAlgorithmDefinition [,
shardingAlgorithmDefinition] ...

DROP SHARDING ALGORITHM algorithmName [, algorithmName] ...

CREATE SHARDING KEY GENERATOR keyGeneratorDefinition [, keyGeneratorDefinition] ...

ALTER SHARDING KEY GENERATOR keyGeneratorDefinition [, keyGeneratorDefinition] ...

DROP SHARDING KEY GENERATOR keyGeneratorName [, keyGeneratorName] ...
```

```
shardingTableRuleDefinition:
    shardingAutoTableRule | shardingTableRule

shardingAutoTableRule:
    tableName(resources, shardingColumn, algorithmDefinition [,
keyGenerateDeclaration])

shardingTableRule:
    tableName(dataNodes [, databaseStrategy] [, tableStrategy] [,
keyGenerateDeclaration])

resources:
    RESOURCES(resource [, resource] ...)

dataNodes:
    DATANODES(dataNode [, dataNode] ...)

resource:
    resourceName | inlineExpression

dataNode:
    resourceName | inlineExpression

shardingColumn:
    SHARDING_COLUMN=columnName

algorithmDefinition:
    TYPE(NAME=shardingAlgorithmType [, PROPERTIES([algorithmProperties])])

keyGenerateDeclaration:
    keyGenerateDefinition | keyGenerateConstruction

keyGenerateDefinition:
    KEY_GENERATE_STRATEGY(COLUMN=columnName, strategyDefinition)

shardingScope:
    DATABASE | TABLE

databaseStrategy:
    DATABASE_STRATEGY(shardingStrategy)

tableStrategy:
    TABLE_STRATEGY(shardingStrategy)

keyGenerateConstruction
    KEY_GENERATE_STRATEGY(COLUMN=columnName, KEY_
GENERATOR=keyGenerateAlgorithmName)
```

```
shardingStrategy:
    TYPE=strategyType, shardingColumn, shardingAlgorithm

shardingAlgorithm:
    existingAlgorithm | autoCreativeAlgorithm

existingAlgorithm:
    SHARDING_ALGORITHM=shardingAlgorithmName

autoCreativeAlgorithm:
    SHARDING_ALGORITHM(algorithmDefinition)

strategyDefinition:
    TYPE(NAME=keyGenerateStrategyType [, PROPERTIES([algorithmProperties])])

shardingAlgorithmDefinition:
    shardingAlgorithmName(algorithmDefinition)

algorithmProperties:
    algorithmProperty [, algorithmProperty] ...

algorithmProperty:
    key=value

keyGeneratorDefinition:
    keyGeneratorName (algorithmDefinition)
```

- `RESOURCES` needs to use data source resources managed by RDL
- `shardingAlgorithmType` specifies the type of automatic sharding algorithm, please refer to Auto Sharding Algorithm
- `keyGenerateStrategyType` specifies the distributed primary key generation strategy, please refer to Key Generate Algorithm
- Duplicate `tableName` will not be created
- `shardingAlgorithm` can be reused by different `Sharding Table Rule`, so when executing `DROP SHARDING TABLE RULE`, the corresponding `shardingAlgorithm` will not be removed
- To remove `shardingAlgorithm`, please execute `DROP SHARDING ALGORITHM`
- `strategyType` specifies the sharding strategy, please refer toSharding Strategy
- `Sharding Table Rule` supports both `Auto Table` and `Table` at the same time. The two types are different in syntax. For the corresponding configuration file, please refer to Sharding
- When using the `autoCreativeAlgorithm` way to specify `shardingStrategy`, a new sharding algorithm will be created automatically. The algorithm naming rule is `tableName_strategyType_shardingAlgorithmType`, such as `t_order_database_inline`

Sharding Binding Table Rule

```
CREATE SHARDING BINDING TABLE RULES bindTableRulesDefinition [,
bindTableRulesDefinition] ...

ALTER SHARDING BINDING TABLE RULES bindTableRulesDefinition [,
bindTableRulesDefinition] ...

DROP SHARDING BINDING TABLE RULES bindTableRulesDefinition [,
bindTableRulesDefinition] ...

bindTableRulesDefinition:
    (tableName [, tableName] ... )
```

- `ALTER` will overwrite the binding table configuration in the database with the new configuration

Sharding Broadcast Table Rule

```
CREATE SHARDING BROADCAST TABLE RULES (tableName [, tableName] ... )

ALTER SHARDING BROADCAST TABLE RULES (tableName [, tableName] ... )

DROP SHARDING BROADCAST TABLE RULES
```

- `ALTER` will overwrite the broadcast table configuration in the database with the new configuration

Example

Sharding Table Rule

Key Generator

```
CREATE SHARDING KEY GENERATOR snowflake_key_generator (
TYPE(NAME="SNOWFLAKE")
);

ALTER SHARDING KEY GENERATOR snowflake_key_generator (
TYPE(NAME="SNOWFLAKE"))
);

DROP SHARDING KEY GENERATOR snowflake_key_generator;
```

Auto Table

```
CREATE SHARDING TABLE RULE t_order (
RESOURCES(resource_0,resource_1),
```

```
SHARDING_COLUMN=order_id,TYPE(NAME="hash_mod",PROPERTIES("sharding-count"="4")),
KEY_GENERATE_STRATEGY(COLUMN=another_id,TYPE(NAME="snowflake"))
);

ALTER SHARDING TABLE RULE t_order (
RESOURCES(resource_0,resource_1,resource_2,resource_3),
SHARDING_COLUMN=order_id,TYPE(NAME="hash_mod",PROPERTIES("sharding-count"="16")),
KEY_GENERATE_STRATEGY(COLUMN=another_id,TYPE(NAME="snowflake"))
);

DROP SHARDING TABLE RULE t_order;

DROP SHARDING ALGORITHM t_order_hash_mod;
```

Table

```
CREATE SHARDING ALGORITHM table_inline (
TYPE(NAME="inline",PROPERTIES("algorithm-expression"="t_order_item_${order_id % 2}
"))
);

CREATE SHARDING TABLE RULE t_order_item (
DATANODES("resource_${0..1}.t_order_item_${0..1}"),
DATABASE_STRATEGY(TYPE="standard",SHARDING_COLUMN=user_id,SHARDING_
ALGORITHM(TYPE(NAME="inline",PROPERTIES("algorithm-expression"="resource_${user_id
% 2}")))),
TABLE_STRATEGY(TYPE="standard",SHARDING_COLUMN=order_id,SHARDING_ALGORITHM=table_
inline),
KEY_GENERATE_STRATEGY(COLUMN=another_id,KEY_GENERATOR=snowflake_key_generator)
);

ALTER SHARDING ALGORITHM database_inline (
TYPE(NAME="inline",PROPERTIES("algorithm-expression"="resource_${user_id % 4}"))
),table_inline (
TYPE(NAME="inline",PROPERTIES("algorithm-expression"="t_order_item_${order_id % 4}
"))
);

ALTER SHARDING TABLE RULE t_order_item (
DATANODES("resource_${0..3}.t_order_item${0..3}"),
DATABASE_STRATEGY(TYPE="standard",SHARDING_COLUMN=user_id,SHARDING_
ALGORITHM=database_inline),
TABLE_STRATEGY(TYPE="standard",SHARDING_COLUMN=order_id,SHARDING_ALGORITHM=table_
inline),
KEY_GENERATE_STRATEGY(COLUMN=another_id,KEY_GENERATOR=snowflake_key_generator)
);

DROP SHARDING TABLE RULE t_order_item;
```

```sql
DROP SHARDING ALGORITHM database_inline;

CREATE DEFAULT SHARDING DATABASE STRATEGY (
TYPE="standard",SHARDING_COLUMN=order_id,SHARDING_ALGORITHM=database_inline
);

ALTER DEFAULT SHARDING DATABASE STRATEGY (
TYPE="standard",SHARDING_COLUMN=another_id,SHARDING_ALGORITHM=database_inline
);

DROP DEFAULT SHARDING DATABASE STRATEGY;
```

Sharding Binding Table Rule

```sql
CREATE SHARDING BINDING TABLE RULES (t_order,t_order_item),(t_1,t_2);

ALTER SHARDING BINDING TABLE RULES (t_order,t_order_item);

DROP SHARDING BINDING TABLE RULES;

DROP SHARDING BINDING TABLE RULES (t_order,t_order_item);
```

Sharding Broadcast Table Rule

```sql
CREATE SHARDING BROADCAST TABLE RULES (t_b,t_a);

ALTER SHARDING BROADCAST TABLE RULES (t_b,t_a,t_3);

DROP SHARDING BROADCAST TABLE RULES;
```

Single Table

Definition

```sql
CREATE DEFAULT SINGLE TABLE RULE singleTableRuleDefinition

ALTER DEFAULT SINGLE TABLE RULE singleTableRuleDefinition

DROP DEFAULT SINGLE TABLE RULE

singleTableRuleDefinition:
    RESOURCE = resourceName
```

- RESOURCE needs to use data source resource managed by RDL

Example

Single Table Rule

```
CREATE DEFAULT SINGLE TABLE RULE RESOURCE = ds_0

ALTER DEFAULT SINGLE TABLE RULE RESOURCE = ds_1

DROP DEFAULT SINGLE TABLE RULE
```

Readwrite-Splitting

Syntax

```
CREATE READWRITE_SPLITTING RULE readwriteSplittingRuleDefinition [,
readwriteSplittingRuleDefinition] ...

ALTER READWRITE_SPLITTING RULE readwriteSplittingRuleDefinition [,
readwriteSplittingRuleDefinition] ...

DROP READWRITE_SPLITTING RULE ruleName [, ruleName] ...

readwriteSplittingRuleDefinition:
    ruleName ([staticReadwriteSplittingRuleDefinition |
dynamicReadwriteSplittingRuleDefinition]
              [, loadBanlancerDefinition])

staticReadwriteSplittingRuleDefinition:
    WRITE_RESOURCE=writeResourceName, READ_RESOURCES(resourceName [, resourceName]
... )

dynamicReadwriteSplittingRuleDefinition:
    AUTO_AWARE_RESOURCE=resourceName [, WRITE_DATA_SOURCE_QUERY_
ENABLED=writeDataSourceQueryEnabled]

loadBanlancerDefinition:
    TYPE(NAME=loadBanlancerType [, PROPERTIES([algorithmProperties] )] )

algorithmProperties:
    algorithmProperty [, algorithmProperty] ...

algorithmProperty:
    key=value
```

```
writeDataSourceQueryEnabled:
    TRUE | FALSE
```

Parameters Explained

name	Da te Ty pe	Description
ruleName	ID EN TI FI ER	Rule name
writeRe source-Name	ID EN TI FI ER	Write data source name
readRe sourceName	ID EN TI FI ER	Read data source name
a utoAwareRe sourceName	ID EN TI FI ER	Database discovery logic data source name
writeDa taSourceQu eryEnabled	B OO LE AN	All read data source are offline, write data source whether the data source is responsible for read traffic
loadBa lancerType	ST RI NG	Load balancing algorithm type

Notes

- Support the creation of static readwrite-splitting rules and dynamic readwrite-splitting rules
- Dynamic readwrite-splitting rules rely on database discovery rules
- `loadBanlancerType` specifies the load balancing algorithm type, please refer to Load Balance Algorithm
- Duplicate `ruleName` will not be created

Example

```
// Static
CREATE READWRITE_SPLITTING RULE ms_group_0 (
WRITE_RESOURCE=write_ds,
READ_RESOURCES(read_ds_0,read_ds_1),
TYPE(NAME="random")
);

// Dynamic
CREATE READWRITE_SPLITTING RULE ms_group_1 (
AUTO_AWARE_RESOURCE=group_0,
WRITE_DATA_SOURCE_QUERY_ENABLED=false,
TYPE(NAME="random",PROPERTIES("read_weight"="2:1"))
```

```
);

ALTER READWRITE_SPLITTING RULE ms_group_1 (
WRITE_RESOURCE=write_ds,
READ_RESOURCES(read_ds_0,read_ds_1,read_ds_2),
TYPE(NAME="random",PROPERTIES("read_weight"="2:0"))
);

DROP READWRITE_SPLITTING RULE ms_group_1;
```

DB Discovery

Syntax

```
CREATE DB_DISCOVERY RULE ruleDefinition [, ruleDefinition] ...

ALTER DB_DISCOVERY RULE ruleDefinition [, ruleDefinition] ...

DROP DB_DISCOVERY RULE ruleName [, ruleName] ...

CREATE DB_DISCOVERY TYPE databaseDiscoveryTypeDefinition [,
databaseDiscoveryTypeDefinition] ...

ALTER DB_DISCOVERY TYPE databaseDiscoveryTypeDefinition [,
databaseDiscoveryTypeDefinition] ...

DROP DB_DISCOVERY TYPE discoveryTypeName [, discoveryTypeName] ...

CREATE DB_DISCOVERY HEARTBEAT databaseDiscoveryHeartbaetDefinition [,
databaseDiscoveryHeartbaetDefinition] ...

ALTER DB_DISCOVERY HEARTBEAT databaseDiscoveryHeartbaetDefinition [,
databaseDiscoveryHeartbaetDefinition] ...

DROP DB_DISCOVERY HEARTBEAT discoveryHeartbeatName [, discoveryHeartbeatName] ...

ruleDefinition:
    (databaseDiscoveryRuleDefinition | databaseDiscoveryRuleConstruction)

databaseDiscoveryRuleDefinition
    ruleName (resources, typeDefinition, heartbeatDefinition)

databaseDiscoveryRuleConstruction
    ruleName (resources, TYPE = discoveryTypeName, HEARTBEAT =
discoveryHeartbeatName)
```

```
databaseDiscoveryTypeDefinition
    discoveryTypeName (typeDefinition)

databaseDiscoveryHeartbaetDefinition
    discoveryHeartbeatName (PROPERTIES (properties))

resources:
    RESOURCES(resourceName [, resourceName] ...)

typeDefinition:
    TYPE(NAME=typeName [, PROPERTIES([properties] )] )

heartbeatDefinition
    HEARTBEAT (PROPERTIES (properties))

properties:
    property [, property] ...

property:
    key=value
```

Parameters Explained

name	DateType	Description
discoveryTypeName	IDENTIFIER	Database discovery type name
ruleName	IDENTIFIER	Rule name
discoveryHeartbeatName	IDENTIFIER	Detect heartbeat name
typeName	STRING	Database discovery type, such as: MySQL.MGR
resourceName	IDENTIFIER	Resource name

Notes

- `discoveryType` specifies the database discovery service type, `ShardingSphere` has built-in support for `MySQL.MGR`
- Duplicate `ruleName` will not be created
- The `discoveryType` and `discoveryHeartbeat` being used cannot be deleted
- Names with `-` need to use `"  "` when changing
- When removing the `discoveryRule`, the `discoveryType` and `discoveryHeartbeat` used by the `discoveryRule` will not be removed

Example

When creating a `discoveryRule`, create both `discoveryType` and `discoveryHeartbeat`

```
CREATE DB_DISCOVERY RULE db_discovery_group_0 (
RESOURCES(ds_0, ds_1, ds_2),
TYPE(NAME='MySQL.MGR',PROPERTIES('group-name'='92504d5b-6dec')),
HEARTBEAT(PROPERTIES('keep-alive-cron'='0/5 * * * * ?'))
);

ALTER DB_DISCOVERY RULE db_discovery_group_0 (
RESOURCES(ds_0, ds_1, ds_2),
TYPE(NAME='MySQL.MGR',PROPERTIES('group-name'='246e9612-aaf1')),
HEARTBEAT(PROPERTIES('keep-alive-cron'='0/5 * * * * ?'))
);

DROP DB_DISCOVERY RULE db_discovery_group_0;

DROP DB_DISCOVERY TYPE db_discovery_group_0_mgr;

DROP DB_DISCOVERY HEARTBEAT db_discovery_group_0_heartbeat;
```

Use the existing `discoveryType` and `discoveryHeartbeat` to create a `discoveryRule`

```
CREATE DB_DISCOVERY TYPE db_discovery_group_1_mgr(
  TYPE(NAME='MySQL.MGR',PROPERTIES('group-name'='92504d5b-6dec'))
);

CREATE DB_DISCOVERY HEARTBEAT db_discovery_group_1_heartbeat(
  PROPERTIES('keep-alive-cron'='0/5 * * * * ?')
);

CREATE DB_DISCOVERY RULE db_discovery_group_1 (
RESOURCES(ds_0, ds_1, ds_2),
TYPE=db_discovery_group_1_mgr,
HEARTBEAT=db_discovery_group_1_heartbeat
);

ALTER DB_DISCOVERY TYPE db_discovery_group_1_mgr(
  TYPE(NAME='MySQL.MGR',PROPERTIES('group-name'='246e9612-aaf1'))
);

ALTER DB_DISCOVERY HEARTBEAT db_discovery_group_1_heartbeat(
  PROPERTIES('keep-alive-cron'='0/10 * * * * ?')
);
```

```sql
ALTER DB_DISCOVERY RULE db_discovery_group_1 (
RESOURCES(ds_0, ds_1),
TYPE=db_discovery_group_1_mgr,
HEARTBEAT=db_discovery_group_1_heartbeat
);

DROP DB_DISCOVERY RULE db_discovery_group_1;

DROP DB_DISCOVERY TYPE db_discovery_group_1_mgr;

DROP DB_DISCOVERY HEARTBEAT db_discovery_group_1_heartbeat;
```

Encrypt

Syntax

```sql
CREATE ENCRYPT RULE encryptRuleDefinition [, encryptRuleDefinition] ...

ALTER ENCRYPT RULE encryptRuleDefinition [, encryptRuleDefinition] ...

DROP ENCRYPT RULE tableName [, tableName] ...

encryptRuleDefinition:
    tableName(COLUMNS(columnDefinition [, columnDefinition] ...), QUERY_WITH_
CIPHER_COLUMN=queryWithCipherColumn)

columnDefinition:
    (NAME=columnName [, PLAIN=plainColumnName] , CIPHER=cipherColumnName,
encryptAlgorithm)

encryptAlgorithm:
    TYPE(NAME=encryptAlgorithmType [, PROPERTIES([algorithmProperties] )] )

algorithmProperties:
    algorithmProperty [, algorithmProperty] ...

algorithmProperty:
    key=value
```

Parameters Explained

name	DateType	Description
tableName	IDENTIFIER	Table name
columnName	IDENTIFIER	Logic column name
plainColumnName	IDENTIFIER	Plain column name
cipherColumnName	IDENTIFIER	Cipher column name
encryptAlgorithmType	STRING	Encryption algorithm type name

Notes

- `PLAIN` specifies the plain column, `CIPHER` specifies the cipher column
- `encryptAlgorithmType` specifies the encryption algorithm type, please refer to Encryption Algorithm
- Duplicate `tableName` will not be created
- `queryWithCipherColumn` support uppercase or lowercase true or false

Example

```
CREATE ENCRYPT RULE t_encrypt (
COLUMNS(
(NAME=user_id,PLAIN=user_plain,CIPHER=user_cipher,TYPE(NAME='AES',PROPERTIES('aes-key-value'='123456abc'))),
(NAME=order_id, CIPHER =order_cipher,TYPE(NAME='MD5'))
), QUERY_WITH_CIPHER_COLUMN=true),
t_encrypt_2 (
COLUMNS(
(NAME=user_id,PLAIN=user_plain,CIPHER=user_cipher,TYPE(NAME='AES',PROPERTIES('aes-key-value'='123456abc'))),
(NAME=order_id, CIPHER=order_cipher,TYPE(NAME='MD5'))
), QUERY_WITH_CIPHER_COLUMN=FALSE);

ALTER ENCRYPT RULE t_encrypt (
COLUMNS(
(NAME=user_id,PLAIN=user_plain,CIPHER=user_cipher,TYPE(NAME='AES',PROPERTIES('aes-key-value'='123456abc'))),
(NAME=order_id,CIPHER=order_cipher,TYPE(NAME='MD5'))
), QUERY_WITH_CIPHER_COLUMN=TRUE);

DROP ENCRYPT RULE t_encrypt,t_encrypt_2;
```

Shadow

Syntax

```
CREATE SHADOW RULE shadowRuleDefinition [, shadowRuleDefinition] ...

ALTER SHADOW RULE shadowRuleDefinition [, shadowRuleDefinition] ...

CREATE SHADOW ALGORITHM shadowAlgorithm [, shadowAlgorithm] ...

ALTER SHADOW ALGORITHM shadowAlgorithm [, shadowAlgorithm] ...

DROP SHADOW RULE ruleName [, ruleName] ...

DROP SHADOW ALGORITHM algorithmName [, algorithmName] ...

CREATE DEFAULT SHADOW ALGORITHM NAME = algorithmName

shadowRuleDefinition: ruleName(resourceMapping, shadowTableRule [, shadowTableRule]
...)

resourceMapping: SOURCE=resourceName, SHADOW=resourceName

shadowTableRule: tableName(shadowAlgorithm [, shadowAlgorithm] ...)

shadowAlgorithm: ([algorithmName, ] TYPE(NAME=shadowAlgorithmType,
PROPERTIES([algorithmProperties] ...)))

algorithmProperties: algorithmProperty [, algorithmProperty] ...

algorithmProperty: key=value
```

Parameters Explained

name	DateType	Description
ruleName	IDENTIFIER	Rule name
resourceName	IDENTIFIER	Resource name
tableName	IDENTIFIER	Shadow table name
algorithmName	IDENTIFIER	Shadow algorithm name
shadowAlgorithmType	STRING	Shadow algorithm type

Notes

- Duplicate ruleName cannot be created
- resourceMapping specifies the mapping relationship between the source database and the shadow library. You need to use the resource managed by RDL, please refer to resource
- shadowAlgorithm can act on multiple shadowTableRule at the same time
- If algorithmName is not specified, it will be automatically generated according to ruleName, tableName and shadowAlgorithmType
- shadowAlgorithmType currently supports VALUE_MATCH, REGEX_MATCH and SIMPLE_HINT
- shadowTableRule can be reused by different shadowRuleDefinition, so when executing DROP SHADOW RULE, the corresponding shadowTableRule will not be removed
- shadowAlgorithm can be reused by different shadowTableRule, so when executing ALTER SHADOW RULE, the corresponding shadowAlgorithm will not be removed

Example

```
CREATE SHADOW RULE shadow_rule(
SOURCE=demo_ds,
SHADOW=demo_ds_shadow,
t_order((simple_hint_algorithm, TYPE(NAME="SIMPLE_HINT", PROPERTIES("shadow"="true
", "foo"="bar"))),(TYPE(NAME="REGEX_MATCH", PROPERTIES("operation"="insert","column
"="user_id", "regex"='[1]')))),
t_order_item((TYPE(NAME="VALUE_MATCH", PROPERTIES("operation"="insert","column"=
"user_id", "value"='1')))));

ALTER SHADOW RULE shadow_rule(
SOURCE=demo_ds,
SHADOW=demo_ds_shadow,
t_order((simple_hint_algorithm, TYPE(NAME="SIMPLE_HINT", PROPERTIES("shadow"="true
", "foo"="bar"))),(TYPE(NAME="REGEX_MATCH", PROPERTIES("operation"="insert","column
"="user_id", "regex"='[1]')))),
t_order_item((TYPE(NAME="VALUE_MATCH", PROPERTIES("operation"="insert","column"=
"user_id", "value"='1')))));

CREATE SHADOW ALGORITHM
(simple_hint_algorithm, TYPE(NAME="SIMPLE_HINT", PROPERTIES("shadow"="true", "foo"=
"bar"))),
(user_id_match_algorithm, TYPE(NAME="REGEX_MATCH",PROPERTIES("operation"="insert",
"column"="user_id", "regex"='[1]')));


ALTER SHADOW ALGORITHM
(simple_hint_algorithm, TYPE(NAME="SIMPLE_HINT", PROPERTIES("shadow"="false", "foo
"="bar"))),
```

```
(user_id_match_algorithm, TYPE(NAME="VALUE_MATCH",PROPERTIES("operation"="insert",
"column"="user_id", "value"='1')));

DROP SHADOW RULE shadow_rule;

DROP SHADOW ALGORITHM simple_hint_algorithm;

CREATE DEFAULT SHADOW ALGORITHM NAME = simple_hint_algorithm;
```

RQL Syntax

RQL (Resource & Rule Query Language) responsible for resources/rules query.

Resource Query

Syntax

```
SHOW DATABASE RESOURCES [FROM databaseName]
```

Return Value Description

Column	Description
name	Data source name
type	Data source type
host	Data source host
port	Data source port
db	Database name
attribute	Data source attribute

Example

```
mysql> SHOW DATABASE RESOURCES;
+------+-------+-----------+------+------+----------------------------------+-------
--------------------+----------------------------+----------------+----------------+--
---------+-----------------------------------------------------------------------
-----------------------------------------------------------------------------
-----------------------------------------------------------------------------
-----------------------------------------------------------------------------
-----------------------------------------------------------------------------
-----------------------------------------------------------------------------
-----------------------------------------------------------------------------
```

```
-----------------------------------------------------------------------------------+
| name | type  | host      | port | db   | connection_timeout_milliseconds | idle_
timeout_milliseconds | max_lifetime_milliseconds | max_pool_size | min_pool_size |
read_only | other_attributes







                                                                             |
+------+-------+-----------+------+------+---------------------------------+-------
--------------------+---------------------------+---------------+---------------+--
---------+-------------------------------------------------------------------------
-----------------------------------------------------------------------------------
-----------------------------------------------------------------------------------
-----------------------------------------------------------------------------------
-----------------------------------------------------------------------------------
-----------------------------------------------------------------------------------
-----------------------------------------------------------------------------------
-----------------------------------------------------------------------------------+
| ds_0 | MySQL | 127.0.0.1 | 3306 | db_0 | 30000                           | 60000
                   | 1800000                   | 50              | 1             |
false     | {"dataSourceProperties":{"cacheServerConfiguration":"true",
"elideSetAutoCommits":"true","useServerPrepStmts":"true","cachePrepStmts":"true",
"rewriteBatchedStatements":"true","cacheResultSetMetadata":"false",
"useLocalSessionState":"true","maintainTimeStats":"false","prepStmtCacheSize":"8192
","tinyInt1isBit":"false","prepStmtCacheSqlLimit":"2048",
"netTimeoutForStreamingResults":"0","zeroDateTimeBehavior":"round"},
"healthCheckProperties":{},"initializationFailTimeout":1,"validationTimeout":5000,
"leakDetectionThreshold":0,"poolName":"HikariPool-1","registerMbeans":false,
"allowPoolSuspension":false,"autoCommit":true,"isolateInternalQueries":false} |
| ds_1 | MySQL | 127.0.0.1 | 3306 | db_1 | 30000                           | 60000
                   | 1800000                   | 50              | 1             |
false     | {"dataSourceProperties":{"cacheServerConfiguration":"true",
"elideSetAutoCommits":"true","useServerPrepStmts":"true","cachePrepStmts":"true",
"rewriteBatchedStatements":"true","cacheResultSetMetadata":"false",
"useLocalSessionState":"true","maintainTimeStats":"false","prepStmtCacheSize":"8192
","tinyInt1isBit":"false","prepStmtCacheSqlLimit":"2048",
"netTimeoutForStreamingResults":"0","zeroDateTimeBehavior":"round"},
"healthCheckProperties":{},"initializationFailTimeout":1,"validationTimeout":5000,
"leakDetectionThreshold":0,"poolName":"HikariPool-2","registerMbeans":false,
"allowPoolSuspension":false,"autoCommit":true,"isolateInternalQueries":false} |
+------+-------+-----------+------+------+---------------------------------+-------
--------------------+---------------------------+---------------+---------------+--
---------+-------------------------------------------------------------------------
-----------------------------------------------------------------------------------
-----------------------------------------------------------------------------------
```

```
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
-------------------------------------------------------------------------------+
2 rows in set (0.84 sec)
```

Rule Query

This chapter describes the syntax of rule query.

Sharding

Syntax

Sharding Table Rule

```
SHOW SHARDING TABLE tableRule | RULES [FROM databaseName]

SHOW SHARDING ALGORITHMS [FROM databaseName]

SHOW UNUSED SHARDING ALGORITHMS [FROM databaseName]

SHOW SHARDING AUDITORS [FROM databaseName]

SHOW SHARDING TABLE RULES USED ALGORITHM shardingAlgorithmName [FROM databaseName]

SHOW SHARDING KEY GENERATORS [FROM databaseName]

SHOW UNUSED SHARDING KEY GENERATORS [FROM databaseName]

SHOW SHARDING TABLE RULES USED KEY GENERATOR keyGeneratorName [FROM databaseName]

SHOW DEFAULT SHARDING STRATEGY

SHOW SHARDING TABLE NODES

tableRule:
    RULE tableName
```

- Support query all data fragmentation rules and specified table query
- Support query all sharding algorithms
- Support query all sharding audit algorithms

Sharding Binding Table Rule

```
SHOW SHARDING BINDING TABLE RULES [FROM databaseName]
```

Sharding Broadcast Table Rule

```
SHOW SHARDING BROADCAST TABLE RULES [FROM databaseName]
```

Sharding Table Rule

Column	Description
table	Logical table name
actual_data_nodes	Actual data node
actual_data_sources	Actual data source (Displayed when creating rules by RDL)
database_strategy_type	Database sharding strategy type
d atabase_sharding_column	Database sharding column
database_ sharding_algorithm_type	Database sharding algorithm type
database_s harding_algorithm_props	Database sharding algorithm properties
table_strategy_type	Table sharding strategy type
table_sharding_column	Table sharding column
table_ sharding_algorithm_type	Table sharding algorithm type
table_s harding_algorithm_props	Table sharding algorithm properties
key_generate_column	Sharding key generator column
key_generator_type	Sharding key generator type
key_generator_props	Sharding key generator properties

Sharding Algorithms

Column	Description
name	Sharding algorithm name
type	Sharding algorithm type
props	Sharding algorithm properties

Unused Sharding Algorithms

Column	Description
name	Sharding algorithm name
type	Sharding algorithm type
props	Sharding algorithm properties

Sharding auditors

Column	Description
name	Sharding audit algorithm name
type	Sharding audit algorithm type
props	Sharding audit algorithm properties

Sharding key generators

Column	Description
name	Sharding key generator name
type	Sharding key generator type
props	Sharding key generator properties

Unused Sharding Key Generators

Column	Description
name	Sharding key generator name
type	Sharding key generator type
props	Sharding key generator properties

Default Sharding Strategy

Column	Description
name	Strategy name
type	Sharding strategy type
sharding_column	Sharding column
sharding_algorithm_name	Sharding algorithm name
sharding_algorithm_type	Sharding algorithm type
sharding_algorithm_props	Sharding algorithm properties

Sharding Table Nodes

Column	Description
name	Sharding rule name
nodes	Sharding nodes

Sharding Binding Table Rule

Column	Description
sharding_binding_tables	sharding Binding Table list

Sharding Broadcast Table Rule

Column	Description
sharding_broadcast_tables	sharding Broadcast Table list

Sharding Table Rule

SHOW SHARDING TABLE RULES

```
mysql> SHOW SHARDING TABLE RULES;
+--------------+---------------------------------+-------------------+-------------
---------+-----------------------+-------------------------------+----------------
------------------------+-------------------+---------------------+----------------
------------+--------------------------------------------------+------------------
-+------------------+-------------------+
| table        | actual_data_nodes               | actual_data_sources | database_
strategy_type | database_sharding_column | database_sharding_algorithm_type |
database_sharding_algorithm_props         | table_strategy_type | table_sharding_
column | table_sharding_algorithm_type | table_sharding_algorithm_props
          | key_generate_column | key_generator_type | key_generator_props |
+--------------+---------------------------------+-------------------+-------------
---------+-----------------------+-------------------------------+----------------
------------------------+-------------------+---------------------+----------------
------------+--------------------------------------------------+------------------
-+------------------+-------------------+
| t_order      | ds_${0..1}.t_order_${0..1}      |                   | INLINE
        | user_id                | INLINE                         | algorithm-
expression:ds_${user_id % 2} | INLINE            | order_id             | INLINE
               | algorithm-expression:t_order_${order_id % 2}      | order_id
    | SNOWFLAKE         |                   |
| t_order_item | ds_${0..1}.t_order_item_${0..1} |                   | INLINE
```

```
        | user_id                  | INLINE                          | algorithm-
expression:ds_${user_id % 2} | INLINE            | order_id            | INLINE
                | algorithm-expression:t_order_item_${order_id % 2} | order_item_id
    | SNOWFLAKE        |                   |
| t2           |                                  | ds_0,ds_1         |
        |                        |                                |
                      | mod               | id                   | mod
         | sharding-count:10                                    |                   |
                |                   |
+--------------+---------------------------------+-------------------+-------------
---------+------------------------+-------------------------------+----------------
------------------------+-------------------+----------------------+-------------
------------+-----------------------------------------------------+------------------
-+-------------------+-------------------+
3 rows in set (0.02 sec)
```

SHOW SHARDING TABLE RULE tableName

```
mysql> SHOW SHARDING TABLE RULE t_order;
+----------+----------------------------+-------------------+----------------------
+-------------------------+-------------------------------+-------------------------
---------------+--------------------+---------------------+-------------------------
---+---------------------------------------------+-------------------+------------
------+-------------------+
| table    | actual_data_nodes          | actual_data_sources | database_strategy_
type | database_sharding_column | database_sharding_algorithm_type | database_
sharding_algorithm_props         | table_strategy_type | table_sharding_column |
table_sharding_algorithm_type | table_sharding_algorithm_props                  |
key_generate_column | key_generator_type | key_generator_props |
+----------+----------------------------+-------------------+----------------------
+-------------------------+-------------------------------+-------------------------
---------------+--------------------+---------------------+-------------------------
---+---------------------------------------------+-------------------+------------
------+-------------------+
| t_order | ds_${0..1}.t_order_${0..1} |                    | INLINE               |
user_id                | INLINE                        | algorithm-expression:ds_$
{user_id % 2} | INLINE            | order_id            | INLINE
 | algorithm-expression:t_order_${order_id % 2} | order_id          | SNOWFLAKE
   |                   |
+----------+----------------------------+-------------------+----------------------
+-------------------------+-------------------------------+-------------------------
---------------+--------------------+---------------------+-------------------------
---+---------------------------------------------+-------------------+------------
------+-------------------+
1 row in set (0.01 sec)
```

SHOW SHARDING ALGORITHMS

```
mysql> SHOW SHARDING ALGORITHMS;
+-------------------------+--------+-----------------------------------------------
------+
| name                    | type   | props
     |
+-------------------------+--------+-----------------------------------------------
------+
| t_order_inline          | INLINE | algorithm-expression=t_order_${order_id % 2}
     |
| t_order_item_inline     | INLINE | algorithm-expression=t_order_item_${order_id %
2}   |
+-------------------------+--------+-----------------------------------------------
------+
2 row in set (0.01 sec)
```

SHOW UNUSED SHARDING ALGORITHMS

```
mysql> SHOW UNUSED SHARDING ALGORITHMS;
+---------------+--------+-----------------------------------------------------+
| name          | type   | props                                               |
+---------------+--------+-----------------------------------------------------+
| t1_inline     | INLINE | algorithm-expression=t_order_${order_id % 2}        |
+---------------+--------+-----------------------------------------------------+
1 row in set (0.01 sec)
```

SHOW SHARDING AUDITORS

```
mysql> SHOW SHARDING AUDITORS;
+------------+-------------------------+-------+
| name       | type                    | props |
+------------+-------------------------+-------+
| dml_audit  | DML_SHARDING_CONDITIONS |       |
+------------+-------------------------+-------+
2 row in set (0.01 sec)
```

SHOW SHARDING TABLE RULES USED ALGORITHM shardingAlgorithmName

```
mysql> SHOW SHARDING TABLE RULES USED ALGORITHM t_order_inline;
+-------+---------+
| type  | name    |
+-------+---------+
| table | t_order |
+-------+---------+
1 row in set (0.01 sec)
```

SHOW SHARDING KEY GENERATORS

```
mysql> SHOW SHARDING KEY GENERATORS;
+-------------------------+-----------+-----------------+
```

```
| name                     | type      | props            |
+--------------------------+-----------+------------------+
| t_order_snowflake        | snowflake |                  |
| t_order_item_snowflake   | snowflake |                  |
| uuid_key_generator       | uuid      |                  |
+--------------------------+-----------+------------------+
3 row in set (0.01 sec)
```

SHOW UNUSED SHARDING KEY GENERATORS

```
mysql> SHOW UNUSED SHARDING KEY GENERATORS;
+--------------------------+-----------+------------------+
| name                     | type      | props            |
+--------------------------+-----------+------------------+
| uuid_key_generator       | uuid      |                  |
+--------------------------+-----------+------------------+
1 row in set (0.01 sec)
```

SHOW SHARDING TABLE RULES USED KEY GENERATOR keyGeneratorName

```
mysql> SHOW SHARDING TABLE RULES USED KEY GENERATOR keyGeneratorName;
+-------+---------+
| type  | name    |
+-------+---------+
| table | t_order |
+-------+---------+
1 row in set (0.01 sec)
```

SHOW DEFAULT SHARDING STRATEGY

```
mysql> SHOW DEFAULT SHARDING STRATEGY ;

+----------+---------+--------------------+-------------------------+--------------
-----------+-----------------------------------------+
| name     | type    | sharding_column    | sharding_algorithm_name | sharding_
algorithm_type | sharding_algorithm_props                 |
+----------+---------+--------------------+-------------------------+--------------
-----------+-----------------------------------------+
| TABLE    | NONE    |                    |                         |
          |                                          |
| DATABASE | STANDARD| order_id           | database_inline         | INLINE
          | {algorithm-expression=ds_${user_id % 2}} |
+----------+---------+--------------------+-------------------------+--------------
-----------+-----------------------------------------+
2 rows in set (0.07 sec)
```

SHOW SHARDING TABLE NODES

```
mysql> SHOW SHARDING TABLE NODES;
+---------+----------------------------------------------------------------+
| name    | nodes                                                          |
+---------+----------------------------------------------------------------+
| t_order | ds_0.t_order_0, ds_1.t_order_1, ds_0.t_order_2, ds_1.t_order_3 |
+---------+----------------------------------------------------------------+
1 row in set (0.02 sec)
```

Sharding Binding Table Rule

```
mysql> SHOW SHARDING BINDING TABLE RULES;
+----------------------+
| sharding_binding_tables |
+----------------------+
| t_order,t_order_item |
| t1,t2                |
+----------------------+
2 rows in set (0.00 sec)
```

Sharding Broadcast Table Rule

```
mysql> SHOW SHARDING BROADCAST TABLE RULES;
+------------------------+
| sharding_broadcast_tables |
+------------------------+
| t_1                    |
| t_2                    |
+------------------------+
2 rows in set (0.00 sec)
```

Single Table

Syntax

```
SHOW SINGLE TABLE (table | RULES) [FROM databaseName]

SHOW SINGLE TABLES

COUNT SINGLE_TABLE RULE [FROM databaseName]

table:
    TABLE tableName
```

Return Value Description

Single Table Rule

Column	Description
name	Rule name
resource_name	Data source name

Single Table

Column	Description
table_name	Single table name
resource_name	The resource name where the single table is located

Single Table Rule Count

列	说明
rule_name	Single table rule name
database	The database name where the single table is located
count	The count of single table rules

Example

SHOW SINGLE TABLES RULES

```
sql>  SHOW SINGLE TABLES RULES;
+---------+---------------+
| name    | resource_name |
+---------+---------------+
| default | ds_1          |
+---------+---------------+
1 row in set (0.01 sec)
```

SHOW SINGLE TABLE tableName

```
sql> SHOW SINGLE TABLE t_single_0;
+----------------+---------------+
| table_name     | resource_name |
+----------------+---------------+
| t_single_0     | ds_0          |
+----------------+---------------+
1 row in set (0.01 sec)
```

SHOW SINGLE TABLES

```
mysql> SHOW SINGLE TABLES;
+--------------+---------------+
| table_name   | resource_name |
+--------------+---------------+
| t_single_0   | ds_0          |
| t_single_1   | ds_1          |
+--------------+---------------+
2 rows in set (0.02 sec)
```

COUNT SINGLE_TABLE RULE

```
mysql> COUNT SINGLE_TABLE RULE;
+--------------+----------+-------+
| rule_name    | database | count |
+--------------+----------+-------+
| t_single_0   | ds       | 2     |
+--------------+----------+-------+
1 row in set (0.02 sec)
```

Readwrite-Splitting

Syntax

```
SHOW READWRITE_SPLITTING RULES [FROM databaseName]
```

Return Value Description

Column	Description
name	Rule name
auto_aware_data_source_name	Auto-Aware discovery data source name (Display configuration dynamic readwrite splitting rules)
writ e_data_source_query_enabled	All read data source are offline, write data source whether the data source is responsible for read traffic
write_data_source_name	Write data source name
read_data_source_names	Read data source name list
load_balancer_type	Load balance algorithm type
load_balancer_props	Load balance algorithm parameter

Example

Static Readwrite Splitting Rules

```
mysql> SHOW READWRITE_SPLITTING RULES;
+------------+---------------------------+------------------------+-------------
----------+-------------------+---------------------+
| name       | auto_aware_data_source_name | write_data_source_name | read_data_
source_names | load_balancer_type | load_balancer_props |
+------------+---------------------------+------------------------+-------------
----------+-------------------+---------------------+
| ms_group_0 |                             | ds_primary             | ds_slave_0,
ds_slave_1 | random            |                     |
+------------+---------------------------+------------------------+-------------
----------+-------------------+---------------------+
1 row in set (0.00 sec)
```

Dynamic Readwrite Splitting Rules

```
mysql> SHOW READWRITE_SPLITTING RULES FROM readwrite_splitting_db;
+--------------+-----------------------------+------------------------+-----------
-----------+--------------------+---------------------+
| name         | auto_aware_data_source_name | write_data_source_query_enabled |
write_data_source_name | read_data_source_names | load_balancer_type | load_
balancer_props |
+--------------+-----------------------------+------------------------+-----------
-----------+--------------------+---------------------+
| readwrite_ds | ms_group_0                  |                                 |
                       |                        | random             | read_weight=2:1
    |
+-------+-----------------------------+------------------------+------------------
-----+--------------------+---------------------+
1 row in set (0.01 sec)
```

Static Readwrite Splitting Rules And Dynamic Readwrite Splitting Rules

```
mysql> SHOW READWRITE_SPLITTING RULES FROM readwrite_splitting_db;
+--------------+-----------------------------+------------------------+-----------
-----------+--------------------+---------------------+
| name         | auto_aware_data_source_name | write_data_source_query_enabled |
write_data_source_name | read_data_source_names | load_balancer_type | load_
balancer_props |
+--------------+-----------------------------+------------------------+-----------
-----------+--------------------+---------------------+
| readwrite_ds | ms_group_0                  |                                 |
write_ds               | read_ds_0, read_ds_1   | random             | read_
weight=2:1     |
+-------+-----------------------------+------------------------+------------------
-----+--------------------+---------------------+
```

```
1 row in set (0.00 sec)
```

DB Discovery

Syntax

```
SHOW DB_DISCOVERY RULES [FROM databaseName]

SHOW DB_DISCOVERY TYPES [FROM databaseName]

SHOW DB_DISCOVERY HEARTBEATS [FROM databaseName]
```

Return Value Description

DB Discovery Rule

Column	Description
group_name	Rule name
data_source_names	Data source name list
primary_data_source_name	Primary data source name
discovery_type	Database discovery service type
discovery_heartbeat	Database discovery service heartbeat

DB Discovery Type

Column	Description
name	Type name
type	Type category
props	Type properties

DB Discovery Heartbeat

Column	Description
name	Heartbeat name
props	Heartbeat properties

Example

DB Discovery Rule

```
mysql> SHOW DB_DISCOVERY RULES;
+-----------------------+-------------------+-------------------------+------------
-------------------------------------------------------------------------+----------
-----------------------------------------------------------------+
| group_name            | data_source_names | primary_data_source_name | discovery_
type                                                                      |
discovery_heartbeat                                                          |
+-----------------------+-------------------+-------------------------+------------
-------------------------------------------------------------------------+----------
-----------------------------------------------------------------+
| db_discovery_group_0 | ds_0,ds_1,ds_2    |        ds_0              | {name=db_
discovery_group_0_mgr, type=MySQL.MGR, props={group-name=92504d5b-6dec}} |
{name=db_discovery_group_0_heartbeat, props={keep-alive-cron=0/5 * * * * ?}} |
+-----------------------+-------------------+-------------------------+------------
-------------------------------------------------------------------------+----------
-----------------------------------------------------------------+
1 row in set (0.20 sec)
```

DB Discovery Type

```
mysql> SHOW DB_DISCOVERY TYPES;
+---------------------------+------------+------------------------------+
| name                      | type       | props                        |
+---------------------------+------------+------------------------------+
| db_discovery_group_0_mgr  | MySQL.MGR  | {group-name=92504d5b-6dec}   |
+---------------------------+------------+------------------------------+
1 row in set (0.01 sec)
```

DB Discovery Heartbeat

```
mysql> SHOW DB_DISCOVERY HEARTBEATS;
+---------------------------------+---------------------------------+
| name                            | props                           |
+---------------------------------+---------------------------------+
| db_discovery_group_0_heartbeat | {keep-alive-cron=0/5 * * * * ?} |
+---------------------------------+---------------------------------+
1 row in set (0.01 sec)
```

Encrypt

Syntax

```
SHOW ENCRYPT RULES [FROM databaseName]

SHOW ENCRYPT TABLE RULE tableName [FROM databaseName]
```

- Support to query all data encryption rules and specify logical table name query

Return Value Description

Column	Description
table	Logical table name
logic_column	Logical column name
logic_data_type	Logical column data type
cipher_column	Ciphertext column name
cipher_data_type	Ciphertext column data type
plain_column	Plaintext column name
plain_data_type	Plaintext column data type
assisted_query_column	Assisted query column name
assisted_query_data_type	Assisted query column data type
encryptor_type	Encryption algorithm type
encryptor_props	Encryption algorithm parameter
query_with_cipher_column	Whether to use encrypted column for query

Example

Show Encrypt Rules

```
mysql> SHOW ENCRYPT RULES FROM encrypt_db;
+-------------+--------------+------------------+----------------+-------------------
+--------------+-----------------+-----------------------+-------------------------
-+---------------+-------------------------+---------------------------+
| table       | logic_column | logic_data_type | cipher_column | cipher_data_type |
plain_column | plain_data_type | assisted_query_column | assisted_query_data_type |
encryptor_type | encryptor_props         | query_with_cipher_column |
+-------------+--------------+------------------+----------------+-------------------
+--------------+-----------------+-----------------------+-------------------------
-+---------------+-------------------------+---------------------------+
| t_encrypt   | user_id      |                  | user_cipher   |                  |
user_plain   |                 |                       |                          |
AES            | aes-key-value=123456abc | true                     |
| t_encrypt   | order_id     |                  | order_cipher  |                  |
```

```
             |                 |                       |                          |
MD5            |                         | true                      |
| t_encrypt_2 | user_id      |                  | user_cipher   |                  |
user_plain   |                 |                       |                          |
AES            | aes-key-value=123456abc | false                     |
| t_encrypt_2 | order_id     |                  | order_cipher  |                  |
             |                 |                       |                          |
MD5            |                         | false                     |
+-------------+--------------+-----------------+---------------+------------------
+--------------+-----------------+-----------------------+-------------------------
-+---------------+-------------------------+-------------------------+
4 rows in set (0.78 sec)
```

Show Encrypt Table Rule Table Name

```
mysql> SHOW ENCRYPT TABLE RULE t_encrypt;
+-------------+--------------+-----------------+---------------+------------------
+--------------+-----------------+-----------------------+-------------------------
-+---------------+-------------------------+-------------------------+
| table       | logic_column | logic_data_type | cipher_column | cipher_data_type |
plain_column | plain_data_type | assisted_query_column | assisted_query_data_type |
encryptor_type | encryptor_props         | query_with_cipher_column |
+-------------+--------------+-----------------+---------------+------------------
+--------------+-----------------+-----------------------+-------------------------
-+---------------+-------------------------+-------------------------+
| t_encrypt   | user_id      |                  | user_cipher   |                  |
user_plain   |                 |                       |                          |
AES            | aes-key-value=123456abc | true                      |
| t_encrypt   | order_id     |                  | order_cipher  |                  |
             |                 |                       |                          |
MD5            |                         | true                      |
+-------------+--------------+-----------------+---------------+------------------
+--------------+-----------------+-----------------------+-------------------------
-+---------------+-------------------------+-------------------------+
2 rows in set (0.01 sec)

mysql> SHOW ENCRYPT TABLE RULE t_encrypt FROM encrypt_db;
+-------------+--------------+-----------------+---------------+------------------
+--------------+-----------------+-----------------------+-------------------------
-+---------------+-------------------------+-------------------------+
| table       | logic_column | logic_data_type | cipher_column | cipher_data_type |
plain_column | plain_data_type | assisted_query_column | assisted_query_data_type |
encryptor_type | encryptor_props         | query_with_cipher_column |
+-------------+--------------+-----------------+---------------+------------------
+--------------+-----------------+-----------------------+-------------------------
-+---------------+-------------------------+-------------------------+
| t_encrypt   | user_id      |                  | user_cipher   |                  |
user_plain   |                 |                       |                          |
```

```
AES              | aes-key-value=123456abc | true                        |
| t_encrypt    | order_id     |                   | order_cipher  |                   |
             |                 |                        |                           |
MD5              |                         | true                        |
+-------------+--------------+-----------------+---------------+------------------
+--------------+-----------------+-----------------------+------------------------
-+----------------+------------------------+--------------------------+
2 rows in set (0.01 sec))
```

Shadow

Syntax

```
SHOW SHADOW shadowRule | RULES [FROM databaseName]

SHOW SHADOW TABLE RULES [FROM databaseName]

SHOW SHADOW ALGORITHMS [FROM databaseName]

shadowRule:
    RULE ruleName
```

- Support querying all shadow rules and specified table query
- Support querying all shadow table rules
- Support querying all shadow algorithms

Return Value Description

Shadow Rule

Column	Description
rule_name	Rule name
source_name	Source database
shadow_name	Shadow database
shadow_table	Shadow table

Shadow Table Rule

Column	Description
shadow_table	Shadow table
shadow_algorithm_name	Shadow algorithm name

Shadow Algorithms

Column	Description
shadow_algorithm_name	Shadow algorithm name
type	Shadow algorithm type
props	Shadow algorithm properties
is_default	Default

Shadow Rule status

Column	Description
status	Enable

Example

SHOW SHADOW RULES

```
mysql> SHOW SHADOW RULES;
+--------------------+-------------+-------------+--------------+
| rule_name          | source_name | shadow_name | shadow_table |
+--------------------+-------------+-------------+--------------+
| shadow_rule_1      | ds_1        | ds_shadow_1 | t_order      |
| shadow_rule_2      | ds_2        | ds_shadow_2 | t_order_item |
+--------------------+-------------+-------------+--------------+
2 rows in set (0.02 sec)
```

SHOW SHADOW RULE ruleName

```
mysql> SHOW SHADOW RULE shadow_rule_1;
+------------------+-------------+-------------+--------------+
| rule_name        | source_name | shadow_name | shadow_table |
+------------------+-------------+-------------+--------------+
| shadow_rule_1    | ds_1        | ds_shadow_1 | t_order      |
+------------------+-------------+-------------+--------------+
1 rows in set (0.01 sec)
```

SHOW SHADOW TABLE RULES

```
mysql> SHOW SHADOW TABLE RULES;
+--------------+------------------------------------------------------------------
-------------+
| shadow_table | shadow_algorithm_name
            |
+--------------+------------------------------------------------------------------
-------------+
| t_order_1    | user_id_match_algorithm,simple_hint_algorithm_1
            |
+--------------+------------------------------------------------------------------
-------------+
1 rows in set (0.01 sec)
```

SHOW SHADOW ALGORITHMS

```
mysql> SHOW SHADOW ALGORITHMS;
+-------------------------+--------------------+-----------------------------------
--------+----------------+
| shadow_algorithm_name   | type               | props
       | is_default     |
+-------------------------+--------------------+-----------------------------------
--------+----------------+
| user_id_match_algorithm | REGEX_MATCH        | operation=insert,column=user_id,
regex=[1] | false          |
| simple_hint_algorithm_1 | SIMPLE_HINT        | shadow=true,foo=bar
       | false          |
+-------------------------+--------------------+-----------------------------------
--------+----------------+
2 rows in set (0.01 sec)
```

RAL Syntax

RAL (Resource & Rule Administration Language) responsible for hint, circuit breaker, configuration import and export, scaling control and other management functions.

Hint

Statement	Function	Example
SET READ WRITE_SPLITTING HINT SOURCE = [auto / write]	For current connection, set readwrite splitting routing strategy (automatic or forced to write data source)	SET **READWRITE_** SPLITTINGHINT SOURCE = write
SET SHARDING HINT DATABASE_VALUE = yy	For current connection, set sharding value for database sharding only, yy: sharding value	SET SHARDING HINT D ATABASE_VALUE = 100
ADD SHARDING HINT DATABASE_VALUE xx = yy	For current connection, add sharding value for table, xx: logic table, yy: database sharding value	ADD SHARDING HINT D ATABASE_VALUE t_order = 100
ADD SHARDING HINT TABLE_VALUE xx = yy	For current connection, add sharding value for table, xx: logic table, yy: table sharding value	ADD SHARDING HINT TABLE_VALUE t_order = 100
CLEAR HINT	For current connection, clear all hint settings	CLEAR HINT
CLEAR [SHARDING HINT / READ WRITE_SPLITTING HINT]	For current connection, clear hint settings of sharding or readwrite splitting	CLEAR READWR ITE_SPLITTING HINT
SHOW [SHARDING / READW RITE_SPLITTING] HINT STATUS	For current connection, query hint settings of sharding or readwrite splitting	SHOW READWR ITE_SPLITTING HINT STATUS

Scaling

Statement	Function	Example
SHOW MIGRATION LIST	Query running list	SHOW MIGRATION LIST
SHOW MIGRATION STATUS jobId	Query migration status, xx: jobId	SHOW MIGRATION STATUS 1234
START MIGRATION jobId	Start migration, xx: jobId	START MIGRATION 1234
STOP MIGRATION jobId	Stop migration, xx: jobId	STOP MIGRATION 1234
CLEAN MIGRATION jobId	Clean migration, xx: jobId	CLEAN MIGRATION 1234
RESET MIGRATION jobId	reset progress, xx: jobId	RESET MIGRATION 1234
CHECK MIGRATION jobId	Data consistency check with algorithm in `server.yaml`, xx: jobId	CHECK MIGRATION 1234
SHOW MIGRATION CHECK ALGORITHMS	Show available consistency check algorithms	SHOW MIGRATION CHECK ALGORITHMS
CHECK MIGRATION jobId (by type(name =algorithmTypeName)?	Data consistency check with defined algorithm	CHECK MIGRATION 1234 by type (name= "DEFAULT")
STOP MIGRATION SOURCE WRITING jobId	The source ShardingSphere data source is discontinued, xx: jobId	STOP MIGRATION SOURCE WRITING 1234
RESTORE MIGRATION SOURCE WRITING jobId	Restore source data source writing, xx: jobId	RESTORE MIGRATION SOURCE WRITING 1234
APPLY MIGRATION jobId	Switch to target ShardingSphere metadata, xx: jobId	APPLY MIGRATION 1234

Circuit Breaker

Statement	Function	Example
[ENABLE / DISABLE] READWRITE_SPLITTING (READ)? resourceName [FROM databaseName]	Enable or disable read data source	ENABLE READWRITE_SPLITTING READ resource_0
[ENABLE / DISABLE] INSTANCE instanceId	Enable or disable proxy instance	DISABLE INSTANCE instance_1
SHOW INSTANCE LIST	Query proxy instance information	SHOW INSTANCE LIST
SHOW READWRITE_SPLITTING (READ)? resourceName [FROM databaseName]	Query all read resources status	SHOW READWRITE_SPLITTING READ RESOURCES

Global Rule

Statement	Function	Example
SHOW AUTHORITY RULE	Query authority rule configuration	SHOW AUTHORITY RULE
SHOW TRANSACTION RULE	Query transaction rule configuration	SHOW TRANSACTION RULE
SHOW SQL_PARSER RULE	Query SQL parser rule configuration	SHOW SQL_PARSER RULE
ALTER TRANSACTION RULE(DEFAULT=xx,TYPE(NAME=xxx, PROPERTIES(key1=value1,key2=value2···)))	Alter transaction rule configuration, `DEFAULT`: default transaction type, support LOCAL, XA, BASE; `NAME`: name of transaction manager, support Atomikos, Narayana and Bitronix	ALTER TRANSACTION RULE(DEFAULT= "XA" ,TYPE(NAME= "Narayana" , PROPERTIES("databaseName" = "jbossts" , "host" = "127.0.0.1")))
ALTER SQL_PARSER RULE SQL_COMMENT_PARSE_ENABLE=xx, PARSE_TREE_CACHE (INITIAL_CAPACITY=xx, MAXIMUM_SIZE=xx, CONCURRENCY_LEVEL=xx), SQL_STATEMENT_CACHE(INITIAL_CAPACITY=xxx, MAXIMUM_SIZE=xxx, CONCURRENCY_LEVEL=xxx)	Alter SQL parser rule configuration, `SQL_COMMENT_PARSE_ENABLE`: whether to parse the SQL comment, `PARSE_TREE_CACHE`: local cache configuration of syntax tree, `SQL_STATEMENT_CACHE`: local cache of SQL statement	ALTER SQL_PARSER RULE SQL_COMMENT_PARSE_ENABLE=false, PARSE_TREE_CACHE(INITIAL_CAPACITY=10, MAXIMUM_SIZE=11, CONCURRENCY_LEVEL=1), SQL_STATEMENT_CACHE(INITIAL_CAPACITY=11, MAXIMUM_SIZE=11, CONCURRENCY_LEVEL=100)

Other

Statement	Function	Example
SHOW INSTANCE INFO	Query the instance information of the proxy	SHOW INSTANCE INFO
SHOW MODE INFO	Query the mode configuration of the proxy	SHOW MODE INFO
COUNT DATABASE RULES [FROM database]	Query the number of rules in a database	COUNT DATABASE RULES
SET VARIABLE prox y_property_name = xx	proxy_property_name is one of properties configuration of proxy, name is split by underscore	SET VARIABLE sql_show = true
SET VARIABLE t ransaction_type = xx	Modify transaction_type of the current connection, supports LOCAL, XA, BASE	SET VARIABLE trans action_type = "XA"
SET VARIABLE **agent_** plugins_enabled = [TRUE / FALSE]	Set whether the agent plugins are enabled, the default value is false	SET VARIABLE agent_plug ins_enabled = TRUE
SHOW ALL VARIABLES	Query proxy all properties configuration	SHOW ALL VARIABLES
SHOW VARIABLE variable_name	Query proxy variable, name is split by underscore	SHOW VARIABLE sql_show
REFRESH TABLE METADATA	Refresh the metadata of all tables	REFRESH TABLE METADATA
REFRESH TABLE METADATA tableName	Refresh the metadata of the specified table	REFRESH TABLE METADATA t_order
REFRESH TABLE METADATA tableName FROM RESOURCE resourceName	Refresh the tables' metadata in the specified data source	REFRESH TABLE METADATA t_order FROM RESOURCE ds_1
REFRESH TABLE METADATA FROM RESOURCE resourceName SCHEMA schemaName	Refresh the tables' metadata in a schema of a specified data source. If there are no tables in the schema, the schema will be deleted.	REFRESH TABLE METADATA FROM RESOURCE ds_1 SCHEMA db_schema
SHOW TABLE METADATA tableName [, tableName] …	Query table metadata	SHOW TABLE METADATA t_order
EXPORT DATABASE CONFIG [FROM database_name] [, fi le= "file_path"]	Export resources and rule configurations to YAML format	EXPORT DATABASE CONFIG FROM readwrite_s plitting_db
IMPORT DATABASE CONFIG F ILE= "file_path"	Import resources and rule configuration from YAML, only supports import into an empty database	IMPORT DATABASE CONFIG FILE = "/xxx /config-sha rding.yaml"
SHOW RULES USED RESOURCE resourceName [from database]	Query the rules for using the specified resource in database	SHOW RULES USED RESOURCE ds_0 FROM d atabaseName

Notice

ShardingSphere-Proxy does not support hint by default, to support it, set `proxy-hint-enabled` to true in `conf/server.yaml`.

RUL Syntax

RUL (Resource Utility Language) responsible for SQL parsing, SQL formatting, preview execution plan and more utility functions.

SQL Utility

St atement	Function	Example
PARSE SQL	Parse SQL and output abstract syntax tree	PARSE SELECT * FROM t_order
FORMAT SQL	Parse SQL and output formated SQL statement	FORMAT SELECT * FROM t_order
PREVIEW SQL	Preview SQL execution plan	PREVIEW SELECT * FROM t_order

Usage

This chapter will introduce how to use DistSQL to manage resources and rules in a distributed database.

Pre-work

Use MySQL as example, can replace to other databases.

1. Start the MySQL service;
2. Create to be registered MySQL databases;
3. Create role and user in MySQL with creation permission for ShardingSphere-Proxy;
4. Start Zookeeper service;
5. Add `mode` and `authentication` configurations to `server.yaml`;
6. Start ShardingSphere-Proxy;
7. Use SDK or terminal connect to ShardingSphere-Proxy.

Create Logic Database

1. Create logic database

```sql
CREATE DATABASE foo_db;
```

2. Use newly created logic database

```sql
USE foo_db;
```

Resource Operation

More details please see concentrate rule examples.

Rule Operation

More details please see concentrate rule examples.

Notice

1. Currently, `DROP DATABASE` will only remove the `logical distributed database`, not the user' s actual database;
2. `DROP TABLE` will delete all logical fragmented tables and actual tables in the database;
3. `CREATE DATABASE` will only create a `logical distributed database`, so users need to create actual databases in advance.

Sharding

Resource Operation

- Configure data source information

```sql
ADD RESOURCE ds_0 (
    HOST="127.0.0.1",
    PORT=3306,
    DB="ds_1",
    USER="root",
    PASSWORD="root"
),ds_1 (
    HOST="127.0.0.1",
    PORT=3306,
    DB="ds_2",
    USER="root",
```

```
    PASSWORD="root"
);
```

Rule Operation

- Create sharding rule

```
CREATE SHARDING TABLE RULE t_order(
RESOURCES(ds_0,ds_1),
SHARDING_COLUMN=order_id,
TYPE(NAME="hash_mod",PROPERTIES("sharding-count"="4")),
KEY_GENERATE_STRATEGY(COLUMN=order_id,TYPE(NAME="snowflake"))
);
```

- Create sharding table

```
CREATE TABLE `t_order` (
  `order_id` int NOT NULL,
  `user_id` int NOT NULL,
  `status` varchar(45) DEFAULT NULL,
  PRIMARY KEY (`order_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4
```

- Drop sharding table

```
DROP TABLE t_order;
```

- Drop sharding rule

```
DROP SHARDING TABLE RULE t_order;
```

- Drop resource

```
DROP RESOURCE ds_0, ds_1;
```

- Drop distributed database

```
DROP DATABASE foo_db;
```

Readwrite_splitting

Resource Operation

```sql
ADD RESOURCE write_ds (
    HOST="127.0.0.1",
    PORT=3306,
    DB="ds_0",
    USER="root",
    PASSWORD="root"
),read_ds (
    HOST="127.0.0.1",
    PORT=3307,
    DB="ds_0",
    USER="root",
    PASSWORD="root"
);
```

Rule Operation

- Create readwrite_splitting rule

```sql
CREATE READWRITE_SPLITTING RULE group_0 (
WRITE_RESOURCE=write_ds,
READ_RESOURCES(read_ds),
TYPE(NAME="random")
);
```

- Alter readwrite_splitting rule

```sql
ALTER READWRITE_SPLITTING RULE group_0 (
WRITE_RESOURCE=write_ds,
READ_RESOURCES(read_ds),
TYPE(NAME="random",PROPERTIES("read_weight"="2:0"))
);
```

- Drop readwrite_splitting rule

```sql
DROP READWRITE_SPLITTING RULE group_0;
```

- Drop resource

```sql
DROP RESOURCE write_ds,read_ds;
```

- Drop distributed database

```sql
DROP DATABASE readwrite_splitting_db;
```

Encrypt

Resource Operation

```sql
ADD RESOURCE ds_0 (
    HOST="127.0.0.1",
    PORT=3306,
    DB="ds_0",
    USER="root",
    PASSWORD="root"
);
```

Rule Operation

- Create encrypt rule

```sql
CREATE ENCRYPT RULE t_encrypt (
    COLUMNS(
        (NAME=user_id,PLAIN=user_plain,CIPHER=user_cipher,TYPE(NAME='AES',
PROPERTIES('aes-key-value'='123456abc'))),
        (NAME=order_id,PLAIN=order_plain,CIPHER =order_cipher,TYPE(NAME='RC4',
PROPERTIES('rc4-key-value'='123456abc')))
));
```

- Create encrypt table

```sql
CREATE TABLE `t_encrypt` (
    `id` int(11) NOT NULL,
    `user_id` varchar(45) DEFAULT NULL,
    `order_id` varchar(45) DEFAULT NULL,
    PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

- Alter encrypt rule

```sql
ALTER ENCRYPT RULE t_encrypt (
    COLUMNS(
        (NAME=user_id,PLAIN=user_plain,CIPHER=user_cipher,TYPE(NAME='AES',
PROPERTIES('aes-key-value'='123456abc')))
));
```

- Drop encrypt rule

```
DROP ENCRYPT RULE t_encrypt;
```

- Drop resource

```
DROP RESOURCE ds_0;
```

- Drop distributed database

```
DROP DATABASE encrypt_db;
```

DB Discovery

Resource Operation

```
ADD RESOURCE ds_0 (
    HOST="127.0.0.1",
    PORT=3306,
    DB="ds_0",
    USER="root",
    PASSWORD="root"
),ds_1 (
    HOST="127.0.0.1",
    PORT=3306,
    DB="ds_1",
    USER="root",
    PASSWORD="root"
),ds_2 (
    HOST="127.0.0.1",
    PORT=3306,
    DB="ds_2",
    USER="root",
    PASSWORD="root"
);
```

Rule Operation

- Create DB discovery rule

```
CREATE DB_DISCOVERY RULE db_discovery_group_0 (
RESOURCES(ds_0, ds_1),
TYPE(NAME='MySQL.MGR',PROPERTIES('group-name'='92504d5b-6dec')),
HEARTBEAT(PROPERTIES('keep-alive-cron'='0/5 * * * * ?'))
);
```

- Alter DB discovery rule

```
ALTER DB_DISCOVERY RULE db_discovery_group_0 (
RESOURCES(ds_0, ds_1, ds_2),
TYPE(NAME='MySQL.MGR',PROPERTIES('group-name'='92504d5b-6dec')),
HEARTBEAT(PROPERTIES('keep-alive-cron'='0/5 * * * * ?'))
);
```

- Drop db_discovery rule

```
DROP DB_DISCOVERY RULE db_discovery_group_0;
```

- Drop db_discovery type

```
DROP DB_DISCOVERY TYPE db_discovery_group_0_mgr;
```

- Drop db_discovery heartbeat

```
DROP DB_DISCOVERY HEARTBEAT db_discovery_group_0_heartbeat;
```

- Drop resource

```
DROP RESOURCE ds_0,ds_1,ds_2;
```

- Drop distributed database

```
DROP DATABASE discovery_db;
```

Shadow

Resource Operation

```
ADD RESOURCE ds_0 (
    HOST="127.0.0.1",
    PORT=3306,
    DB="ds_0",
    USER="root",
    PASSWORD="root"
),ds_1 (
    HOST="127.0.0.1",
    PORT=3306,
    DB="ds_1",
    USER="root",
    PASSWORD="root"
),ds_2 (
    HOST="127.0.0.1",
    PORT=3306,
    DB="ds_2",
    USER="root",
```

```
    PASSWORD="root"
);
```

Rule Operation

- Create shadow rule

```
CREATE SHADOW RULE group_0(
SOURCE=ds_0,
SHADOW=ds_1,
t_order((simple_hint_algorithm, TYPE(NAME="SIMPLE_HINT", PROPERTIES("foo"="bar"))),
(TYPE(NAME="REGEX_MATCH", PROPERTIES("operation"="insert","column"="user_id",
"regex"='[1]')))),
t_order_item((TYPE(NAME="SIMPLE_HINT", PROPERTIES("foo"="bar")))));
```

- Alter shadow rule

```
ALTER SHADOW RULE group_0(
SOURCE=ds_0,
SHADOW=ds_2,
t_order_item((TYPE(NAME="SIMPLE_HINT", PROPERTIES("foo"="bar")))));
```

- Drop shadow rule

```
DROP SHADOW RULE group_0;
```

- Drop resource

```
DROP RESOURCE ds_0,ds_1,ds_2;
```

- Drop distributed database

```
DROP DATABASE foo_db;
```

6.2.4 Data Migration

Introduction

ShardingSphere provides solution of migrating data since **4.1.0**, current state is **Experimental** version.

Build

Background

For systems running on a single database that urgently need to securely and simply migrate data to a horizontally sharded database.

Prerequisites

- Proxy is developed in JAVA, and JDK version 1.8 or later is recommended.
- Data migration adopts the cluster mode, and ZooKeeper is currently supported as the registry.

Procedure

1. Run the following command to compile the ShardingSphere-Proxy binary package:

```
git clone --depth 1 https://github.com/apache/shardingsphere.git
cd shardingsphere
mvn clean install -Dmaven.javadoc.skip=true -Dcheckstyle.skip=true -Drat.skip=true
-Djacoco.skip=true -DskipITs -DskipTests -Prelease
```

Release package: - /shardingsphere-distribution/shardingsphere-proxy-distribution/target/apache-shardingsphere-${latest.release.version}-shardingsphere-proxy-bin.tar.gz

Or you can get the installation package through the Download Page

2. Decompress the proxy release package and modify the configuration file `conf/config-sharding.yaml`. Please refer to proxy startup guide for details.
3. Modify the configuration file `conf/server.yaml`. Please refer to mode configuration for details.

Currently, `mode` must be `Cluster`, and the corresponding registry must be started in advance.

Configuration sample:

```
mode:
  type: Cluster
  repository:
    type: ZooKeeper
    props:
      namespace: governance_ds
      server-lists: localhost:2181
      retryIntervalMilliseconds: 500
      timeToLiveSeconds: 60
      maxRetries: 3
      operationTimeoutMilliseconds: 500
  overwrite: false
```

4. Introduce JDBC driver.

If the backend is connected to the following databases, download the corresponding JDBC driver jar package and put it into the ${shardingsphere-proxy}/lib directory.

Database	JDBC Driver	Reference
MySQL	**`mysql-co nnector-java-5.1.47.jar < https://repo1.maven.org/m aven2/mysql/mysql-connect or-java/5.1.47/mysql-conn ector-java-5.1.47.jar>`__**	Connector/J Versions
openGauss	opengauss-jd bc-2.0.1-compatibility.ja r	

5. Start ShardingSphere-Proxy:

```
sh bin/start.sh
```

6. View the proxy log `logs/stdout.log`. If you see the following statements:

```
[INFO ] [main] o.a.s.p.frontend.ShardingSphereProxy - ShardingSphere-Proxy start
success
```

The startup will have been successful.

7. Configure and migrate on demand.

7.1. Query configuration.

```
SHOW MIGRATION PROCESS CONFIGURATION;
```

The default configuration is as follows.

```
+--------------------------------------------------------------+--------------------
--------------------+------------------------------------------------------+
| read                                                         | write
                  | stream_channel                                         |
+--------------------------------------------------------------+--------------------
--------------------+------------------------------------------------------+
| {"workerThread":40,"batchSize":1000,"shardingSize":10000000} | {"workerThread
":40,"batchSize":1000} | {"type":"MEMORY","props":{"block-queue-size":10000}} |
+--------------------------------------------------------------+--------------------
--------------------+------------------------------------------------------+
```

7.2. New configuration (Optional).

A default value is available if there is no configuration.

A completely configured DistSQL is as follows.

```
CREATE MIGRATION PROCESS CONFIGURATION (
READ(
  WORKER_THREAD=40,
```

```
  BATCH_SIZE=1000,
  SHARDING_SIZE=10000000,
  RATE_LIMITER (TYPE(NAME='QPS',PROPERTIES('qps'='500')))
),
WRITE(
  WORKER_THREAD=40,
  BATCH_SIZE=1000,
  RATE_LIMITER (TYPE(NAME='TPS',PROPERTIES('tps'='2000')))
),
STREAM_CHANNEL (TYPE(NAME='MEMORY',PROPERTIES('block-queue-size'='10000')))
);
```

Configuration item description:

```
CREATE MIGRATION PROCESS CONFIGURATION (
READ( -- Data reading configuration. If it is not configured, part of the
parameters will take effect by default.
  WORKER_THREAD=40, -- Obtain the thread pool size of all the data from the source
side. If it is not configured, the default value is used.
  BATCH_SIZE=1000, -- The maximum number of records returned by a query operation.
If it is not configured, the default value is used.
  SHARDING_SIZE=10000000, -- Sharding size of all the data. If it is not
configured, the default value is used.
  RATE_LIMITER ( -- Traffic limit algorithm. If it is not configured, traffic is
not limited.
  TYPE( -- Algorithm type. Option: QPS
  NAME='QPS',
  PROPERTIES( -- Algorithm property
  'qps'='500'
  )))
),
WRITE( -- Data writing configuration. If it is not configured, part of the
parameters will take effect by default.
  WORKER_THREAD=40, -- The size of the thread pool on which data is written into
the target side. If it is not configured, the default value is used.
  BATCH_SIZE=1000, -- The maximum number of records for a batch write operation. If
it is not configured, the default value is used.
  RATE_LIMITER ( -- Traffic limit algorithm. If it is not configured, traffic is
not limited.
  TYPE( -- Algorithm type. Option: TPS
  NAME='TPS',
  PROPERTIES( -- Algorithm property.
  'tps'='2000'
  )))
),
STREAM_CHANNEL ( -- Data channel. It connects producers and consumers, used for
reading and writing procedures. If it is not configured, the MEMORY type is used by
default.
```

```
TYPE( -- Algorithm type. Option: MEMORY
NAME='MEMORY',
PROPERTIES( -- Algorithm property
'block-queue-size'='10000' -- Property: blocking queue size.
)))
);
```

DistSQL sample: configure READ for traffic limit.

```
CREATE MIGRATION PROCESS CONFIGURATION (
READ(
  RATE_LIMITER (TYPE(NAME='QPS',PROPERTIES('qps'='500')))
)
);
```

Configure data reading for traffic limit. Other configurations use default values.

7.3. Modify configuration.

ALTER MIGRATION PROCESS CONFIGURATION, and its internal structure is the same as that of CREATE MIGRATION PROCESS CONFIGURATION.

DistSQL sample: modify traffic limit parameter

```
ALTER MIGRATION PROCESS CONFIGURATION (
READ(
  RATE_LIMITER (TYPE(NAME='QPS',PROPERTIES('qps'='1000')))
)
);
---
ALTER MIGRATION PROCESS CONFIGURATION (
READ(
  RATE_LIMITER (TYPE(NAME='QPS',PROPERTIES('qps'='1000')))
), WRITE(
  RATE_LIMITER (TYPE(NAME='TPS',PROPERTIES('tps'='1000')))
)
);
```

7.4. Clear configuration.

DistSQL sample: clear the configuration of READ and restore it to the default value.

```
DROP MIGRATION PROCESS CONFIGURATION '/READ';
```

DistSQL sample: clear the configuration of READ/RATE_LIMITER.

```
DROP MIGRATION PROCESS CONFIGURATION '/READ/RATE_LIMITER';
```

Manual

MySQL user guide

Environment

Supported MySQL versions: 5.1.15 to 5.7.x.

Authority required

1. Enable binlog

MySQL 5.7 my.cnf configuration sample:

```
[mysqld]
server-id=1
log-bin=mysql-bin
binlog-format=row
binlog-row-image=full
max_connections=600
```

Run the following command and check whether binlog is enabled.

```
show variables like '%log_bin%';
show variables like '%binlog%';
```

If the following information is displayed, binlog is enabled.

```
+-----------------------------------------+---------------------------------------+
| Variable_name                           | Value                                 |
+-----------------------------------------+---------------------------------------+
| log_bin                                 | ON                                    |
| binlog_format                           | ROW                                   |
| binlog_row_image                        | FULL                                  |
+-----------------------------------------+---------------------------------------+
```

2. Grant Replication-related permissions for MySQL account.

Run the following command and see whether the user has migration permission.

```
SHOW GRANTS FOR 'user';
```

Result sample:

```
+------------------------------------------------------------------------------+
|Grants for ${username}@${host}                                                |
+------------------------------------------------------------------------------+
|GRANT REPLICATION SLAVE, REPLICATION CLIENT ON *.* TO ${username}@${host}     |
```

```
|.......                                                                      |
+-----------------------------------------------------------------------------+
```

Complete procedure example

Prerequisite

1. Prepare the source database, table, and data in MySQL.

Sample:

```
DROP DATABASE IF EXISTS migration_ds_0;
CREATE DATABASE migration_ds_0 DEFAULT CHARSET utf8;

USE migration_ds_0

CREATE TABLE t_order (order_id INT NOT NULL, user_id INT NOT NULL, status
VARCHAR(45) NULL, PRIMARY KEY (order_id));

INSERT INTO t_order (order_id, user_id, status) VALUES (1,2,'ok'),(2,4,'ok'),(3,6,
'ok'),(4,1,'ok'),(5,3,'ok'),(6,5,'ok');
```

2. Prepare the target database in MySQL.

Sample:

```
DROP DATABASE IF EXISTS migration_ds_10;
CREATE DATABASE migration_ds_10 DEFAULT CHARSET utf8;

DROP DATABASE IF EXISTS migration_ds_11;
CREATE DATABASE migration_ds_11 DEFAULT CHARSET utf8;

DROP DATABASE IF EXISTS migration_ds_12;
CREATE DATABASE migration_ds_12 DEFAULT CHARSET utf8;
```

Procedure

1. Create a new logical database in proxy and configure resources and rules.

```
CREATE DATABASE sharding_db;

USE sharding_db

ADD RESOURCE ds_2 (
    URL="jdbc:mysql://127.0.0.1:3306/migration_ds_10?serverTimezone=UTC&
useSSL=false",
    USER="root",
```

```
    PASSWORD="root",
    PROPERTIES("minPoolSize"="1","maxPoolSize"="20","idleTimeout"="60000")
), ds_3 (
    URL="jdbc:mysql://127.0.0.1:3306/migration_ds_11?serverTimezone=UTC&
useSSL=false",
    USER="root",
    PASSWORD="root",
    PROPERTIES("minPoolSize"="1","maxPoolSize"="20","idleTimeout"="60000")
), ds_4 (
    URL="jdbc:mysql://127.0.0.1:3306/migration_ds_12?serverTimezone=UTC&
useSSL=false",
    USER="root",
    PASSWORD="root",
    PROPERTIES("minPoolSize"="1","maxPoolSize"="20","idleTimeout"="60000")
);

CREATE SHARDING TABLE RULE t_order(
RESOURCES(ds_2,ds_3,ds_4),
SHARDING_COLUMN=order_id,
TYPE(NAME="hash_mod",PROPERTIES("sharding-count"="6")),
KEY_GENERATE_STRATEGY(COLUMN=order_id,TYPE(NAME="snowflake"))
);
```

If you are migrating to a heterogeneous database, you need to execute the table-creation statements in proxy.

2. Configure the source resources in proxy.

```
ADD MIGRATION SOURCE RESOURCE ds_0 (
    URL="jdbc:mysql://127.0.0.1:3306/migration_ds_0?serverTimezone=UTC&useSSL=false
",
    USER="root",
    PASSWORD="root",
    PROPERTIES("minPoolSize"="1","maxPoolSize"="20","idleTimeout"="60000")
);
```

3. Start data migration.

```
MIGRATE TABLE ds_0.t_order INTO t_order;
```

Or you can specify a target logical database.

```
MIGRATE TABLE ds_0.t_order INTO sharding_db.t_order;
```

4. Check the data migration job list.

```
SHOW MIGRATION LIST;
```

Result example:

```
+--------------------------------------+----------+----------------------+--------+--
-------------------+------------+
| id                                   | tables   | sharding_total_count | active |
create_time         | stop_time |
+--------------------------------------+----------+----------------------+--------+--
-------------------+------------+
| j015d4ee1b8a5e7f95df19babb2794395e8 | t_order | 1                    | true   |
2022-08-22 16:37:01 | NULL      |
+--------------------------------------+----------+----------------------+--------+--
-------------------+------------+
```

5. View the data migration details.

```
SHOW MIGRATION STATUS 'j015d4ee1b8a5e7f95df19babb2794395e8';
+------+-------------+---------------------------+--------+-----------------------
------+--------------------------+
| item | data_source | status                    | active | inventory_finished_
percentage | incremental_idle_seconds |
+------+-------------+---------------------------+--------+-----------------------
------+--------------------------+
| 0    | ds_0        | EXECUTE_INCREMENTAL_TASK | true   | 100
     | 141                      |
+------+-------------+---------------------------+--------+-----------------------
------+--------------------------+
```

6. Verify data consistency.

```
CHECK MIGRATION 'j015d4ee1b8a5e7f95df19babb2794395e8' BY TYPE (NAME='CRC32_MATCH');
+------------+----------------------+----------------------+-----------------------
+-------------------------+
| table_name | source_records_count | target_records_count | records_count_matched
| records_content_matched |
+------------+----------------------+----------------------+-----------------------
+-------------------------+
| t_order    | 6                    | 6                    | true
| true                    |
+------------+----------------------+----------------------+-----------------------
+-------------------------+
```

Data consistency check algorithm list:

```
SHOW MIGRATION CHECK ALGORITHMS;
+-------------+--------------------------------------------------------------+-----
----------------------+
| type        | supported_database_types                                     |
description               |
+-------------+--------------------------------------------------------------+-----
----------------------+
| CRC32_MATCH | MySQL                                                        |
```

```
Match CRC32 of records.    |
| DATA_MATCH  | SQL92,MySQL,MariaDB,PostgreSQL,openGauss,Oracle,SQLServer,H2 |
Match raw data of records. |
+-------------+--------------------------------------------------------------+-----
-----------------------+
```

If encrypt rule is configured in target proxy, then CRC32_MATCH could be not used.

7. Stop the job.

```
STOP MIGRATION 'j015d4ee1b8a5e7f95df19babb2794395e8';
```

8. Clear the job.

```
CLEAN MIGRATION 'j015d4ee1b8a5e7f95df19babb2794395e8';
```

PostgreSQL user guide

Environment

Supported PostgreSQL version: 9.4 or later.

Authority required

1. Enable test_decoding.
2. Modify WAL Configuration.

postgresql.conf configuration sample:

```
wal_level = logical
max_wal_senders = 10
max_replication_slots = 10
max_connections = 600
```

Please refer to Write Ahead Log and Replication for details.

3. Configure PostgreSQL and grant Proxy the replication permission.

pg_hba.conf instance configuration:

```
host replication repl_acct 0.0.0.0/0 md5
```

Please refer to The pg_hba.conf File for details.

Complete procedure example

Prerequisite

1. Prepare the source database, table, and data in PostgreSQL.

```
DROP DATABASE IF EXISTS migration_ds_0;
CREATE DATABASE migration_ds_0;

\c migration_ds_0

CREATE TABLE t_order (order_id INT NOT NULL, user_id INT NOT NULL, status
VARCHAR(45) NULL, PRIMARY KEY (order_id));

INSERT INTO t_order (order_id, user_id, status) VALUES (1,2,'ok'),(2,4,'ok'),(3,6,
'ok'),(4,1,'ok'),(5,3,'ok'),(6,5,'ok');
```

2. Prepare the target database in PostgreSQL.

```
DROP DATABASE IF EXISTS migration_ds_10;
CREATE DATABASE migration_ds_10;

DROP DATABASE IF EXISTS migration_ds_11;
CREATE DATABASE migration_ds_11;

DROP DATABASE IF EXISTS migration_ds_12;
CREATE DATABASE migration_ds_12;
```

Procedure

1. Create a new logical database in proxy and configure resources and rules.

```
CREATE DATABASE sharding_db;

\c sharding_db

ADD RESOURCE ds_2 (
    URL="jdbc:postgresql://127.0.0.1:5432/migration_ds_10",
    USER="postgres",
    PASSWORD="root",
    PROPERTIES("minPoolSize"="1","maxPoolSize"="20","idleTimeout"="60000")
), ds_3 (
    URL="jdbc:postgresql://127.0.0.1:5432/migration_ds_11",
    USER="postgres",
    PASSWORD="root",
    PROPERTIES("minPoolSize"="1","maxPoolSize"="20","idleTimeout"="60000")
), ds_4 (
```

```
    URL="jdbc:postgresql://127.0.0.1:5432/migration_ds_12",
    USER="postgres",
    PASSWORD="root",
    PROPERTIES("minPoolSize"="1","maxPoolSize"="20","idleTimeout"="60000")
);

CREATE SHARDING TABLE RULE t_order(
RESOURCES(ds_2,ds_3,ds_4),
SHARDING_COLUMN=order_id,
TYPE(NAME="hash_mod",PROPERTIES("sharding-count"="6")),
KEY_GENERATE_STRATEGY(COLUMN=order_id,TYPE(NAME="snowflake"))
);
```

If you are migrating to a heterogeneous database, you need to execute the table-creation statements in proxy.

2. Configure the source resources in proxy.

```
ADD MIGRATION SOURCE RESOURCE ds_0 (
    URL="jdbc:postgresql://127.0.0.1:5432/migration_ds_0",
    USER="postgres",
    PASSWORD="root",
    PROPERTIES("minPoolSize"="1","maxPoolSize"="20","idleTimeout"="60000")
);
```

3. Enable data migration.

```
MIGRATE TABLE ds_0.t_order INTO t_order;
```

Or you can specify a target logical database.

```
MIGRATE TABLE ds_0.t_order INTO sharding_db.t_order;
```

Or you can specify a source schema name.

```
MIGRATE TABLE ds_0.public.t_order INTO sharding_db.t_order;
```

4. Check the data migration job list.

```
SHOW MIGRATION LIST;
```

Result sample:

```
+---------------------------------------+----------+-----------------------+---------+--
-------------------+------------+
| id                                    | tables   | sharding_total_count | active |
create_time          | stop_time |
+---------------------------------------+----------+-----------------------+---------+--
-------------------+------------+
```

```
| j015d4ee1b8a5e7f95df19babb2794395e8 | t_order | 1                     | true    |
2022-08-22 16:37:01 | NULL       |
+------------------------------------+---------+----------------------+--------+--
-------------------+-----------+
```

5. View the data migration details.

```
SHOW MIGRATION STATUS 'j015d4ee1b8a5e7f95df19babb2794395e8';
+------+-------------+--------------------------+--------+-------------------------
------+--------------------------+
| item | data_source | status                   | active | inventory_finished_
percentage | incremental_idle_seconds |
+------+-------------+--------------------------+--------+-------------------------
------+--------------------------+
| 0    | ds_0        | EXECUTE_INCREMENTAL_TASK | true   | 100
     | 141                      |
+------+-------------+--------------------------+--------+-------------------------
------+--------------------------+
```

6. Verify data consistency.

```
CHECK MIGRATION 'j015d4ee1b8a5e7f95df19babb2794395e8' BY TYPE (NAME='DATA_MATCH');
+------------+----------------------+----------------------+-----------------------
+-------------------------+
| table_name | source_records_count | target_records_count | records_count_matched
| records_content_matched |
+------------+----------------------+----------------------+-----------------------
+-------------------------+
| t_order    | 6                    | 6                    | true
| true                    |
+------------+----------------------+----------------------+-----------------------
+-------------------------+
```

7. Stop the job.

```
STOP MIGRATION 'j015d4ee1b8a5e7f95df19babb2794395e8';
```

8. Clear the job.

```
CLEAN MIGRATION 'j015d4ee1b8a5e7f95df19babb2794395e8';
```

openGauss user guide

Environment

Supported openGauss version: 2.0.1 to 3.0.0.

Authority required

1. Modify WAL configuration.

`postgresql.conf` configuration sample:

```
wal_level = logical
max_wal_senders = 10
max_replication_slots = 10
wal_sender_timeout = 0
max_connections = 600
```

Please refer to Write Ahead Log and Replication for details.

2. Configure PostgreSQL and grant Proxy the replication permission.

`pg_hba.conf` instance configuration:

```
host replication repl_acct 0.0.0.0/0 md5
```

Please refer to Configuring Client Access Authentication and Example: Logic Replication Code for details.

Complete procedure example

Prerequisite

1. Prepare the source database, table, and data in openGauss.

```
DROP DATABASE IF EXISTS migration_ds_0;
CREATE DATABASE migration_ds_0;

\c migration_ds_0

CREATE TABLE t_order (order_id INT NOT NULL, user_id INT NOT NULL, status
VARCHAR(45) NULL, PRIMARY KEY (order_id));

INSERT INTO t_order (order_id, user_id, status) VALUES (1,2,'ok'),(2,4,'ok'),(3,6,
'ok'),(4,1,'ok'),(5,3,'ok'),(6,5,'ok');
```

2. Prepare the target database in openGauss.

```sql
DROP DATABASE IF EXISTS migration_ds_10;
CREATE DATABASE migration_ds_10;

DROP DATABASE IF EXISTS migration_ds_11;
CREATE DATABASE migration_ds_11;

DROP DATABASE IF EXISTS migration_ds_12;
CREATE DATABASE migration_ds_12;
```

Procedure

1. Create a new logical database and configure resources and rules.

```
CREATE DATABASE sharding_db;

\c sharding_db

ADD RESOURCE ds_2 (
    URL="jdbc:opengauss://127.0.0.1:5432/migration_ds_10",
    USER="gaussdb",
    PASSWORD="Root@123",
    PROPERTIES("minPoolSize"="1","maxPoolSize"="20","idleTimeout"="60000")
), ds_3 (
    URL="jdbc:opengauss://127.0.0.1:5432/migration_ds_11",
    USER="gaussdb",
    PASSWORD="Root@123",
    PROPERTIES("minPoolSize"="1","maxPoolSize"="20","idleTimeout"="60000")
), ds_4 (
    URL="jdbc:opengauss://127.0.0.1:5432/migration_ds_12",
    USER="gaussdb",
    PASSWORD="Root@123",
    PROPERTIES("minPoolSize"="1","maxPoolSize"="20","idleTimeout"="60000")
);

CREATE SHARDING TABLE RULE t_order(
RESOURCES(ds_2,ds_3,ds_4),
SHARDING_COLUMN=order_id,
TYPE(NAME="hash_mod",PROPERTIES("sharding-count"="6")),
KEY_GENERATE_STRATEGY(COLUMN=order_id,TYPE(NAME="snowflake"))
);
```

If you are migrating to a heterogeneous database, you need to execute the table-creation statements in proxy.

2. Configure the source resources in proxy.

```
ADD MIGRATION SOURCE RESOURCE ds_2 (
    URL="jdbc:opengauss://127.0.0.1:5432/migration_ds_0",
    USER="gaussdb",
    PASSWORD="Root@123",
    PROPERTIES("minPoolSize"="1","maxPoolSize"="20","idleTimeout"="60000")
);
```

3. Enable data migration.

```
MIGRATE TABLE ds_0.t_order INTO t_order;
```

Or you can specify a target logical database.

```
MIGRATE TABLE ds_0.t_order INTO sharding_db.t_order;
```

Or you can specify a source schema name.

```
MIGRATE TABLE ds_0.public.t_order INTO sharding_db.t_order;
```

4. Check the data migration job list.

```
SHOW MIGRATION LIST;
```

Result example:

```
+---------------------------------------+----------+-----------------------+--------+--
-------------------+-----------+
| id                                    | tables   | sharding_total_count | active |
create_time          | stop_time |
+---------------------------------------+----------+-----------------------+--------+--
-------------------+-----------+
| j015d4ee1b8a5e7f95df19babb2794395e8 | t_order | 1                     | true   |
2022-08-22 16:37:01 | NULL      |
+---------------------------------------+----------+-----------------------+--------+--
-------------------+-----------+
```

5. View the data migration details.

```
SHOW MIGRATION STATUS 'j015d4ee1b8a5e7f95df19babb2794395e8';
+------+-------------+---------------------------+--------+-------------------------
------+---------------------------+
| item | data_source | status                    | active | inventory_finished_
percentage | incremental_idle_seconds |
+------+-------------+---------------------------+--------+-------------------------
------+---------------------------+
| 0    | ds_0        | EXECUTE_INCREMENTAL_TASK | true   | 100
     | 141                       |
+------+-------------+---------------------------+--------+-------------------------
------+---------------------------+
```

6. Verify data consistency.

```
CHECK MIGRATION 'j015d4ee1b8a5e7f95df19babb2794395e8' BY TYPE (NAME='DATA_MATCH');
+------------+----------------------+----------------------+-----------------------
+-------------------------+
| table_name | source_records_count | target_records_count | records_count_matched
| records_content_matched |
+------------+----------------------+----------------------+-----------------------
+-------------------------+
| t_order    | 6                    | 6                    | true
| true                    |
+------------+----------------------+----------------------+-----------------------
+-------------------------+
```

7. Stop the job.

```
STOP MIGRATION 'j015d4ee1b8a5e7f95df19babb2794395e8';
```

8. Clear the job.

```
CLEAN MIGRATION 'j015d4ee1b8a5e7f95df19babb2794395e8';
```

6.2.5 Observability

Compile source code

Download Apache ShardingSphere from GitHub,Then compile.

```
git clone --depth 1 https://github.com/apache/shardingsphere.git
cd shardingsphere
mvn clean install -Dmaven.javadoc.skip=true -Dcheckstyle.skip=true -Drat.skip=true
-Djacoco.skip=true -DskipITs -DskipTests -Prelease
```

Output directory: shardingsphere-agent/shardingsphere-agent-distribution/target/apache-shardingsphere-${latest.release.version}-shardingsphere-agent-bin.tar.gz

Agent configuration

- Directory structure

  Create agent directory, and unzip agent distribution package to the directory.

```
mkdir agent
tar -zxvf apache-shardingsphere-${latest.release.version}-shardingsphere-agent-
bin.tar.gz -C agent
cd agent
tree
.
```

```
└── apache-shardingsphere-${latest.release.version}-shardingsphere-agent-bin
    ├── LICENSE
    ├── NOTICE
    ├── conf
    │   ├── agent.yaml
    │   └── logback.xml
    ├── plugins
    │   ├── shardingsphere-agent-logging-base-${latest.release.version}.jar
    │   ├── shardingsphere-agent-metrics-prometheus-${latest.release.version}.jar
    │   ├── shardingsphere-agent-tracing-jaeger-${latest.release.version}.jar
    │   ├── shardingsphere-agent-tracing-opentelemetry-${latest.release.version}.
jar
    │   ├── shardingsphere-agent-tracing-opentracing-${latest.release.version}.
jar
    │   └── shardingsphere-agent-tracing-zipkin-${latest.release.version}.jar
    └── shardingsphere-agent.jar
```

- Configuration file

`conf/agent.yaml` is used to manage agent configuration. Built-in plugins include Jaeger, OpenTracing, Zipkin, OpenTelemetry, Logging and Prometheus. When a plugin needs to be enabled, just remove the corresponding name in `ignoredPluginNames`.

```yaml
applicationName: shardingsphere-agent
ignoredPluginNames:
  - Jaeger
  - OpenTracing
  - Zipkin
  - OpenTelemetry
  - Logging
  - Prometheus

plugins:
  Prometheus:
    host:  "localhost"
    port: 9090
    props:
      JVM_INFORMATION_COLLECTOR_ENABLED : "true"
  Jaeger:
    host: "localhost"
    port: 5775
    props:
      SERVICE_NAME: "shardingsphere-agent"
      JAEGER_SAMPLER_TYPE: "const"
      JAEGER_SAMPLER_PARAM: "1"
  Zipkin:
    host: "localhost"
    port: 9411
    props:
```

```yaml
      SERVICE_NAME: "shardingsphere-agent"
      URL_VERSION: "/api/v2/spans"
      SAMPLER_TYPE: "const"
      SAMPLER_PARAM: "1"
  OpenTracing:
    props:
      OPENTRACING_TRACER_CLASS_NAME: "org.apache.skywalking.apm.toolkit.opentracing.SkywalkingTracer"
  OpenTelemetry:
    props:
      otel.resource.attributes: "service.name=shardingsphere-agent"
      otel.traces.exporter: "zipkin"
  Logging:
    props:
      LEVEL: "INFO"
```

- Parameter description:

Name	Description	Value range	Default value
JVM_INFORMATION_COLLECTOR_ENABLED	Start JVM collector	true, false	true
SERVICE_NAME	Tracking service name	Custom	shardingsphere-agent
JAEGER_SAMPLER_TYPE	Jaeger sample rate type	const, probabilistic, ratelimiting, remote	const
JAEGER_SAMPLER_PARAM	Jaeger sample rate parameter	const:0, 1, pr obabilistic:0.0 - 1.0, ratelimiting: > 0, Customize the number of acquisitions per second, remote：need to customize the remote service addres,JAEGER_SAMPLER_MA NAGER_HOST_PORT	1 (const type)
SAMPLER_TYPE	Zipkin sample rate type	const, counting, ratelimiting, boundary	const
SAMPLER_PARAM	Zipkin sampling rate parameter	const:0, 1, counting:0.01 - 1.0, ratelimiting: > 0, boundary:0.0001 - 1.0	1 (const type)
otel.resource.attributes	opentelemetry properties	String key value pair (, split)	service.name=shardingsphere-agent
otel.traces.exporter	Tracing expoter	zipkin, jaeger	zipkin
otel.traces.sampler	Opentelemetry sample rate type	always_on, always_off, traceidratio	always_on
otel.traces.sampler.arg	Opentelemetry sample rate parameter	tr aceidratio：0.0 - 1.0	1.0

Usage in ShardingSphere-Proxy

- Edit the startup script

Configure the absolute path of shardingsphere-agent.jar to the start.sh startup script of shardingsphere proxy.

```
nohup java ${JAVA_OPTS} ${JAVA_MEM_OPTS} \
-javaagent:/xxxxx/agent/shardingsphere-agent.jar \
-classpath ${CLASS_PATH} ${MAIN_CLASS} >> ${STDOUT_FILE} 2>&1 &
```

- Start ShardingSphere-Proxy

```
bin/start.sh
```

After startup, you can find the plugin info in the log of ShardingSphere-Proxy, `Metric` and `Tracing` data can be viewed through the configured monitoring address.

6.3 Common Configuration

This chapter mainly introduces general configuration, including property configuration and built-in algorithm configuration.

6.3.1 Properties Configuration

Background

Apache ShardingSphere provides the way of property configuration to configure system level configuration.

Parameters

N ame	• D a t a T y p e *	*Description*	• D e f a u l t V a l u e *
sql- show (?)	b o o l e a n	Whether show SQL or not in log. Print SQL details can help developers debug easier. The log details include: logic SQL, actual SQL and SQL parse result. Enable this property will log into log topic `ShardingSphere-SQL`, log level is INFO	f a l s e
sq l-si mple (?)	b o o l e a n	Whether show SQL details in simple style	f a l s e
kern el-e xecu tor- size (?)	i n t	The max thread size of worker group to execute SQL. One ShardingSphereDataSource will use a independent thread pool, it does not share thread pool even different data source in same JVM	i n f i n i t e
ma x-co nnec tion s-si ze-p er-q uery (?)	i n t	Max opened connection size for each query	1
chec k-ta ble- meta data -ena bled (?)	b o o l e a n	Whether validate table meta data consistency when application startup or updated	f a l s e
sq l-fe dera tion -ena bled (?)	b o o l e a n	Whether enable SQL federation	f a l s e

Procedure

1. Properties configuration is directly configured in the profile used by ShardingSphere-JDBC. The format is as follows:

```
props:
    sql-show: true
```

Sample

The example of ShardingSphere warehouse contains property configurations of various scenarios. Please refer to: https://github.com/apache/shardingsphere/tree/master/examples/shardingsphere-jdbc-example

6.3.2 Builtin Algorithm

Introduction

Apache ShardingSphere allows developers to implement algorithms via SPI; At the same time, Apache ShardingSphere also provides a couple of builtin algorithms for simplify developers.

Usage

The builtin algorithms are configured by type and props. Type is defined by the algorithm in SPI, and props is used to deliver the customized parameters of the algorithm.

No matter which configuration type is used, the configured algorithm is named and passed to the corresponding rule configuration. This chapter distinguishes and lists all the builtin algorithms of Apache ShardingSphere according to its functions for developers' reference.

Metadata Repository

Background

Apache ShardingSphere provides different metadata persistence methods for different running modes. Users can choose an appropriate way to store metadata while configuring the running mode.

Parameters

File Repository

Type: File

Mode: Standalone

Attributes:

Name	*Type*	*Description*	*Default Value*
path	String	Path for metadata persist	.shardingsphere

ZooKeeper Repository

Type: ZooKeeper

Mode: Cluster

Attributes:

Name	*Type*	*Description*	*Default Value*
retryInte rvalMilliseconds	int	Milliseconds of retry interval	500
maxRetries	int	Max retries of client connection	3
t imeToLiveSeconds	int	Seconds of ephemeral data live	60
operationTim eoutMilliseconds	int	Milliseconds of operation timeout	500
digest	String	Password of login	

Etcd Repository

Type: Etcd

Mode: Cluster

Attributes:

Name	*Type*	*Description*	*Default Value*
timeToLiveSeconds	long	Seconds of ephemeral data live	30
connectionTimeout	long	Seconds of connection timeout	30

Procedure

1. Configure running mode in server.yaml.
2. Configure metadata persistence warehouse type.

Sample

- Standalone mode configuration method.

```yaml
mode:
  type: Standalone
  repository:
    type: File
    props:
       path: ~/user/.shardingsphere
  overwrite: false
```

- Cluster mode.

```yaml
mode:
  type: Cluster
  repository:
    type: zookeeper
    props:
      namespace: governance_ds
      server-lists: localhost:2181
      retryIntervalMilliseconds: 500
      timeToLiveSeconds: 60
      maxRetries: 3
      operationTimeoutMilliseconds: 500
  overwrite: false
```

Sharding Algorithm

Background

ShardingSphere built-in algorithms provide a variety of sharding algorithms, which can be divided into automatic sharding algorithms, standard sharding algorithms, composite sharding algorithms, and hint sharding algorithms, and can meet the needs of most business scenarios of users.

Additionally, considering the complexity of business scenarios, the built-in algorithm also provides a way to customize the sharding algorithm. Users can complete complex sharding logic by writing java code.

Parameters

Auto Sharding Algorithm

Modulo Sharding Algorithm

Type: MOD

Attributes:

Name	*DataType*	*Description*
sharding-count	int	Sharding count

Hash Modulo Sharding Algorithm

Type: HASH_MOD

Attributes:

Name	*DataType*	*Description*
sharding-count	int	Sharding count

Volume Based Range Sharding Algorithm

Type: VOLUME_RANGE

Attributes:

Name	*DataType*	*Description*
range-lower	long	Range lower bound, throw exception if lower than bound
range-upper	long	Range upper bound, throw exception if upper than bound
sharding-volume	long	Sharding volume

Boundary Based Range Sharding Algorithm

Type: BOUNDARY_RANGE

Attributes:

Name	*Data Type*	*Description*
shardi ng-ranges	S tring	Range of sharding border, multiple boundaries separated by commas

Auto Interval Sharding Algorithm

Type: AUTO_INTERVAL

Attributes:

Na me	• DataType*	*Description*
da tet ime -lo wer	String	Shard datetime begin boundary, pattern: yyyy-MM-dd HH:mm:ss
da tet ime -up per	String	Shard datetime end boundary, pattern: yyyy-MM-dd HH:mm:ss
s har din g-s eco nds	long	Max seconds for the data in one shard, allows sharding key timestamp format seconds with time precision, but time precision after seconds is automatically erased

Standard Sharding Algorithm

Apache ShardingSphere built-in standard sharding algorithm are:

Inline Sharding Algorithm

With Groovy expressions, `InlineShardingStrategy` provides single-key support for the sharding operation of = and `IN` in SQL. Simple sharding algorithms can be used through a simple configuration to avoid laborious Java code developments. For example, `t_user_$->{u_id % 8}` means table t_user is divided into 8 tables according to u_id, with table names from `t_user_0` to `t_user_7`. Please refer to Inline Expression for more details.

Type: INLINE

Attributes:

Name	• DataType*	*Description*	*D efa ult Val ue*
algori thm-expression	String	Inline expression sharding algorithm	•
allow-rang e-query-with-i nline-sharding (?)	boolean	Whether range query is allowed. Note: range query will ignore sharding strategy and conduct full routing	fa lse

Interval Sharding Algorithm

This algorithm actively ignores the time zone information of `datetime-pattern`. This means that when `datetime-lower`, `datetime-upper` and the incoming shard key contain time zone information, time zone conversion will not occur due to time zone inconsistencies. When the incoming sharding key is `java.time.Instant`, there is a special case, which will carry the time zone information of the system and convert it into the string format of `datetime-pattern`, and then proceed to the next sharding.

Type: INTERVAL

Attributes:

N ame	• DataType*	*Description*	• DefaultValue*
date time -pat tern	String	Timestamp pattern of sharding value, must can be transformed to Java LocalDateTime. For example: yyyy-MM-dd HH:mm:ss, yyyy-MM-dd or HH:mm:ss etc. But Gy-MM etc. related to `java.time.chrono.JapaneseDate` are not supported	•
da teti me-l ower	String	Datetime sharding lower boundary, pattern is defined `datetime-pattern`	•
da teti me-u pper (?)	String	Datetime sharding upper boundary, pattern is defined `datetime-pattern`	Now
sha rdin g-su ffix -pat tern	String	Suffix pattern of sharding data sources or tables, must can be transformed to Java LocalDateTime, must be consistent with `datetime-interval-unit`. For example: yyyyMM	•
date time -int erva l-am ount (?)	int	Interval of sharding value	1
da teti me-i nter val-unit (?)	String	Unit of sharding value interval, must can be transformed to Java ChronoUnit' s Enum value. For example: MONTHS	DAYS

Complex Sharding Algorithm

Complex Inline Sharding Algorithm

Please refer to Inline Expression for more details.

Type: COMPLEX_INLINE

Name	• DataType*	*Description*	*D efa ult Val ue*
sh arding-columns (?)	String	sharing column names	•
algori thm-expression	String	Inline expression sharding algorithm	•
allow-rang e-query-with-i nline-sharding (?)	boolean	Whether range query is allowed. Note: range query will ignore sharding strategy and conduct full routing	fa lse

Hint Sharding Algorithm

Hint Inline Sharding Algorithm

Please refer to Inline Expression for more details.

Type: COMPLEX_INLINE

Name	*DataType*	*Description*	*Default Value*
algor ithm-expression	String	Inline expression sharding algorithm	${value}

Class Based Sharding Algorithm

Realize custom extension by configuring the sharding strategy type and algorithm class name. `CLASS_BASED` allows additional custom properties to be passed into the algorithm class. The passed properties can be retrieved through the `java.util.Properties` class instance with the property name `props`. Refer to Git' s `org.apache.shardingsphere.example.extension.sharding.algortihm.classbased.fixture.ClassBasedStandardShardingAlgorithmFixture`.

Type: CLASS_BASED

Attributes:

Name	*Data Type*	*Description*
strategy	S tring	Sharding strategy type, support STANDARD, COMPLEX or HINT (case insensitive)
algor ithmClass-Name	S tring	Fully qualified name of sharding algorithm

Procedure

1. When using data sharding, configure the corresponding data sharding algorithm under the shardingAlgorithms attribute.

Sample

```
rules:
- !SHARDING
  tables:
    t_order:
      actualDataNodes: ds_${0..1}.t_order_${0..1}
      tableStrategy:
        standard:
          shardingColumn: order_id
          shardingAlgorithmName: t-order-inline
      keyGenerateStrategy:
        column: order_id
        keyGeneratorName: snowflake
    t_order_item:
      actualDataNodes: ds_${0..1}.t_order_item_${0..1}
      tableStrategy:
        standard:
          shardingColumn: order_id
          shardingAlgorithmName: t_order-item-inline
      keyGenerateStrategy:
        column: order_item_id
        keyGeneratorName: snowflake
    t_account:
      actualDataNodes: ds_${0..1}.t_account_${0..1}
      tableStrategy:
        standard:
          shardingAlgorithmName: t-account-inline
      keyGenerateStrategy:
        column: account_id
        keyGeneratorName: snowflake
  defaultShardingColumn: account_id
  bindingTables:
```

```
  - t_order,t_order_item
broadcastTables:
  - t_address
defaultDatabaseStrategy:
  standard:
    shardingColumn: user_id
    shardingAlgorithmName: database-inline
defaultTableStrategy:
  none:

shardingAlgorithms:
  database-inline:
    type: INLINE
    props:
      algorithm-expression: ds_${user_id % 2}
  t-order-inline:
    type: INLINE
    props:
      algorithm-expression: t_order_${order_id % 2}
  t_order-item-inline:
    type: INLINE
    props:
      algorithm-expression: t_order_item_${order_id % 2}
  t-account-inline:
    type: INLINE
    props:
      algorithm-expression: t_account_${account_id % 2}
keyGenerators:
  snowflake:
    type: SNOWFLAKE
```

Related References

- Core Feature: Data Sharding
- Developer Guide: Data Sharding

Key Generate Algorithm

Background

In traditional database software development, automatic primary key generation is a basic requirement and various databases provide support for this requirement, such as MySQL's self-incrementing keys, Oracle's self-incrementing sequences, etc.

After data sharding, it is a very tricky problem to generate global unique primary keys from different data nodes. Self-incrementing keys between different actual tables within the same logical table generate duplicate primary keys because they are not mutually perceived.

Although collisions can be avoided by constraining the initial value and step size of self-incrementing primary keys, additional O&M rules must to be introduced, making the solution lack completeness and scalability.

There are many third-party solutions that can perfectly solve this problem, such as UUID, which relies on specific algorithms to generate non-duplicate keys, or by introducing primary key generation services.

In order to cater to the requirements of different users in different scenarios, Apache ShardingSphere not only provides built-in distributed primary key generators, such as UUID, SNOWFLAKE, but also abstracts the interface of distributed primary key generators to facilitate users to implement their own customized primary key generators.

Parameters

Snowflake

Type: SNOWFLAKE

Attributes:

Name	• DataType*	*Description*	*Def ault Va lue*
worker-id (?)	long	The unique ID for working machine	0
max -tolerate-time-diff erence-milliseconds (?)	long	The max tolerate time for different server's time difference in milliseconds	10 mill isec onds
m ax-vibration-offset (?)	int	The max upper limit value of vibrate number, range `[0, 4096)`. Notice: To use the generated value of this algorithm as sharding value, it is recommended to configure this property. The algorithm generates key mod `2^n` (`2^n` is usually the sharding amount of tables or databases) in different milliseconds and the result is always `0` or `1`. To prevent the above sharding problem, it is recommended to configure this property, its value is `(2^n)-1`	1

Note: worker-id is optional 1. In standalone mode, support user-defined configuration, if the user does not configure the default value of 0. 2. In cluster mode, it will be automatically generated by the system, and duplicate values will not be generated in the same namespace.

Nano ID

Type:NANOID

Configurable Property:none

UUID

Type: UUID

Attributes: None

CosId

Type: COSID

Attributes：

• Name*	• DataType*	*Description*	*D ef au lt Va lu e*
id-name	String	ID generator name	` _ _s ha re __ `
as-string	bool	Whether to generate a string type ID: Convert `long` type ID to Base-62 `String` type (`Long.MAX_VALUE` maximum string length is 11 digits), and ensure the ordering of string IDs	` f al se `

CosId-Snowflake

Type: COSID_SNOWFLAKE

Attributes：

• Name*	• DataType*	*Description*	*Default Value*
epoch	String	EPOCH of Snowflake ID Algorithm	` 1 47 79 29 60 00 00 `
as-string	bool	Whether to generate a string type ID: Convert `long` type ID to Base-62 `String` type (`Long.MAX_VALUE` maximum string length is 11 digits), and ensure the ordering of string IDs	` false `

Procedure

1. Policy of distributed primary key configurations is for columns when configuring data sharding rules.

Sample

- Snowflake Algorithms

```
keyGenerators:
  snowflake:
    type: SNOWFLAKE
```

- NanoID

```
keyGenerators:
  nanoid:
    type: NANOID
```

- UUID

```
keyGenerators:
  nanoid:
    type: UUID
```

Load Balance Algorithm

Background

ShardingSphere built-in provides a variety of load balancer algorithms, including polling algorithm, random access algorithm and weight access algorithm, which can meet users' needs in most business scenarios.

Moreover, considering the complexity of the business scenario, the built-in algorithm also provides an extension mode. Users can implement the load balancer algorithm they need based on SPI interface.

Parameters

Type	*Describe*	*Limitations*
ROUND_ROBIN	Within the transaction, read query are routed to the primary, and outside the transaction, the round-robin strategy is used to route to the replica	
RANDOM	Within the transaction, read query are routed to the primary, and outside the transaction, the random strategy is used to route to the replica	
WEIGHT	Within the transaction, read query are routed to the primary, and outside the transaction, the weight strategy is used to route to the replica	Attributes need to be configured, attribute name: ${replica-name}, data type: double, attribute name uses the name of the replica, and the parameter fills in the weight value corresponding to the replica. Weight parameter range min > 0, total <= Double.MAX_VALUE.
TRANSACTION_RANDOM	Display/non-display open transaction, read query are routed to multiple replicas using random strategy	
TRANSACTION_ROUND_ROBIN	Display/non-display open transaction, read query are routed to multiple replicas using round-robin strategy	
TRANSACTION_WEIGHT	Display/non-display open transaction, read query are routed to multiple replicas using weight strategy	Attributes need to be configured, attribute name: ${replica-name}, data type: double, attribute name uses the name of the replica, and the parameter fills in the weight value corresponding to the replica. Weight parameter range min > 0, total <= Double.MAX_VALUE.
FIXED_REPLICA_RANDOM	Open transaction displayed, and the read query is routed to a fixed replica using random strategy; otherwise, each read traffic is routed to a different replica using random strategy	
FIXED_REPLICA_ROUND_ROBIN	Open transaction displayed, and the read query is routed to a fixed replica using round-robin strategy; otherwise, each read traffic is routed to a different replica using round-robin strategy	
FIXED_REPLICA_WEIGHT	Open transaction displayed, and the read query is routed to a fixed replica using weight strategy; otherwise, each read traffic is routed to a different replica using weight strat-	Attributes need to be configured, attribute name: ${replica-name}, data type: double, attribute name uses the name of the replica, and the parameter fills in the weight value corresponding to the replica. Weight parameter range min > 0, total <= Dou-

Procedure

1. Configure a load balancer algorithm for the loadBalancers attribute to use read/write splitting.

Sample

```yaml
rules:
- !READWRITE_SPLITTING
  dataSources:
    readwrite_ds:
      staticStrategy:
        writeDataSourceName: write_ds
        readDataSourceNames:
          - read_ds_0
          - read_ds_1
      loadBalancerName: random
  loadBalancers:
    random:
      type: RANDOM
```

Related References

- Core Feature: Read/Write Splitting
- Developer Guide: Read/Write Splitting

Encryption Algorithm

Background

Encryption algorithms are the algorithms used by the encryption features of Apache ShardingSphere. A variety of algorithms are built-in to make it easy for users to fully leverage the feature.

Parameters

MD5 Encrypt Algorithm

Type: MD5

Attributes: None

AES Encrypt Algorithm

Type: AES

Attributes:

Name	*DataType*	*Description*
aes-key-value	String	AES KEY

RC4 Encrypt Algorithm

Type: RC4

Attributes:

Name	*DataType*	*Description*
rc4-key-value	String	RC4 KEY

SM3 Encrypt Algorithm

Type: SM3

Attributes:

Name	*DataType*	*Description*
sm3-salt	String	SM3 SALT (should be blank or 8 bytes long)

SM4 Encrypt Algorithm

Type: SM4

Attributes:

Name	*DataType*	*Description*
sm4-key	String	SM4 KEY (should be 16 bytes)
sm4-mode	String	SM4 MODE (should be CBC or ECB)
sm4-iv	String	SM4 IV (should be specified on CBC, 16 bytes long)
sm4-padding	String	SM4 PADDING (should be PKCS5Padding or PKCS7Padding, NoPadding excepted)

Operating Procedures

1. Configure encryptors in an encryption rule.
2. Use relevant algorithm types in encryptors.

Configuration Examples

```yaml
rules:
- !ENCRYPT
  tables:
    t_user:
      columns:
        username:
          plainColumn: username_plain
          cipherColumn: username
          encryptorName: name-encryptor
  encryptors:
    name-encryptor:
      type: AES
      props:
        aes-key-value: 123456abc
```

Related References

- Core Feature: Data Encrypt
- Developer Guide: Data Encrypt

Shadow Algorithm

Background

The shadow DB feature carries out shadow measurement to SQL statements executed. Shadow measurement supports two types of algorithms, and users can choose one or a combination of them based on actual business needs.

Parameters

Column-based shadow algorithm

Column value matching shadow algorithm

Type： VALUE_MATCH

Attribute Name	*Data Type*	*Description*
column	String	shadow column
operation	String	SQL operation type (INSERT, UPDATE, DELETE, SELECT)
value	String	value matched by shadow column

Column-based Regex matching algorithm

Type： REGEX_MATCH

Attribute Name	*Data Type*	*Description*
column	String	match a column
operation	String	SQL operation type （INSERT, UPDATE, DELETE, SELECT）
regex	String	shadow column matching Regex

Hint-based shadow algorithm

Simple Hint matching shadow algorithm

Type： SIMPLE_HINT

Attribute Name	*Data Type*	*Description*
foo	String	bar

Configuration sample

- Java API

```java
public final class ShadowConfiguration {
    // ...

    private AlgorithmConfiguration createShadowAlgorithmConfiguration() {
        Properties userIdInsertProps = new Properties();
        userIdInsertProps.setProperty("operation", "insert");
        userIdInsertProps.setProperty("column", "user_id");
        userIdInsertProps.setProperty("value", "1");
        return new AlgorithmConfiguration("VALUE_MATCH", userIdInsertProps));
    }

    // ...
}
```

- YAML:

```yaml
shadowAlgorithms:
  user-id-insert-algorithm:
    type: VALUE_MATCH
    props:
      column: user_id
      operation: insert
      value: 1
```

- Spring Boot Starter:

```properties
spring.shardingsphere.rules.shadow.shadow-algorithms.user-id-insert-algorithm.
type=VALUE_MATCH
spring.shardingsphere.rules.shadow.shadow-algorithms.user-id-insert-algorithm.
props.operation=insert
spring.shardingsphere.rules.shadow.shadow-algorithms.user-id-insert-algorithm.
props.column=user_id
spring.shardingsphere.rules.shadow.shadow-algorithms.user-id-insert-algorithm.
props.value=1
```

- Spring Namespace:

```xml
<shadow:shadow-algorithm id="user-id-insert-algorithm" type="VALUE_MATCH">
    <props>
        <prop key="operation">insert</prop>
        <prop key="column">user_id</prop>
        <prop key="value">1</prop>
    </props>
</shadow:shadow-algorithm>
```

SQL Translator

Native SQL translator

Type: NATIVE

Attributes:

None

Default SQL translator, does not implement yet.

JooQ SQL translator

Type: JOOQ

Attributes:

None

Because of it need JooQ dependency, ShardingSphere does not include the module, please use below XML to import it by Maven.

```
<dependency>
    <groupId>org.apache.shardingsphere</groupId>
    <artifactId>shardingsphere-sql-translator-jooq-provider</artifactId>
    <version>${project.version}</version>
</dependency>
```

6.4 Error Code

This chapter lists error codes of Apache ShardingSphere. They include SQL error codes and server error codes.

6.4.1 SQL Error Code

SQL error codes provide by standard `SQL State`, `Vendor Code` and `Reason`, which return to client when SQL execute error.

SQL State	Vendor Code	Reason
01000	10000	Circuit break open, the request has been ignored
08000	10001	The URL \`%s\` is not recognized, please refer to the pattern \`%s\`
42000	10002	Can not support 3-tier structure for actual data node \`%s\` with JDBC \`%s\`
42000	10003	Unsupported SQL node conversion for SQL statement \`%s\`
HY004	10004	Unsupported conversion data type \`%s\` for value \`%s\`
HY004	10100	Can not register driver, reason is: %s
34000	10200	Can not get cursor name from fetch statement
42000	11000	You have an error in your SQL syntax: %s
42000	11001	configuration error
42000	11002	Resource does not exist
42000	11003	Rule does not exist
HY000	11004	File access failed, reason is: %s
42000	11200	Can not support database \`%s\` in SQL translation
42000	11201	Translation error, SQL is: %s
25000	11320	Switch transaction type failed, please terminate the current transaction
25000	11321	JDBC does not support operations across multiple logical databases in transaction

Table 1 – continued from previous page

SQL State	Vendor Code	Reason
25000	11322	Failed to create \`%s\` XA data source
42S02	11400	Can not get traffic execution unit
42000	12000	Unsupported command: %s
44000	13000	SQL check failed, error message: %s
HY000	14000	The table \`%s\` of schema \`%s\` is locked
HY000	14001	The table \`%s\` of schema \`%s\` lock wait timeout of %s ms exceeded
HY000	14010	Can not find \`%s\` file for datetime initialize
HY000	14011	Load datetime from database failed, reason: %s
HY000	15000	Work ID assigned failed, which can not exceed 1024
HY000	16000	Can not find pipeline job \`%s\`
HY000	16001	Failed to get DDL for table \`%s\`
42S02	17000	Single table \`%s\` does not exist
42S02	17001	Schema \`%s\` does not exist
0A000	17010	DROP TABLE …CASCADE is not supported
HY000	20000	Sharding algorithm class \`%s\` should be implement \`%s\`
44000	20001	Can not get uniformed table structure for logic table \`%s\`, it has different meta data of actu
42S02	20002	Can not get route result, please check your sharding rule configuration
44000	20003	Can not find table rule with logic tables \`%s\`
44000	20010	Sharding value can't be null in insert statement
HY004	20011	Found different types for sharding value \`%s\`
44000	20012	Can not update sharding value for table \`%s\`
0A000	20013	Can not support operation \`%s\` with sharding table \`%s\`
0A000	20014	Can not support DML operation with multiple tables \`%s\`
0A000	20015	The CREATE VIEW statement contains unsupported query statement
42000	20016	%s …LIMIT can not support route to multiple data nodes
44000	20017	PREPARE statement can not support sharding tables route to same data sources
42000	20018	INSERT INTO …SELECT can not support applying key generator with absent generate key
44000	20019	The table inserted and the table selected must be the same or bind tables
44000	20020	Inline sharding algorithms expression \`%s\` and sharding column \`%s\` do not match
42S01	20030	Index \`%s\` already exists
42S02	20031	Index \`%s\` does not exist
44000	20032	Actual tables \`%s\` are in use
42S01	20033	View name has to bind to %s tables
HY000	20050	Routed target \`%s\` does not exist, available targets are \`%s\`
HY000	20051	\`%s %s\` can not route correctly for %s \`%s\`
42S02	20052	Can not find data source in sharding rule, invalid actual data node \`%s\`
HY004	24000	Encrypt algorithm \`%s\` initialize failed, reason is: %s
HY004	24001	The SQL clause \`%s\` is unsupported in encrypt rule
44000	24002	Altered column \`%s\` must use same encrypt algorithm with previous column \`%s\` in table
42000	24003	Insert value of index \`%s\` can not support for encrypt
44000	24004	Can not find logic encrypt column by \`%s\`

Table 1 – continued from previous page

SQL State	Vendor Code	Reason
HY004	25000	Shadow column \`%s\` of table \`%s\` does not support \`%s\` type
42000	25003	Insert value of index \`%s\` can not support for shadow
42000	30000	Unknown exception: %s

6.4.2 Server Error Code

Unique codes provided when server exception occur, which printed by Proxy backend or JDBC startup logs.

TODO

7 Dev Manual

Apache ShardingSphere provides dozens of SPI based extensions. it is very convenient to customize the functions for developers.

This chapter lists all SPI extensions of Apache ShardingSphere. If there is no special requirement, users can use the built-in implementation provided by Apache ShardingSphere; advanced users can refer to the interfaces for customized implementation.

Apache ShardingSphere community welcomes developers to feed back their implementations to the open-source community, so that more users can benefit from it.

7.1 Mode

7.1.1 StandalonePersistRepository

Fully-qualified class name

`org.apache.shardingsphere.mode.repository.standalone. StandalonePersistRepository <https://github.com/apache/shardingsphere/blob/master/shardingsphere-mode/shardingsphere-mode-type/shardingsphere-standalone-mode/shardingsphere-standalone-mode-repository/shardingsphere-standalone-mode-repository-api/src/main/java/org/apache/shardingsphere/mode/repository/standalone/StandalonePersistRepository.java>`__

Definition

Standalone mode configuration information persistence definition

Implementation classes

Configuration Type	*Description*	*Fully-qualified class name*
H2	H2-based persistence	`org.apache.sharding sphere.mode.repository.sta ndalone.h2. H2Repository <https://github.com/apache /shardingsphere/blob/maste r/shardingsphere-mode/shar dingsphere-mode-type/shard ingsphere-standalone-mode/ shardingsphere-standalone- mode-repository/shardingsp here-standalone-mode-repos itory-provider/shardingsph ere-standalone-mode-reposi tory-jdbc-h2/src/main/java /org/apache/shardingsphere /mode/repository/standalon e/h2/H2Repository.java>`__

7.1.2 ClusterPersistRepository

Fully-qualified class name

`org.apache.shardingsphere.mode.repository.cluster.ClusterPersistRepository <https://github.com/apache/shardingsphere/blob/master/shardingsphere-mode/shardingsphere-mode-type/shardingsphere-cluster-mode/shardingsphere-cluster-mode-repository/shardingsphere-cluster-mode-repository-api/src/main/java/org/apache/shardingsphere/mode/repository/cluster/ClusterPersistRepository.java>`__

Definition

Cluster mode configuration information persistence definition

Implementation classes

• Configuration Type*	*Description*	*Fully-qualified class name*
ZooKeeper	ZooKeeper-based persistence	`org.apache.shardingsphere.mode.repository.cluster.zookeeper.CuratorZookeeperRepository <https://github.com/apache/shardingsphere/blob/master/shardingsphere-mode/shardingsphere-mode-type/shardingsphere-cluster-mode/shardingsphere-cluster-mode-repository/shardingsphere-cluster-mode-repository-provider/shardingsphere-cluster-mode-repository-zookeeper-curator/src/main/java/org/apache/shardingsphere/mode/repository/cluster/zookeeper/CuratorZookeeperRepository.java>`__
etcd	Etcd-based persistence	`org.apache.shardingsphere.mode.repository.cluster.etcd.EtcdRepository <https://github.com/apache/shardingsphere/blob/master/shardingsphere-mode/shardingsphere-mode-type/shardingsphere-cluster-mode/shardingsphere-cluster-mode-repository/shardingsphere-cluster-mode-repository-provider/shardingsphere-cluster-mode-repository-etcd/src/main/java/org/apache/shardingsphere/mode/repository/cluster/etcd/EtcdRepository.java>`__

7.1.3 GovernanceWatcher

Fully-qualified class name

`org.apache.shardingsphere.mode.manager.cluster.coordinator.registry. GovernanceWatcher <https://github.com/apache/shardingsphere/blob/master/shardingsphere-mode/shardingsphere-mode-type/shardingsphere-cluster-mode/shardingsphere-cluster-mode-core/src/main/java/org/apache/shardingsphere/mode/manager/cluster/coordinator/registry/GovernanceWatcher.java>`__

Definition

Governance listener definition

Implementation classes

Configuration Type	*Description*	*Full y-qualified class name*
Types: ADDED, UPDATED, DELETED; WatchingKeys: /nodes/compute_nodes	Compute node state change listener	`` *org.apache .shardingsp here.mode.m anager.clus ter.coordin ator.regist ry.status.c ompute.watc her.Compute NodeStateCh angedWatche r`* <https: //github.co m/apache/sh ardingspher e/blob/mast er/sharding sphere-mode /shardingsp here-mode-t ype/shardin gsphere-clu ster-mode/s hard- ingsphe re-cluster- mode- core/s rc/main/jav a/org/apach e/shardings phere/mode/ man- ager/clu ster/coordi nator/regis try/status/ compute/wat cher/Comput eNodeStateC hangedWatch er.java>`__
Types: ADDED, DELETED; WatchingKeys: /lock/database/locks	Database lock state change listener	`org. apache.shar ding- sphere. mode.manage r.cluster.c oordi- nator. lock.databa se.watcher. Database- Loc kChangedWat cher <htt ps://github .com/apache /shardingsp here/blob/m aster/shard ingsphere-m ode/shardin gsphere-mod e- type/shar dingsphere- cluster- mod e/shardings phere-clust er-mode-cor e/src/main/ java/org/ap ache/shardi ng- sphere/mo de/manager/ clus- ter/coo rdinator/lo ck/database /watcher/Da tabaseLockC hangedWatch er.java>`__
Types: ADDED, DELETED; WatchingKeys: /lock/distributed/locks	Distributed lock change listener	`org.a pache.shard ing- sphere.m ode.manager . cluster.co ordinator.l ock.distrib uted.watche r.Distribut edLockChang edWatcher <https://g ithub.com/a pache/shard ingsphere/b lob/master/ shardingsph ere-mode/sh

7.2 Configuration

7.2.1 RuleBuilder

Fully-qualified class name

`org.apache.shardingsphere.infra.rule.builder.RuleBuilder <https://github.com/apache/shardingsphere/blob/master/shardingsphere-infra/shardingsphere-infra-common/src/main/java/org/apache/shardingsphere/infra/rule/builder/RuleBuilder.java>`__

Definition

Used to convert user configurations into rule objects

Implementation classes

Configuration Type	*Description*	*F ully-qu alified class name*
AuthorityR uleConfiguration	Used to convert authority user configuration into authority rule objects	`o rg.apac he.shar dingsph ere.aut hority. rule.bu ilder.A uthorit yRuleBu ilder <https ://gith ub.com/ apache/ shardin gsphere /blob/m aster/s hard- ing sphere- kernel/ shardin gsphere -author ity/sha rdingsp here-au thority -core/s rc/main /java/o rg/apac he/shar dingsph ere/aut hority/ rule/bu ilder/A uthorit yRuleBu ilder.j ava>`__
SQLParserR uleConfiguration	Used to convert SQL parser user configuration into SQL parser rule objects	``*org .apache .shardi ngspher e.parse r.rule. builder .SQLPar serRule Builder* ` <htt ps://gi thub.co m/apach e/shard ingsphe re/blob /master /shardi ngspher e-kerne l/shard ingsphe re-pars er/shar dingsph ere-par ser-cor e/src/m ain/jav a/org/a pache/s hard- ing sphere/ parser/ rule/bu ilder/S QLParse rRuleBu ilder.j ava>`__
TransactionR uleConfiguration	Used to convert transaction user configuration into transaction rule objects	`` *org.ap ache.sh ardings phere.t ransact ion.rul e.build er.Tran saction RuleBui lder* ` <https: //githu b.com/a pache/s hard- ing sphere/ blob/ma ster/sh ardings phere-k ernel/s hard- ing sphere- transac tion/sh ardings phere-t ransact ion-cor e/src/m ain/jav a/org/a pache/s harding sphere/ transac tion/ru le/buil der/Tra nsactio nRuleBu ilder.j ava>`__
SingleTableR uleConfiguration	Used to convert single-table user configuration into a single-table rule objects	`o rg.apac he.shar dingsph ere.sin gletabl e.rule. builder .Single TableRu leBuild er <h ttps:// github. com/apa che/sha rdingsp here/bl ob/mast er/shar dingsph ere-ker nel/sha rdingsp here-si ngle-ta ble/sha rdingsp here-si ngle-ta ble-cor

7.2.2 YamlRuleConfigurationSwapper

Fully-qualified class name

`org.apache.shardingsphere.infra.yaml.config.swapper.YamlRuleConfigurationSwapper <https://github.com/apache/shardingsphere/blob/master/shardingsphere-infra/shardingsphere-infra-common/src/main/java/org/apache/shardingsphere/infra/yaml/config/swapper/YamlRuleConfigurationSwapper.java>`__

Definition

Used to convert YAML configuration to standard user configuration

Implementation classes

Con figur ation Type	*Description*	*Fully- qualified class name*
AUTH ORITY	Used to convert the YAML configuration of authority rules into standard configuration of authority rules	``*org.a pache.sha rdingsphe re.author ity.yaml. swapper.Y amlAuthor ityRuleCo nfigurati onSwapper* ` <https ://github .com/apac he/shardi ngsphere/ blob/mast er/shardi ngsphere-kernel/sh ardingsph ere-autho rity/shar dingspher e-authori ty-core/s rc/main/j ava/org/a pache/sha rdingsphe re/author ity/yaml/ swapper/Y amlAuthor ityRuleCo nfigurati onSwapper .java>`__
SQL_P ARSER	Used to convert the YAML configuration of the SQL parser into the standard configuration of the SQL parser	``*or g.apache. shardings phere.par ser.yaml. swapper.Y amlSQLPar serRuleCo nfigurati onSwapper* ` <https ://github .com/apac he/shardi ngsphere/ blob/mast er/shardi ngsphere-kernel/sh ardingsph ere-parse r/shardin gsphere-p arser-cor e/src/mai n/java/or g/apache/ shardings phere/par ser/yaml/ swapper/Y amlSQLPar serRuleCo nfigurati onSwapper .java>`__
T RANSA CTION	Used to convert the YAML configuration of the transaction into the standard configuration of the transaction	`` *org.apac he.shardi ngsphere. transacti on.yaml.s wapper.Ya mlTransac tionRuleC onfigurat ionSwappe r*` <http s://githu b.com/apa che/shard ingsphere /blob/mas ter/shard ingsphere -kernel/s hardingsp here-tran saction/s hardingsp here-tran saction-c ore/src/m ain/java/ org/apach e/shardin gsphere/t ransactio n/yaml/sw apper/Yam lTransact ionRuleCo nfigurati onSwapper .java>`__
S INGLE	Used to convert the YAML configuration of the single table into the standard configuration of the single table	`` `org.ap ache.shar ding-spher e.singlet able. yaml .config.s wapper. Ya mlSingleT ableRuleC onfigurat ionSwappe r `` <http s://githu b.com/apa

7.2.3 ShardingSphereYamlConstruct

Fully-qualified class name

`org.apache.shardingsphere.infra.yaml.engine.constructor.ShardingSphereYamlConstruct <https://github.com/apache/shardingsphere/blob/master/shardingsphere-infra/shardingsphere-infra-common/src/main/java/org/apache/shardingsphere/infra/yaml/engine/constructor/ShardingSphereYamlConstruct.java>`__

Definition

Used to convert custom objects and YAML to and from each other

Implementation classes

Configuration Type	*Description*	*Fully-qualified class name*
YamlNoneShardingStrategyConfiguration	Used to convert non-sharding policy objects and YAML to and from each other	` org.apache. shardingsph ere.shardin g.yaml.engi ne. construc t.NoneShard ingStrategy Configurati onYaml-Const ruct <htt ps://github .com/apache /shardingsp here/blob/m aster/shard ingsphere-f eatures/sha rdingsphere -sharding/s hardingsphe re-sharding -core/src/m ain/java/or g/apache/sh ardingspher e/sharding/ yaml/engine /construct/ NoneShardin gStrategyCo nfiguration YamlCon-stru ct.java>`__

7.3 Kernel

7.3.1 SQLRouter

Fully-qualified class name

`org.apache.shardingsphere.infra.route.SQLRouter <https://github.com/apache/shardingsphere/blob/master/shardingsphere-infra/shardingsphere-infra-route/src/main/java/org/apache/shardingsphere/infra/route/SQLRouter.java>`__

Definition

Used to process routing results

Implementation classes

Configuration type	*Description*	• Fully-qualified class name*
Single TableRule.class	Used to process single-table routing results	`org.ap ache.shardingsp here.singletabl e. route.SingleT ableSQL-Router <https://githu b.com/apache/sh arding-sphere/bl ob/master/shard ingsphere-kerne l/shardingspher e-single-table/ shardingsphere- single-table-co re/src/main/jav a/org/apache/sh arding-sphere/si ngletable/route /SingleTableSQL Router.java>`__
Sha rdingRule.class	Used to process sharding routing results	`or g.apache.shardi ng-sphere.shardi ng.route. engine .ShardingSQLRou ter <https:// github.com/apac he/shardingsphe re/blob/master/ shardingsphere- features/shardi ngsphere-shardi ng/shardingsphe re-sharding-cor e/src/main/java /org/apache/sha rd-ingsphere/sha rd-ing/route/eng ine/ShardingSQL Router.java>`__
ReadwriteSpli ttingRule.class	Used to process read-write splitting routing results	`org.a pache.shardings phere.readwrite splitting.route . ReadwriteSplit tingSQL-Router <https://githu b.com/apache/sh arding-sphere/bl ob/master/shard ingsphere-featu res/shardingsph ere-readwrite-s plitting/shardi ngsphere-readwr ite-splitting-c ore/src/main/ja va/org/apache/s harding-sphere/r eadwritesplitti ng/route/Readwr iteSplit-tingSQL Router.java>`

7.3.2 SQLRewriteContextDecorator

Fully-qualified class name

`org.apache.shardingsphere.infra.rewrite.context.SQLRewriteContextDecorator <https://github.com/apache/shardingsphere/blob/master/shardingsphere-infra/shardingsphere-infra-rewrite/src/main/java/org/apache/shardingsphere/infra/rewrite/context/SQLRewriteContextDecorator.java>`__

Definition

Used to handle SQL rewrite results

Implementation classes

Configuration type	*Description*	*Fully-qualified class name*
ShardingRule.class	Used to process sharding SQL rewrite results	`org.apache.shardingsphere.sharding.rewrite.context.ShardingSQLRewriteContextDecorator <https://github.com/apache/shardingsphere/blob/master/shardingsphere-features/shardingsphere-sharding/shardingsphere-sharding-core/src/main/java/org/apache/shardingsphere/sharding/rewrite/context/ShardingSQLRewriteContextDecorator.java>`__
EncryptRule.class	Used to process encryption SQL rewrite results	`org.apache.shardingsphere.encrypt.rewrite.context.EncryptSQLRewriteContextDecorator <https://github.com/apache/shardingsphere/blob/master/shardingsphere-features/shardingsphere-encrypt/shardingsphere-encrypt-core/src/main/java/org/apache/shardingsphere/encrypt/rewrite/context/EncryptSQLRewriteContextDecorator.java>`__

7.3.3 SQLExecutionHook

Fully-qualified class name

`org.apache.shardingsphere.infra.executor.sql.hook.SQLExecutionHook <https://github.com/apache/shardingsphere/blob/master/shardingsphere-infra/shardingsphere-infra-executor/src/main/java/org/apache/shardingsphere/infra/executor/sql/hook/SQLExecutionHook.java>`__

Definition

SQL execution process listener

Implementation classes

Configuration type	*Description*	*Fully-qualified class name*
Empty	Transaction hook of SQL execution	`org.apache.shardingsphere.transaction.base.seata.at.TransactionalSQLExecutionHook <https://github.com/apache/shardingsphere/blob/master/shardingsphere-kernel/shardingsphere-transaction/shardingsphere-transaction-type/shardingsphere-transaction-base/shardingsphere-transaction-base-seata-at/src/main/java/org/apache/shardingsphere/transaction/base/seata/at/TransactionalSQLExecutionHook.java>`__

7.3.4 ResultProcessEngine

Fully-qualified class name

`org.apache.shardingsphere.infra.merge.engine.ResultProcessEngine <https://github.com/apache/shardingsphere/blob/master/shardingsphere-infra/shardingsphere-infra-merge/src/main/java/org/apache/shardingsphere/infra/merge/engine/ResultProcessEngine.java>`__

Definition

Used to process result sets

Implementation classes

Configuration type	*Description*	*Fully-qualified class name*
ShardingRule.class	Used to handle sharding result set merge	`org .apache.shardingsp here.sharding.merg e. ShardingResultMe rgerEngine <http s://github.com/apa che/shardingsphere /blob/master/shard ingsphere-features /shardingsphere-sh arding/shardingsph ere-sharding-core/ src/main/java/org/ apache/shardingsph ere/sharding/merge /ShardingResultMer gerEngine.java>`__
EncryptRule.class	Used to handle encrypted result set overrides	`org .apache.shardingsp here.encrypt.merge . EncryptResultDeco ratorEngine <htt ps://github.com/ap ache/shardingspher e/blob/master/shar dingsphere-feature s/shardingsphere-e ncrypt/shardingsph ere-encrypt-core/s rc/main/java/org/a pache/shardingsphe re/encrypt/merge/E ncryptResultDecora torEngine.java>`__

7.4 DataSource

7.4.1 DatabaseType

Fully-qualified class name

`org.apache.shardingsphere.infra.database.type.DatabaseType <https://github.com/apache/shardingsphere/blob/master/shardingsphere-infra/shardingsphere-infra-common/src/main/java/org/apache/shardingsphere/infra/database/type/DatabaseType.java>`__

Definition

Supported database types definition

Implementation classes

• Configuration Type*	*Description*	*Fully-qualified class name*
SQL92	SQL92 database type	`org.a pache. shardingspher e.infra. database.ty pe.dialect. SQL92Dat abaseType <https: //github.com/apache /shardingsphere/blo b/master/shardingsp here- infra/sharding sphere-infra- common /src/main/java/org/ apache/shardingsphe re/infra/database/t ype/dialect/SQL92Da tabase- Type.java>`__
MySQL	MySQL database	`org.a pache. shardingspher e.infra. database.ty pe.dialect. MySQLDat abaseType <https: //github.com/apache /shardingsphere/blo b/master/shardingsp here- infra/sharding sphere-infra- common /src/main/java/org/ apache/shardingsphe re/infra/database/t ype/dialect/MySQLDa tabase- Type.java>`__
MariaDB	MariaDB database	`org.apach e. shardingsphere.in fra. database.type.d ialect. MariaDBDatab aseType <https:// github.com/apache/s hardingsphere/blob/ mas- ter/shardingsphe re- infra/shardingsp here-infra- common/s rc/main/java/org/ap ache/shardingsphere /infra/database/typ e/dialect/MariaDBDa tabase- Type.java>`__
PostgreSQL	PostgreSQL database	`org.apache.shar dingsphere.infra.da tabase.type.dialect .PostgreSQLDatabase Type <h t t p s : / / g i t hub.com/apache/shar

7.4.2 DialectSchemaMetaDataLoader

Fully-qualified class name

`org.apache.shardingsphere.infra.metadata.database.schema.loader.spi.
DialectSchemaMetaDataLoader <https://github.com/apache/shardingsphere/blob/master/shardingsphere-infra/shardingsphere-infra-common/src/main/java/org/apache/shardingsphere/infra/metadata/database/schema/loader/spi/DialectSchemaMetaDataLoader.java>`__

Definition

Use SQL dialect to load meta data rapidly

Implementation classes

• Configuration Type*	*Description*	*Fully-qualified class name*
MySQL	Use MySQL dialect to load meta data	` org.apache.sharding sphere.infra.metada ta.database.schema. loader.dialect.MySQ LSchemaMetaDat- aLoad er <https://githu b.com/apache/shardi ngsphere/blob/maste r/shardingsphere-in fra/shardingsphere- infra-common/src/ma in/java/org/apache/ shardingsphere/infr a/metadata/database /schema/loader/dial ect/MySQLSchemaMeta Dat- aLoader.java>`__
Oracle	Use Oracle dialect to load meta data	`or g.apache.shardingsp here.infra.metadata .database.schema. lo ader.dialect. Oracle SchemaMetaDat- aLoade r <https://github .com/apache/shardin gsphere/blob/master /shardingsphere-inf ra/shardingsphere-i nfra-common/src/mai n/java/org/apache/s hardingsphere/infra /metadata/database/ schema/loader/diale ct/OracleSchemaMeta Dat- aLoader.java>`__
PostgreSQL	Use PostgreSQL dialect to load meta data	`org.apache . shardingsphere.inf ra.metadata.databas e.schema.loader.dia lect.PostgreSQLSche maMetaDataLoader <https://github.com /apache/shardingsph ere/blob/master/sha rdingsphere-infra/s hardingsphere-infra -

7.4.3 DataSourcePoolMetaData

Fully-qualified class name

`org.apache.shardingsphere.infra.datasource.pool.metadata.DataSourcePoolMetaData <https://github.com/apache/shardingsphere/blob/master/shardingsphere-infra/shardingsphere-infra-common/src/main/java/org/apache/shardingsphere/infra/datasource/pool/metadata/DataSourcePoolMetaData.java>`__

Definition

Data source connection pool metadata

Implementation classes

Configuration Type	*Description*	*Fully -qualified class name*
or g.apache.commons.dbcp.BasicDataSource, org.ap ache.tomcat.dbcp.dbcp2.BasicDataSource	DBCP data source pool meta data	`org.a pache.shar dingsphere .infra.dat asource.po ol.metadat a.type.dbc p.DBCPData SourcePool MetaData <https:// github.com /apache/sh ardingsphe re/blob/ma ster/shard ingsphere- infra/shar dingsphere -infra-com mon/src/ma in/java/or g/apache/s hardingsph ere/infra/ datasource /pool/meta data/type/ dbcp/DBCPD ataSourceP oolMetaDat a.java>`__
com.zaxxer.hikari.HikariDataSource	Hikari data source pool meta data	`org .apache.sh ardingsphe re.infra.d atasource. pool.metad ata.type.h ikari.Hika riDataSour cePoolMeta Data <ht tps://gith ub.com/apa che/shardi ngsphere/b lob/master /shardings phere-infr a/sharding sphere-inf ra-common/ src/main/j ava/org/ap ache/shard ingsphere/ infra/data source/poo l/metadata /type/hika ri/HikariD ataSourceP oolMetaDat a.java>`__
com .mchange.v2.c3p0.ComboPooledDataSource	C3P0 data source pool meta data	`org.a pache.shar dingsphere .infra.dat asource.po ol.metadat a.type.c3p 0.C3P0Data SourcePool MetaData <https:// github.com /apache/sh ardingsphe re/blob/ma ster/shard ingsphere- infra/shar dingsphere -infra-com mon/src/ma in/java/or g/apache/s hardingsph ere/infra/ datasource /pool/meta data/type/ c3p0/C3P0D ataSourceP oolMetaDat a.java>`__

7.4.4 DataSourcePoolActiveDetector

Fully-qualified class name

`org.apache.shardingsphere.infra.datasource.pool.destroyer.detector.DataSourcePoolActiveDetector <https://github.com/apache/shardingsphere/blob/master/shardingsphere-infra/shardingsphere-infra-common/src/main/java/org/apache/shardingsphere/infra/datasource/pool/destroyer/detector/DataSourcePoolActiveDetector.java>`__

Definition

Data source connection pool active detector

Implementation classes

Configuration Type	*Description*	*Fully-qualified class name*
Default	Default data source pool active detector	`org.apa che.shardingsphe re.infra.datasou rce.pool.destroy er.detector.type .DefaultDataSour cePoolActiveDete ctor <https:// github.com/apach e/shardingsphere /blob/master/sha rdingsphere-infr a/shardingsphere -infra-common/sr c/main/java/org/ apache/shardings phere/infra/data source/pool/dest royer/detector/t ype/DefaultDataS ourcePoolActiveD etector.java>`__
com.zaxxer.hikari.HikariDataSource	Hikari data source pool active detector	`org.a pache.shardingsp here.infra.datas ource.pool.destr oyer.detector.ty pe.HikariDataSou rcePoolActiveDet ector <https:/ /github.com/apac he/shardingspher e/blob/master/sh ardingsphere-inf ra/shardingspher e-infra-common/s rc/main/java/org /apache/sharding sphere/infra/dat asource/pool/des troyer/detector/ type/HikariDataS ourcePoolActiveD etector.java>`__

7.5 SQL Parser

7.5.1 DatabaseTypedSQLParserFacade

Fully-qualified class name

`org.apache.shardingsphere.sql.parser.spi.DatabaseTypedSQLParserFacade <https://github.com/apache/shardingsphere/blob/master/shardingsphere-sql-parser/shardingsphere-sql-pa

rser-spi/src/main/java/org/apache/shardingsphere/sql/parser/spi/DatabaseTypedSQLParserFacade.java>`__

Definition

Database typed SQL parser facade service definition

Implementation classes

Configuration Type	*Description*	*Fully-qualified class name*
MySQL	SQL parser entry based on MySQL	`org.apache.s hard-ingsphere.sql.parser.mysql.parser.MySQ LParserFacade <http s://github.com/apache/shardingsphere/blob/master/shardingsphere-sql-parser/shardingsphere-sql-parser-dialect/shardingsphere-sql-parser-mysql/src/main/java/org/apache/shardingsphere/sql/parser/mysql/parser/MySQLParserFacade.java>`__
PostgreSQL	SQL parser entry based on PostgreSQL	`org.apache.shard in-gsphere.sql.parser.postgresql.parser.PostgreSQLParserFacade <https://github.com/apache/shardingsphere/blob/master/sharding sphere-sql-parser/sha rdingsphere-sql-parse r-dialect/shardingsphere-sql-parser-postgresql/src/main/java/org/apache/shardingsphere/sql/parser/postgresql/parser/PostgreSQL Parser-Facade.java>`__
SQLServer	SQL parser entry based on SQLServer	`org.apache. shard-ingsphere.sql.pa rser.sqlserver.parser .SQLServerParserFacade <https://github.com/apache/shardingsphere/blob/master/shardingsphere-sql-parser/shardingsphere-sql-parser-dialect/shardingsphere-sql-parser-sqlserver/src/main/java/org/apache/shardingsphere/sql/parser/sqlserver/parser/SQLServer Parser-

7.5.2 SQLVisitorFacade

Fully-qualified class name

`org.apache.shardingsphere.sql.parser.spi.SQLVisitorFacade <https://github.com/apache/shardingsphere/blob/master/shardingsphere-sql-parser/shardingsphere-sql-parser-spi/src/main/java/org/apache/shardingsphere/sql/parser/spi/SQLVisitorFacade.java>`__

Definition

SQL visitor facade class definition

Implementation classes

• Configuration Type*	*Description*	*Fully-qualified class name*
MySQL	MySQL syntax tree visitor entry	`org.apa che. shardingsphere. sql.parser.mysql.vi sitor.statement.fac ade.MySQLStatementS QLVisitorFacade < https://github.com/ apache/shardingsphe re/blob/master/shar dingsphere-sql-pars er/shardingsphere-s ql-parser-dialect/s hardingsphere-sql-p arser-mysql/src/mai n/java/org/apache/s hardingsphere/sql/p ar- ser/mysql/visitor /state- ment/facade/M ySQL- StatementSQLVis itorFa- cade.java>`__
PostgreSQL	PostgreSQL syntax tree visitor entry	``*org.apache.sh arding- sphere.sql.pa rser.postgresql.vis itor.statement.faca de.PostgreSQLStatem entSQLVis- itorFacade* ` <https://github. com/apache/sharding sphere/blob/master/ shardingsphere-sql- parser/shardingsphe re-sql-parser-diale ct/shardingsphere-s ql-parser- postgresq l/src/main/java/org /apache/shardingsph ere/sql/parser/post gresql/visitor/stat ement/facade/Postgr eS- QLStatementSQLVis itorFa- cade.java>`__
SQLServer	SQLServer syntax tree visitor entry	`org.apac he. shardingsphere.s ql. parser.sqlserver . visitor.statement. facade.SQLServerSta tementSQLVisitor- Fac ade <https://gith ub.com/apache/shard

7.6 Proxy

7.6.1 DatabaseProtocolFrontendEngine

Fully-qualified class name

`org.apache.shardingsphere.proxy.frontend.spi.DatabaseProtocolFrontendEngine <https://github.com/apache/shardingsphere/blob/master/shardingsphere-proxy/shardingsphere-proxy-frontend/shardingsphere-proxy-frontend-spi/src/main/java/org/apache/shardingsphere/proxy/frontend/spi/DatabaseProtocolFrontendEngine.java>`__

Definition

Protocols for ShardingSphere-Proxy to parse and adapt for accessing databases.

Implementation classes

• ConfigurationType*	• Desc ript ion*	*Fully-qualified class name*
MySQL	Prot ocol im plem enta tion for M ySQL	``*org.apach e.shardingsphere.proxy.frontend.mysql.MySQLFront* ` <https://github.com/apache/shardingsphere/blob/master/sh ardingsphere-proxy/shardingsphere-proxy-frontend/shardings phere-proxy-frontend-mysql/src/main/java/org/apache/shardi ng-sphere/proxy/frontend/mysql/MySQLFrontendEn
PostgreSQL	Prot ocol im plem enta tion for Po stgr eSQL	`org.apache. shardingsphere.proxy. fr ontend.postgresql. PostgreSQLFrontendEngine <https://github.com/apache/shardingsphere/blob/master/shar pr oxy/shardingsphere-proxy-frontend/shardingsphere-proxy-fro ntend-postgresql/src/main/java/org/apache/shardingsph roxy/frontend/postgresql/PostgreSQLFrontendEn
openGauss	Prot ocol im plem enta tion for o penG auss	`org.apache. shardingsphere.pro xy.frontend.opengauss. OpenGaussFrontendEngine <https://github.com/apache/shardingsphere/blob/master/s -proxy/shardingsphere-proxy-frontend/shardingsphere-proxy- frontend-opengauss/src/main/java/org/apache/shardingsph /proxy/frontend/opengauss/OpenGaussFrontendl

7.6.2 AuthorityProvideAlgorithm

Fully-qualified class name

`org.apache.shardingsphere.authority.spi.AuthorityProviderAlgorithm <https://github.com/apache/shardingsphere/blob/master/shardingsphere-kernel/shardingsphere-authority/shardingsphere-authority-api/src/main/java/org/apache/shardingsphere/authority/spi/AuthorityProviderAlgorithm.java>`__

Definition

Loading logic for user permission.

Implementation classes

• ConfigurationType*	*De scrip tion*	*Fully-qualified class name*
ALL_PERMITTED	Grant all p ermis sions by de fault (no foren sics)	`org.apache. shardingsphere. authority.provider. simple.Al lPermitted- PrivilegesProvider- Algorithm <https://github.com/apache/shardingsphere/blob/master/sharding kerne l/shardingsphere- authority/shardingsphere- authority-core/ src/main/java/org/apache/shardingsphere/author r/simple/AllPermittedPrivilegesProviderAlgorithm
DATABASE_PERMITTED	P ermis sions confi gured by us er-da tabas e-map pings	`org.apache. shard- ingsphere.authority. provider.database. DatabasePermit tedPriv- ilegesProviderAlgorithm <https://github.com/apache/shardingsphere/blob/master/shardingsphere- kernel/shard ingsphere- authority/shardingsphere- authority-core/src/main/java/org/apache/shardingsphere/authority/prov ase/DatabasePermittedPrivilegesProviderAlgorit

7.7 Data Sharding

7.7.1 ShardingAlgorithm

Fully-qualified class name

`org.apache.shardingsphere.sharding.spi.ShardingAlgorithm <https://github.com/apache/shardingsphere/blob/master/shardingsphere-features/shardingsphere-sharding/shardingsphere-sharding-api/src/main/java/org/apache/shardingsphere/sharding/spi/ShardingAlgorithm.java>`__

Definition

Sharding Algorithm definition

Implementation classes

Conf iguration Type	*Auto Create Tables*	*Description*	*Full y-qualified class name*
MOD	Y	Modulo sharding algorithm	`org.a pache. shard ing- sphere.s hard- ing.alg orithm. shar ding.mod. Mo dShardingAl gorithm < https://git hub.com/apa che/shardin gsphere/blo b/master/sh ard- ingspher e-features/ shardingsph ere- shardin g/shardings phere-shard ing- core/sr c/main/java /org/apache /shard- ingsp here/shardi ng/algorith m/sharding/ mod/ModShar din- gAlgorit hm.java>`__
HASH_MOD	Y	Hash modulo sharding algorithm	`or g.apache.sh ardingspher e. sharding. algo- rithm.s harding. mod .HashModSha rdingAlgori thm <http s://github. com/apache/ shard- ingsph ere/blob/ma ster/shardi ngsphere- fe atures/shar dingsphere- shard- ing/sh ardingspher e- sharding- core/src/ma in/java/org /apache/sha rding- sphere /sharding/a lgorithm/sh ard- ing/mod/ HashMod- Shar dingAlgorit hm.java>`__
BOUND ARY_RANGE	Y	Boundary based range sharding algorithm	`org.ap ache. shardi ngsphere.

7.7.2 KeyGenerateAlgorithm

Fully-qualified class name

`org.apache.shardingsphere.sharding.spi.KeyGenerateAlgorithm <https://github.com/apache/shardingsphere/blob/master/shardingsphere-features/shardingsphere-sharding/shardingsphere-sharding-api/src/main/java/org/apache/shardingsphere/sharding/spi/KeyGenerateAlgorithm.java>`__

Definition

Distributed Key Generating Algorithm definition

Implementation classes

Configuration Type	*Description*	*Fully-qualified class name*
SNOWFLAKE	Snowflake key generate algorithm	``*org.apache.sh ardingsphere.shar ding.algorithm.ke ygen.SnowflakeKey GenerateAlgorithm* ` <https://githu b.com/apache/shar dingsphere/blob/m aster/shardingsph ere-features/shar dingsphere-shardi ng/shardingsphere -sharding-core/sr c/main/java/org/a pache/shardingsph ere/sharding/algo rithm/keygen/Snow flakeKeyGenerateA lgorithm.java>`__
UUID	UUID key generate algorithm	`` `org .apache.shardings phere.sharding.al gorithm.keygen.UU IDKeyGenerateAlgo rithm`` <https:// github.com/apache /shardingsphere/b lob/master/shardi ngsphere-features /shardingsphere-s harding/shardings phere-sharding-co re/src/main/java/ org/apache/shardi ngsphere/sharding /algorithm/keygen /UUIDKeyGenerateA lgorithm.java>`__
NANOID	NanoId key generate algorithm	`` `org .apache.shardings phere.sharding.na noid.algorithm. ke ygen.NanoIdKeyGen erateAlgorithm`` <https://github.c om/apache/shardin gsphere/blob/mast er/shardingsphere -features/shardin gsphere-sharding/ shardingsphere-sh arding-plugin/sha rdingsphere-shard ing-nanoid/src/ma in/java/org/apach e/shardingsphere/ sharding/nanoid/a lgorithm/keygen/N anoIdKeyGenerateA lgorithm.java>`__
COSID	CosId key generate algorithm	` ``org.apache.shar dingsphere.shardi ng.cosid.algorith m.keygen.CosIdKey GenerateAlgorithm `` <https://githu b.com/apache/shar dingsphere/blob/m aster/shardingsph ere-features/shar dingsphere-shardi ng/shardingsphere -sharding-plugin/ shardingsphere-sh arding-cosid/src/ main/java/org/apa che/shardingspher e/sharding/cosid/ algorithm/keygen/ CosIdKeyGenerateA lgorithm.java>`__
COSID_SNOWFLAKE	Snowflake key generate algorithm provided by CosId	`` *org.apache.shard ingsphere.shardin g.cosid.algorithm .keygen.CosIdSnow flakeKeyGenerateA lgorithm*` <https ://github.com/apa che/shardingspher e/blob/master/sha rdingsphere-featu res/shardingspher e-sharding/shardi ngsphere-sharding -plugin/shardings phere-sharding-co sid/src/main/java /org/apache/shard ingsphere/shardin g/cosid/algorithm /keygen/CosIdSnow flakeKeyGenerateA lgorithm.java>`__

7.7.3 ShardingAuditAlgorithm

Fully-qualified class name

`org.apache.shardingsphere.sharding.spi.ShardingAuditAlgorithm <https://github.com/apache/shardingsphere/blob/master/shardingsphere-features/shardingsphere-sharding/shardingsphere-sharding-api/src/main/java/org/apache/shardingsphere/sharding/spi/ShardingAuditAlgorithm.java>`__

Definition

Sharding audit algorithm definition

Implementation classes

Co nfiguration Type	*Description*	• Fully-qualified class name*
D ML_SHARDING _CONDI-TIONS	Prohibit DML auditing algorithm without sharding conditions	` org.apache.shar ding-sphere.shar ding. algorithm. audit. DMLShardi ngCondition-sSha rdingAuditAlgor ithm <https:/ /github.com/apache/shardingsph ere/blob/master /shard-ingsphere -features/shard ingsphere-shard ing/shardingsph ere-sharding-co re/src/main/jav a/org/apache/sh arding-sphere/sh arding/algorith m/audit/DMLShar dingCon-ditionsS hardingAuditAlg orithm.java>`__

7.7.4 DatetimeService

Fully-qualified class name

`org.apache.shardingsphere.infra.datetime.DatetimeService <https://github.com/apache/shardingsphere/blob/master/shardingsphere-infra/shardingsphere-infra-datetime/shardingsphere-infra-datetime-spi/src/main/java/org/apache/shardingsphere/infra/datetime/DatetimeService.java>`__

Definition

Obtain the current date for routing definition

Implementation classes

Co nfiguration Type	*Description*	• Fully-qualified class name*
D atabaseDate timeService	Get the current time from the database for routing	`org.apa che. shardingsph ere.agent. metri cs.prometheus.s ervice.Promethe us-PluginBootSer vice <h t t p s : / /github.com/apa che/shardingsph ere/blob/master /shard-ingsphere -infra/sharding sphere-infra-da tetime/sharding sphere-infra-da tetime-type/sha rdingsphere-dat abase-datetime/ src/main/java/o rg/apache/shard ing-sphere/datet ime/database/Da tabaseDatetimeS er-vice.java>`__
Sys temDatetime	Get the current time from the application system for routing	`org.apac he. shardingsphe re. datetime.sys tem. SystemDatet imeSer-vice <h ttps://github.c om/apache/shard ing-sphere/blob/ master/sharding sphere-infra/sh ardingsphere-in fra-datetime/sh ardingsphere-in fra-datetime-ty pe/shardingsphe re-system-datet ime/src/main/ja va/org/apache/s harding-sphere/d atetime/system/ Sys-temDatetimeS ervice.java>`__

7.8 Readwrite-splitting

7.8.1 ReadQueryLoadBalanceAlgorithm

Fully-qualified class name

`org.apache.shardingsphere.readwritesplitting.spi.ReadQueryLoadBalanceAlgorithm <https://github.com/apache/shardingsphere/blob/master/shardingsphere-features/shardingsphere-readwrite-splitting/shardingsphere-readwrite-splitting-api/src/main/java/org/apache/shardingsphere/readwritesplitting/spi/ReadQueryLoadBalanceAlgorithm.java>`__

Definition

Read query load balance algorithm' s definition

Implementation classes

C onf igu rat ion Ty pe	Description	Ful ly-q uali fied c lass n ame
RO UND _RO BIN	the read database load balancer algorithm based on polling	`` org .apa che. shar ding sphe re.r eadw rite spli ttin g.al gori thm. load bala nce. Roun dRob inRe adQu eryL oadB alan ceAl gori thm ` <h ttps ://g ithu b.co m/ap ache /sha rdin gsph ere/ blob /mas ter/ shar ding sphe re-f eatu res/ shar ding sphe re-r eadw rite -spl itti ng/s hard ings pher e-re adwr ite- spli ttin g-co re/s rc/m ain/ java /org /apa che/ shar ding sphe re/r eadw rite spli ttin g/al gori thm/ load bala nce/ Roun dRob inRe adQu eryL oadB alan ceAl gori thm. java >`__
RAN DOM	the read database load balancer algorithm based on random	`` org .apa che. shar ding sphe re.r eadw rite spli ttin g.al gori thm. load bala nce. Rand omRe adQu eryL oadB alan ceAl gori thm ` <h ttps ://g ithu b.co m/ap ache /sha rdin gsph ere/ blob /mas ter/ shar ding sphe re-f eatu res/ shar ding sphe re-r eadw rite -spl itti ng/s hard ings pher e-re adwr ite- spli ttin g-co re/s rc/m ain/ java /org /apa che/ shar ding sphe re/r eadw rite spli ttin g/al gori thm/ load bala nce/ Rand omRe adQu eryL oadB alan ceAl gori thm. java >`__
WEI GHT	the read database load balancer algorithm based on weight	`` org .apa che. shar ding sphe re.r eadw rite spli ttin g.al gori thm. load bala nce. Weig htRe adQu eryL oadB alan ceAl gori thm ` <h ttps ://g ithu b.co m/ap ache /sha rdin gsph ere/ blob /mas ter/ shar ding sphe re-f eatu res/ shar ding sphe re-r eadw rite -spl itti ng/s hard ings pher e-re adwr ite- spli ttin g-co re/s rc/m ain/ java /org /apa che/ shar ding sphe re/r eadw rite spli ttin g/al gori thm/ load bala nce/ Weig htRe adQu eryL oadB alan ceAl gori thm. java >`__
TRA NSA CTI ON_ RAN DOM	Whether in a transaction or not, read requests are routed to multiple replicas using a random strategy	``o rg.a pach e.sh ardi ngsp here .rea dwri tesp litt ing. algo rith m.lo adba lanc e.Tr ansa ctio nRan domR eadQ uery Load Bala nceA lgor ithm ` < http s:// gith ub.c om/a pach e/sh ardi ngsp here /blo b/ma ster /sha rdin gsph ere- feat ures /sha rdin gsph ere- read writ e-sp litt ing/ shar ding sphe re-r eadw rite -spl itti ng-c ore/ src/ main /jav a/or g/ap ache /sha rdin gsph ere/ read writ espl itti ng/a lgor ithm /loa dbal ance /Tra nsac tion Rand omRe adQu eryL oadB alan ceAl gori thm. java >`__
TR ANS ACT ION _RO UND RO BIN	Whether in a transaction or not, read requests are routed to multiple replicas using a round-robin strategy	``o rg.a pach e.sh ardi ngsp here .rea dwri tesp litt ing. algo rith m.lo adba lanc e.Tr ansa ctio nRou ndRo binR eadQ uery Load Bala nceA lgor ithm `< http s:// gith ub.c om/a pach e/sh ardi ngsp here /blo b/ma ster /sha rdin gsph ere- feat ures /sha rdin gsph ere- read writ e-sp litt ing/ shar ding sphe re-r eadw rite -spl itti ng-c ore/ src/ main /jav a/or g/ap ache /sha rdin gsph ere/ read writ espl itti ng/a lgor ithm /loa dbal ance /Tra nsac tion Roun dRob inRe adQu eryL oadB alan ceAl gori thm. java >`__
TRA NSA	Whether in a transaction or not, read requests are	``o rg.a pach e.sh ardi ngsp here .rea dwri tesp litt ing. algo rith m.lo adba lanc e.Tr ansa ctio nWei ghtR eadQ uery Load Bala nceA lgor

7.9 HA

7.9.1 DatabaseDiscoveryProviderAlgorithm

Fully-qualified class name

`org.apache.shardingsphere.dbdiscovery.spi.DatabaseDiscoveryProviderAlgorithm <https://github.com/apache/shardingsphere/blob/master/shardingsphere-features/shardingsphere-db-discovery/shardingsphere-db-discovery-api/src/main/java/org/apache/shardingsphere/dbdiscovery/spi/DatabaseDiscoveryProviderAlgorithm.java>`__

Definition

Database discovery provider algorithm's definition

Implementation classes

• Configuration Type*	*Description*	*Fu lly-qualified class name*
MySQL.MGR	MySQL MGR-based database discovery provider algorithm	`org.apac he. shardingsp here. dbdiscov ery.mysql.typ e.MGRMySQLDat abaseDis- cover yProviderAlgo rithm <http s://github.co m/apache/shar dingsphere/bl ob/master/sha rdingsphere-f eatures/shard ingsphere-db- discovery/sha rdingsphere-d b-discovery-p rovider/shard ingsphere-db- discovery-mys ql/src/main/j ava/org/apach e/shardingsph ere/dbdiscove ry/mysql/type /MGRMySQL- Data baseDiscovery Provider- Algor ithm.java>`__
MySQL.NORMA L_REPLICATION	Database discovery provider algorithm of MySQL's replication	`org.apache . shardingsphe re. dbdiscover y.mysql. type. MySQLNormalRe plicationData baseDis- covery ProviderAlgor ithm <https ://github.com /apache/shard ingsphere/blo b/master/shar dingsphere-fe atures/shardi ngsphere-db-d iscovery/shar dingsphere-db -discovery-pr ovider/shardi ngsphere-db-d iscovery-mysq l/src/main/ja va/org/apache /shardingsphe re/dbdiscover y/mysql/type/ MySQL- NormalRe plicationData baseDiscovery ProviderAlgor ithm.java>`__
op enGauss.NORMA L_REPLICATION	Database discovery provider algorithm of openGauss' s replication	`org.apa che.shardings phere.dbdisco very. opengaus s.OpenGaussNo rmalReplicati on- DatabaseDis cov- eryProvide rAlgorithm <https://gith ub.com/apache /shardingsphe re/blob/maste r/shardingsph ere-features/

7.10 Distributed Transaction

7.10.1 ShardingSphereTransactionManager

Fully-qualified class name

`org.apache.shardingsphere.transaction.spi.ShardingSphereTransactionManager <https://github.com/apache/shardingsphere/blob/master/shardingsphere-kernel/shardingsphere-transaction/shardingsphere-transaction-core/src/main/java/org/apache/shardingsphere/transaction/spi/ShardingSphereTransactionManager.java>`__

Definition

ShardingSphere transaction manager service definition

Implementation classes

• Configuration Type*	*Description*	*Fully-qualified class name*
XA	XA distributed transaction manager	``*org.apache.shard in-gsphere.transactio n.xa.XAShardingSpher eTransactionManager* ` <https://github.co m/apache/shardingsph ere/blob/master/shar dingsphere-kernel/sh ardingsphere-transac tion/shardingsphere-transaction-type/sha rdingsphere-transact ion-xa/shardingspher e-transaction-xa-cor e/src/main/java/org/ apache/shardingspher e/transaction/xa/XAS hard-ingSphereTransac tionMan-ager.java>`__
BASE	Seata distributed transaction manager	`org.apache.shard in-gsphere.transactio n.base.seata.at.Seat aATShardingSphere-Tra nsactionManager <h ttps://github.com/ap ache/shardingsphere/ blob/master/sharding sphere-kernel/shardi ngsphere-transaction /shardingsphere-tran saction-type/shardin gsphere-transaction-base/shardingsphere-transaction-base-sea ta-at/src/main/java/ org/apache/shardings phere/transaction/ba se/seata/at/SeataATS hard-ingSphereTransac tionMan-ager.java>`__

7.10.2 XATransactionManagerProvider

Fully-qualified class name

`org.apache.shardingsphere.transaction.xa.spi.XATransactionManagerProvider <https://github.com/apache/shardingsphere/blob/master/shardingsphere-kernel/shardingsphere-transaction/shardingsphere-transaction-type/shardingsphere-transaction-xa/shardingsphere-transaction-xa-spi/src/main/java/org/apache/shardingsphere/transaction/xa/spi/XATransactionManagerProvider.java>`__

Definition

XA transaction manager provider definition

Implementation classes

Configuration Type	*Description*	*Fully-qualified class name*
Atomikos	XA distributed transaction manager based on Atomikos	`org.apache. shardingsphere.tr ansaction.xa.atom ikos. manager.Atom ikosTransactionMa nagerProvider < https://github .co m/apache/sharding sphere/blob/maste r/shardingsphere- kernel/shardingsp here-transaction/ shardingsphere-tr ansaction-type/sh ardingsphere-tran saction-xa/shardi ngsphere-transact ion-xa-provider/s hardingsphere-tra nsaction-xa-atomi kos/src/main/java /org/apache/shard ingsphere/transac tion/xa/atomikos/ manager/AtomikosT ransactionManager Provider.java>`__
Narayana	XA distributed transaction manager based on Narayana	` org.apache.shar dingsphere.transa ction.xa.narayana .manager.Narayana XATransactionMana gerProvider <ht tps://github.com/ apache/shardingsp here/blob/master/ shardingsphere-ke rnel/shardingsphe re-transaction/sh ardingsphere-tran saction-type/shar dingsphere-transa ction-xa/sharding sphere-transactio n-xa-provider/sha rdingsphere-trans action-xa-narayan a/src/main/java/o rg/apache/shardin gsphere/transacti on/xa/narayana/ma nager/NarayanaXAT ransactionManager Provider.java>`__
Bitronix	XA distributed transaction manager based on Bitronix	` org.apache.shar dingsphere.transa ction.xa.bitronix .manager.Bitronix XATransactionMana gerProvider <ht tps://github.com/ apache/shardingsp here/blob/master/ shardingsphere-ke rnel/shardingsphe re-transaction/sh ardingsphere-tran saction-type/shar dingsphere-transa ction-xa/sharding sphere-transactio n-xa-provider/sha rdingsphere-trans action-xa-bitroni x/src/main/java/o rg/apache/shardin gsphere/transacti on/xa/bitronix/ma nager/BitronixXAT ransactionManager Provider.java>`__

7.10.3 XADataSourceDefinition

Fully-qualified class name

`org.apache.shardingsphere.transaction.xa.jta.datasource.properties.XADataSourceDefinition <https://github.com/apache/shardingsphere/blob/master/shardingsphere-kernel/shardingsphere-transaction/shardingsphere-transaction-type/shardingsphere-transaction-xa/shardingsphere-transaction-xa-core/src/main/java/org/apache/shardingsphere/transaction/xa/jta/datasource/properties/XADataSourceDefinition.java>`__

Definition

XA Data source definition

Implementation classes

Configuration Type	*Description*	*Fu lly-qualified class name*
MySQL	Auto convert Non XA MySQL data source to XA MySQL data source	`o rg.apache.sha rdingsphere.t ransaction.xa .jta.datasour ce.properties .dialect.MySQ LXADataSource Definition <https://gith ub.com/apache /shardingsphe re/blob/maste r/shardingsph ere-kernel/sh ardingsphere- transaction/s hardingsphere -transaction-type/sharding sphere-transa ction-xa/shar dingsphere-tr ansaction-xa-core/src/main /java/org/apa che/shardings phere/transac tion/xa/jta/d ata-source/pro perties/diale ct/MySQLXADat aSourceDefini tion.java>`__
MariaDB	Auto convert Non XA MariaDB data source to XA MariaDB data source	`org.a pache.shardin gsphere.trans action.xa.jta . datasource.p roperties.dia lect.MariaDBX ADataSourceDe finition <h ttps://github .com/apache/s hardingsphere /blob/master/ shardingspher e-kernel/shar dingsphere-tr ansaction/sha rdingsphere-t ransaction-ty pe/shardingsp here-transact ion-xa/shardi ngsphere-tran saction-xa-co re/src/main/j ava/org/apach e/shardingsph ere/transacti on/xa/jta/dat asource/prope rties/dialect /MariaDBXADat aSourceDefini tion.java>`__
PostgreSQL	Auto convert Non XA PostgreSQL data source to XA PostgreSQL data source	` org.apache. shardingspher e.transaction .xa.jta.datas ource.propert ies.dialect.P ostgreSQLXADa taSourceDefin ition <http s://github.co m/apache/shar dingsphere/bl ob/master/sha rdingsphere-k ernel/shardin gsphere-trans action/shardi ngsphere-tran saction-type/ shardingspher e-transaction -xa/shardings phere-transac tion-xa-core/ src/main/java /org/apache/s hardingsphere /transaction/ xa/jta/dataso urce/properti es/dialect/Po stgreSQLXADat aSourceDefini tion.java>`__
Oracle	Auto convert Non XA Oracle data source to XA Oracle data source	`org .apache.shard ingsphere.tra nsaction.xa.j ta. datasource .properties.d ialect.Oracle XADataSourceD efinition < https://githu b.com/apache/ shardingspher e/blob/master /shardingsphe re-kernel/sha rdingsphere-t ransaction/sh ardingsphere- transaction-type/shardings phere-transac tion-xa/shard ingsphere-tra nsaction-xa-c ore/src/main/ java/org/apac he/shardingsp here/transact ion/xa/jta/da tasource/prop erties/dialec t/OracleXADat aSourceDefini tion.java>`__
SQLServer	Auto convert Non XA SQLServer data source to XA SQLServer data source	`org.apach e.shardingsph ere.transacti on.xa.jta.dat asource.prope rties.dialect .SQLServerXAD ataSourceDefi nition <htt ps://github.c om/apache/sha rdingsphere/b lob/master/sh ardingsphere- kernel/shardi ngsphere-tran saction/shard ingsphere-tra nsaction-type /shardingsphe re-transactio n-xa/sharding sphere-transa ction-xa-core /src/main/jav a/org/apache/ shardingspher e/transaction /xa/jta/datas ource/propert ies/dialect/S QLServerXADat aSourceDefini tion.java>`__
H2	Auto convert Non XA H2 data	``org.apac he.shardingsp here.transact ion.xa.jta.da tasource.prop erties.dialec t.H2XADataSou rceDefinition ` <https://g ithub.com/apa che/shardings phere/blob/ma ster/sharding sphere-kernel /shardingsphe re-transactio

7.10.4 DataSourcePropertyProvider

Fully-qualified class name

`org.apache.shardingsphere.transaction.xa.jta.datasource.swapper.DataSourcePropertyProvider` <https://github.com/apache/shardingsphere/blob/master/shardingsphere-kernel/shardingsphere-transaction/shardingsphere-transaction-type/shardingsphere-transaction-xa/shardingsphere-transaction-xa-core/src/main/java/org/apache/shardingsphere/transaction/xa/jta/datasource/swapper/DataSourcePropertyProvider.java>`__

Definition

Data source property provider service definition

Implementation classes

Configuration Type	*Description*	*Fully-qualified class name*
com.zaxxer.hikari.HikariDataSource	Used to get standard properties of HikariCP	`org.apache.s hardingsphere.tr ansaction.xa.jta .datasource.swap per.impl.HikariC PPropertyProvide r <https://github.com/apache/s hardingsphere/bl ob/master/shardi ngsphere-kernel/shardingsphere-t ransaction/shard ingsphere-transa ction-type/shard ingsphere-transa ction-xa/shardin gsphere-transact ion-xa-core/src/ main/java/org/ap ache/shardingsph ere/transaction/ xa/jta/datasourc e/swapper/impl/H ikariCPPropertyP rovider.java>`__

7.11 SQL Checker

7.11.1 SQLChecker

Fully-qualified class name

`org.apache.shardingsphere.infra.executor.check.SQLChecker` <https://github.com/apache/shardingsphere/blob/master/shardingsphere-infra/shardingsphere-infra-executor/src/main/java/org/apache/shardingsphere/infra/executor/check/SQLChecker.java>`__

Definition

SQL checker class definition

Implementation classes

Configuration Type	*Description*	*Fully-qualified class name*
AuthorityRule.class	Authority checker	`org.apache.shardingsphere.authority.checker.AuthorityChecker <https://github.com/apache/shardingsphere/blob/master/shardingsphere-kernel/shardingsphere-authority/shardingsphere-authority-core/src/main/java/org/apache/shardingsphere/authority/checker/AuthorityChecker.java>`__
ShardingRule.class	Sharding audit checker	``*org.apache.shardingsphere.sharding.checker.audit.ShardingAuditChecker* ` <https://github.com/apache/shardingsphere/blob/master/shardingsphere-features/shardingsphere-sharding/shardingsphere-sharding-core/src/main/java/org/apache/shardingsphere/sharding/checker/audit/ShardingAuditChecker.java>`__

7.12 Encryption

7.12.1 EncryptAlgorithm

Fully-qualified class name

`org.apache.shardingsphere.encrypt.spi.EncryptAlgorithm <https://github.com/apache/shardingsphere/blob/master/shardingsphere-features/shardingsphere-encrypt/shardingsphere-encrypt-api/src/main/java/org/apache/shardingsphere/encrypt/spi/EncryptAlgorithm.java>`__

Definition

Data encrypt algorithm definition

Implementation classes

Configuration Type	*Description*	*Fully-qualified class name*
MD5	MD5 data encrypt algorithm	`org.apache. shardingsphere.encryption.algorithm.MD5Encrypt <https://github.com/apache/shardingsphere/blob/master/shardingsphere-features/shardingsphere-encrypt/shardingsphere-encrypt-core/src/main/java/org/apache/shardingsphere/encrypt/algorithm/MD5EncryptAlgorithm.java>`__
AES	AES data encrypt algorithm	`org.apache. shardingsphere.encryption.algorithm.AESEncrypt <https://github.com/apache/shardingsphere/blob/master/shardingsphere-features/shardingsphere-encrypt/shardingsphere-encrypt-core/src/main/java/org/apache/shardingsphere/encrypt/algorithm/AESEncryptAlgorithm.java>`__
RC4	RC4 data encrypt algorithm	`org.apache. shardingsphere.encryption.algorithm.RC4Encrypt <https://github.com/apache/shardingsphere/blob/master/shardingsphere-features/shardingsphere-encrypt/shardingsphere-encrypt-core/src/main/java/org/apache/shardingsphere/encrypt/algorithm/RC4EncryptAlgorithm.java>`__
SM3	SM3 data encrypt algorithm	`org.apache.shardingsphere.encryption. algor ithm. SM3Encrypt <https://github.com/apache/shardingsphere/blob/master/shardingsphere-features/

7.13 Shadow DB

7.13.1 ShadowAlgorithm

Fully-qualified class name

`org.apache.shardingsphere.shadow.spi.ShadowAlgorithm <https://github.com/apache/shardingsphere/blob/master/shardingsphere-features/shardingsphere-shadow/shardingsphere-shadow-api/src/main/java/org/apache/shardingsphere/shadow/spi/ShadowAlgorithm.java>`__

Definition

Shadow algorithm' s definition

Implementation classes

Configuration Type	*Description*	*Fully-qualified class name*
VALUE_MATCH	Match shadow algorithms based on field values	`org.apache.shardingsphere.shadow.algorithm.shadow.column.ColumnValueMatchedShadowAlgorithm <https://github.com/apache/shardingsphere/blob/master/shardingsphere-features/shardingsphere-shadow/shardingsphere-shadow-core/src/main/java/org/apache/shardingsphere/shadow/algorithm/shadow/column/ColumnValueMatchedShadowAlgorithm.java>`__
REGEX_MATCH	Regular matching shadow algorithm based on field value	`org.apache.shardingsphere.shadow.algorithm.shadow.column.ColumnRegexMatchedShadowAlgorithm <https://github.com/apache/shardingsphere/blob/master/shardingsphere-features/shardingsphere-shadow/shardingsphere-shadow-core/src/main/java/org/apache/shardingsphere/shadow/algorithm/shadow/column/ColumnRegexMatchedShadowAlgorithm.java>`__
SIMPLE_HINT	Simple match shadow algorithm based on Hint	`org.apache.shardingsphere.shadow.algorithm.shadow.hint.SimpleHintShadowAlgorithm <https://github.com/apache/shardingsphere/blob/master/shardingsphere-features/shardingsphere-shadow/shardingsphere-shadow-core/src/main/java/org/apache/shardingsphere/shadow/algorithm/shadow/hint/SimpleHintShadowAlgorithm.java>`__

7.14 Observability

7.14.1 PluginBootService

Fully-qualified class name

`org.apache.shardingsphere.agent.spi.boot.PluginBootService <https://github.com/apache/shardingsphere/blob/master/shardingsphere-agent/shardingsphere-agent-api/src/main/java/org/apache/shardingsphere/agent/spi/boot/PluginBootService.java>`__

Definition

Plugin startup service definition

Implementation classes

• Configuration Type*	*Description*	*Fully-qualified class name*
Prometheus	Prometheus plugin startup class	` org.apache.shardingsphere.agent.metrics.prometheus.service.PrometheusPluginBootService <https://github.com/apache/shardingsphere/blob/master/shardin gsphere-agent/shard ingsphere-agent-plu gins/shardingsphere-agent-plugin-metrics/shardingsphere-agent-metrics-prometheus/src/main/java/org/apache/shardingsphere/agent/metrics/prometheus/service/PrometheusPluginB oot-Service.java>`__
Logging	Logging plugin startup class	`org.apache.shar ding-sphere.agent.pl ugin.logging.base.s ervice.BaseLoggingP luginBoot-Service <https://github.com/apache/shardingsphere/blob/master/shardingsphere-agent/shardingsphere-agent-plugins/shardingsphere-agent-plugin-l og-ging/shardingsphe re-agent-logging-ba se/src/main/java/org/apache/shardingsphere/agent/plugin/l og-ging/base/service /BaseL-oggingPluginB ootSer-vice.java>`__
Jaeger	Jaeger plugin startup class	`org.ap ache.shardingsphere .agent.plugin.traci ng.jaeger.service.J aegerTrac-ingPluginB ootService <https ://github.com/apache/shardingsphere/blob/master/shardings phere-

7.14.2 PluginDefinitionService

Fully-qualified class name

`org.apache.shardingsphere.agent.spi.definition.PluginDefinitionService <https://github.com/apache/shardingsphere/blob/master/shardingsphere-agent/shardingsphere-agent-api/src/main/java/org/apache/shardingsphere/agent/spi/definition/PluginDefinitionService.java>`__

Definition

Agent plugin definition

Implementation classes

• Configuration Type*	*Description*	*Fully-qualified class name*
Prometheus	Prometheus plugin definition	``*org.apache.shard ingsphere.agent.metrics.prometheus.definition.PrometheusPlug inDefinitionService*` <https://github.com/apache/shardingsphere/blob/master/shardingsphere-agent/shardingsphere-agent-plugins/shardingsphere-agent-plugin-metrics/shardingsphere-agent-metrics-prometheus/src/main/java/org/apache/shardingsphere/agent/metrics/prometheus/definition/PrometheusPluginDefini tionService.java>`__
Logging	Logging plugin definition	`` `org.apache.sha rdingsphere.agent.plugin.logging.base.definition.BaseLogging PluginDefinitionServ ice `` <https://github.com/apache/shardingsphere/blob/master/shardingsphere-agent/shardingsphere-agent-plugins/shardingsphere-agent-plugin-logging/shardingsphere-agent-logging-base/src/main/java/org/apache/shardingsphere/agent/plugin/logging/base/definition/Bas eLoggingPluginDefini tionService.java>`__
Jaeger	Jaeger plugin definition	`` `org.apache .shardingsphere.agent.plugin.tracing.jae ger.definition.Jaege rPluginDefinitionSer vice `` <https://gith

8 Test Manual

Apache ShardingSphere provides test engines for integration, module and performance.

8.1 Integration Test

Provide point to point test which connect real ShardingSphere and database instances.

They define SQLs in XML files, engine run for each database independently. All test engines designed to modify the configuration files to execute all assertions without any **Java code** modification. It does not depend on any third-party environment, ShardingSphere-Proxy and database used for testing are provided by docker image.

8.2 Module Test

Provide module test engine for complex modules.

They define SQLs in XML files, engine run for each database independently too It includes SQL parser and SQL rewriter modules.

8.3 Performance Test

Provide multiple performance test methods, includes Sysbench, JMH or TPCC and so on.

8.4 Sysbench Test

8.5 Integration Test

8.5.1 Design

The integration testing consists of three modules: test case, test environment and test engine.

Test case

It is used to define the SQL to be tested and the assertion data of the test results.

Each case defines one SQL, which can define multiple database execution types.

Test environment

It is used to set up the database and ShardingSphere-Proxy environment for running test cases. The environment is classified into environment preparation mode, database type, and scenario.

Environment preparation mode is divided into Native and Docker, and Embed type will be supported in the future. - Native environment is used for test cases to run directly in the test environment provided by the developer, suitable for debugging scenarios; - Docker environment is directly built when Maven runs the Docker-Compose plug-in. It is suitable for cloud compilation environment and testing ShardingSphere-Proxy, such as GitHub Action; - Embed environment is built when the test framework automatically builds embedded MySQL. It is suitable for the local environment test of ShardingSphere-JDBC.

Currently, the Native environment is adopted by default, and ShardingSphere-JDBC + H2 database is used to run test cases. Maven's `-pit. Env.docker` parameter specifies how the Docker environment is run. In the future, ShardingSphere-JDBC + MySQL of the Embed environment will be adopted to replace the default environment type used when Native executes test cases.

Database types currently support MySQL, PostgreSQL, SQLServer, and Oracle, and test cases can be executed using ShardingSphere-JDBC or ShardingSphere-Proxy.

Scenarios are used to test the supporting rules of ShardingSphere. Currently, data sharding and read/write splitting and other related scenarios are supported, and the combination of scenarios will be improved continuously in the future.

Test engine

It is used to read test cases in batches and execute and assert test results line by line.

The test engine arranges test cases and environments to test as many scenarios as possible with the fewest test cases.

Each SQL generates a test report in the combination of `database type * access port type * SQL execution mode * JDBC execution mode * Scenario`. Currently, each dimension is supported as follows:

- Database types: H2, MySQL, PostgreSQL, SQLServer, and Oracle;
- Access port types: ShardingSphere-JDBC and ShardingSphere-Proxy;
- SQL execution modes: Statement and PreparedStatement;
- JDBC execution modes: execute and executeQuery/executeUpdate;
- Scenarios: database shards, table shards, read/write splitting and sharding + read/write splitting

Therefore, one SQL will drive `Database type (5) * Access port type (2) * SQL execution mode (2) * JDBC execution mode (2) * Scenario (4) = 160` test cases to be run to achieve the pursuit of high quality.

8.5.2 User Guide

Module path: `shardingsphere-test/shardingsphere-integration-test/shardingsphere-integration-test-suite`

Test case configuration

SQL test case is in `resources/cases/${SQL-TYPE}/${SQL-TYPE}-integration-test-cases.xml`.

The case file format is as follows:

```
<integration-test-cases>
    <test-case sql="${SQL}">
        <assertion parameters="${value_1}:${type_1}, ${value_2}:${type_2}"
expected-data-file="${dataset_file_1}.xml" />
        <!-- ... more assertions -->
        <assertion parameters="${value_3}:${type_3}, ${value_4}:${type_4}"
expected-data-file="${dataset_file_2}.xml" />
     </test-case>

    <!-- ... more test cases -->
</integration-test-cases>
```

The lookup rule of `expected-data-file`is as follows: 1. Find the file `dataset\${SCENARIO_NAME}\${DATABASE_TYPE}\${dataset_file}.xml` in the same level directory; 2.

Find the file `dataset\${SCENARIO_NAME}\${dataset_file}.xml` in the same level directory; 3. Find the file `dataset\${dataset_file}.xml` in the same level directory; 4. Report an error if none of them are found.

The assertion file format is as follows:

```
<dataset>
    <metadata>
        <column name="column_1" />
        <!-- ... more columns -->
        <column name="column_n" />
    </metadata>
    <row values="value_01, value_02" />
    <!-- ... more rows -->
    <row values="value_n1, value_n2" />
</dataset>
```

Environment configuration

`${SCENARIO-TYPE}` Refers to the scenario name used to identify a unique scenario during the test engine run. `${DATABASE-TYPE}` refers to the database types.

Native environment configuration

Directory: `src/test/resources/env/${SCENARIO-TYPE}`

- `scenario-env.properties`: data source configuration；
- `rules.yaml`: rule configuration；
- `databases.xml`: name of the real database；
- `dataset.xml`: initialize the data；
- `init-sql\${DATABASE-TYPE}\init.sql`: initialize the database and table structure；
- `authority.xml`: to be supplemented.

Docker environment configuration

Directory: `src/test/resources/docker/${SCENARIO-TYPE}`

- `docker-compose.yml`: Docker-Compose config files, used for Docker environment startup；
- `proxy/conf/config-${SCENARIO-TYPE}.yaml`: rule configuration。

The Docker environment configuration provides a remote debugging port for ShardingSphere-Proxy. You can find the second exposed port for remote debugging in ``shardingsphere-proxy`` of the ``docker-comemage. yml`` file.

Run the test engine

Configure the running environment of the test engine

Control the test engine by configuring `src/test/resources/env/engine-env.properties`.

All attribute values can be dynamically injected via Maven command line `-D`.

```
# Scenario type. Multiple values can be separated by commas. Optional values: db,
tbl, dbtbl_with_replica_query, replica_query
it.scenarios=db,tbl,dbtbl_with_replica_query,replica_query

# Whether to run additional test cases
it.run.additional.cases=false

# Configure the environment type. Only one value is supported. Optional value:
docker or null. The default value: null.
it.cluster.env.type=${it.env}
# Access port types to be tested. Multiple values can be separated by commas.
Optional value: jdbc, proxy. The default value: jdbc
it.cluster.adapters=jdbc

# Scenario type. Multiple values can be separated by commas. Optional value: H2,
MySQL, Oracle, SQLServer, PostgreSQL
it.cluster.databases=H2,MySQL,Oracle,SQLServer,PostgreSQL
```

Run debugging mode

- Standard test engine Run `org.apache.shardingsphere.test.integration.engine.${SQL-TYPE}.General${SQL-TYPE}IT` to start the test engines of different SQL types.
- Batch test engine Run `org.apache.shardingsphere.test.integration.engine.dml.BatchDMLIT` to start the batch test engine for the test `addBatch()` provided for DML statements.
- Additional test engine Run `org.apache.shardingsphere.test.integration.engine.${SQL-TYPE}.Additional${SQL-TYPE}IT` to start the test engine with more JDBC method calls. Additional test engines need to be enabled by setting `it.run.additional.cases=true`.

Run Docker mode

```
./mvnw -B clean install -f shardingsphere-test/shardingsphere-integration-test/pom.
xml -Pit.env.docker -Dit.cluster.adapters=proxy,jdbc -Dit.scenarios=${scenario_
name_1,scenario_name_2,scenario_name_n} -Dit.cluster.databases=MySQL
```

Run the above command to build a Docker mirror `apache/shardingsphere-proxy-test:latest` used for integration testing. If you only modify the test code, you can reuse the existing test mirror without rebuilding it. Skip the mirror building and run the integration testing directly with the following command:

```
./mvnw -B clean install -f shardingsphere-test/shardingsphere-integration-test/
shardingsphere-integration-test-suite/pom.xml -Pit.env.docker -Dit.cluster.
adapters=proxy,jdbc -Dit.scenarios=${scenario_name_1,scenario_name_2,scenario_name_
n} -Dit.cluster.databases=MySQL
```

Notice

1. To test Oracle, add an Oracle driver dependency to pom.xml.
2. In order to ensure the integrity and legibility of the test data, 10 database shards and 10 table shards are used in the sharding of the integration testing, which takes a long time to run the test cases completely.

8.6 Performance Test

Provides result for each performance test tools.

8.6.1 SysBench ShardingSphere-Proxy Empty Rule Performance Test

Objectives

Compare the performance of ShardingSphere-Proxy and MySQL 1. Sysbench directly carries out stress testing on the performance of MySQL. 1. Sysbench directly carries out stress testing on ShardingSphere-Proxy (directly connect MySQL).

Based on the above two groups of experiments, we can figure out the loss of MySQL when using ShardingSphere-Proxy.

Set up the test environment

Server information

1. Db-related configuration: it is recommended that the memory is larger than the amount of data to be tested, so that the data is stored in the memory hot block, and the rest can be adjusted.
2. ShardingSphere-Proxy-related configuration: it is recommended to use a high-performance, multi-core CPU, and other configurations can be customized.
3. Disable swap partitions on all servers involved in the stress testing.

Database

```
[mysqld]
innodb_buffer_pool_size=${MORE_THAN_DATA_SIZE}
innodb-log-file-size=3000000000
innodb-log-files-in-group=5
innodb-flush-log-at-trx-commit=0
innodb-change-buffer-max-size=40
back_log=900
innodb_max_dirty_pages_pct=75
innodb_open_files=20480
innodb_buffer_pool_instances=8
innodb_page_cleaners=8
innodb_purge_threads=2
innodb_read_io_threads=8
innodb_write_io_threads=8
table_open_cache=102400
log_timestamps=system
thread_cache_size=16384
transaction_isolation=READ-COMMITTED

# Appropriate tuning can be considered to magnify the underlying DB performance, so
that the experiment doesn't subject to DB performance bottleneck.
```

Stress testing tool

Refer to sysbench' s GitHub

ShardingSphere-Proxy

bin/start.sh

```
-Xmx16g -Xms16g -Xmn8g  # Adjust JVM parameters
```

config.yaml

```
databaseName: sharding_db

dataSources:
  ds_0:
    url: jdbc:mysql://***.***.***.***:****/test?serverTimezone=UTC&useSSL=false #
Parameters can be adjusted appropriately
    username: test
    password:
```

```
    connectionTimeoutMilliseconds: 30000
    idleTimeoutMilliseconds: 60000
    maxLifetimeMilliseconds: 1800000
    maxPoolSize: 200 # The maximum ConnPool is set to ${the number of concurrencies in stress testing}, which is consistent with the number of concurrencies in stress testing to shield the impact of additional connections in the process of stress testing.
    minPoolSize: 200 # The minimum ConnPool is set to ${the number of concurrencies in stress testing}, which is consistent with the number of concurrencies in stress testing to shield the impact of connections initialization in the process of stress testing.

rules: []
```

Test phase

Environment setup

```
sysbench oltp_read_write --mysql-host=${DB_IP} --mysql-port=${DB_PORT} --mysql-user=${USER} --mysql-password=${PASSWD} --mysql-db=test --tables=10 --table-size=1000000 --report-interval=10 --time=100 --threads=200  cleanup
sysbench oltp_read_write --mysql-host=${DB_IP} --mysql-port=${DB_PORT} --mysql-user=${USER} --mysql-password=${PASSWD} --mysql-db=test --tables=10 --table-size=1000000 --report-interval=10 --time=100 --threads=200  prepare
```

Stress testing command

```
sysbench oltp_read_write --mysql-host=${DB/PROXY_IP} --mysql-port=${DB/PROXY_PORT} --mysql-user=${USER} --mysql-password=${PASSWD} --mysql-db=test --tables=10 --table-size=1000000 --report-interval=10 --time=100 --threads=200  run
```

Stress testing report analysis

```
sysbench 1.0.20 (using bundled LuaJIT 2.1.0-beta2)
Running the test with following options:
Number of threads: 200
Report intermediate results every 10 second(s)
Initializing random number generator from current time
Initializing worker threads...
Threads started!
# Report test results every 10 seconds, and the number of tps, reads per second, writes per second, and the total response time of more than 95th percentile.
[ 10s ] thds: 200 tps: 11161.70 qps: 223453.06 (r/w/o: 156451.76/44658.51/22342.80)
```

```
lat (ms,95%): 27.17 err/s: 0.00 reconn/s: 0.00
...
[ 120s ] thds: 200 tps: 11731.00 qps: 234638.36 (r/w/o: 164251.67/46924.69/23462.
00) lat (ms,95%): 24.38 err/s: 0.00 reconn/s: 0.00
SQL statistics:
    queries performed:
        read:                           19560590                       # number of
reads
        write:                          5588740                        # number of
writes
        other:                          27943700                       # number of
other operations (COMMIT etc.)
        total:                          27943700                       # the total
number
    transactions:                       1397185 (11638.59 per sec.)    # number of
transactions (per second)
    queries:                            27943700 (232771.76 per sec.)  # number of
statements executed (per second)
    ignored errors:                     0      (0.00 per sec.)         # number of
ignored errors (per second)
    reconnects:                         0      (0.00 per sec.)         # number of
reconnections (per second)

General statistics:
    total time:                         120.0463s                      # total
time
    total number of events:             1397185                        # toal
number of transactions

Latency (ms):
         min:                                    5.37                  # minimum
latency
         avg:                                   17.13                  # average
latency
         max:                                  109.75                  # maximum
latency
         95th percentile:                       24.83                  # average
response time of over 95th percentile.
         sum:                             23999546.19

Threads fairness:
    events (avg/stddev):          6985.9250/34.74                      # On
average, 6985.9250 events were completed per thread, and the standard deviation is
34.74
    execution time (avg/stddev):   119.9977/0.01                       # The
average time of each thread is 119.9977 seconds, and the standard deviation is 0.01
```

Noticeable features

1. CPU utilization ratio of the server where ShardingSphere-Proxy resides. It is better to make full use of CPU.
2. I/O of the server disk where the DB resides. The lower the physical read value is, the better.
3. Network IO of the server involved in the stress testing.

8.6.2 BenchmarkSQL ShardingSphere-Proxy Sharding Performance Test

Objective

BenchmarkSQL tool is used to test the sharding performance of ShardingSphere-Proxy.

Method

ShardingSphere-Proxy supports the TPC-C test through BenchmarkSQL 5.0. In addition to the content described in this document, BenchmarkSQL is operated according to the original document `HOW-TO-RUN.txt`.

Fine tuning to test tools

Unlike stand-alone database stress testing, distributed database solutions inevitably face trade-offs in functions. It is recommended to make the following adjustments when using BenchmarkSQL to carry out stress testing on ShardingSphere-Proxy.

Remove the foreign key and extraHistID

Modify `run/runDatabaseBuild.sh` in the BenchmarkSQL directory at line 17.

Before modification:

```
AFTER_LOAD="indexCreates foreignKeys extraHistID buildFinish"
```

After modification:

```
AFTER_LOAD="indexCreates buildFinish"
```

Stress testing environment or parameter recommendations

Note: None of the parameters mentioned in this section are absolute values and need to be adjusted based on actual test results.

It is recommended to run ShardingSphere using Java 17

ShardingSphere can be compiled using Java 8.

When using Java 17, maximize the ShardingSphere performance by default.

ShardingSphere data sharding recommendations

The data sharding of BenchmarkSQL can use the warehouse id in each table as the sharding key.

One of the tables `bmsql_item` has no warehouse id and has a fixed data volume of 100,000 rows: - You can take `i_id` as a sharding key. However, the same Proxy connection may hold connections to multiple different data sources at the same time. - Or you can give up sharding and store it in a single data source. But a data source may be under great pressure. - Or you may choose range-based sharding for `i_id`, such as 1-50000 for data source 0 and 50001-100000 for data source 1.

BenchmarkSQL has the following SQL involving multiple tables:

```sql
SELECT c_discount, c_last, c_credit, w_tax
FROM bmsql_customer
    JOIN bmsql_warehouse ON (w_id = c_w_id)
WHERE c_w_id = ? AND c_d_id = ? AND c_id = ?
```

```sql
SELECT o_id, o_entry_d, o_carrier_id
FROM bmsql_oorder
WHERE o_w_id = ? AND o_d_id = ? AND o_c_id = ?
  AND o_id = (
      SELECT max(o_id)
      FROM bmsql_oorder
      WHERE o_w_id = ? AND o_d_id = ? AND o_c_id = ?
      )
```

If the warehouse id is used as the sharding key, the tables involved in the above SQL can be configured as bindingTable:

```yaml
rules:
  - !SHARDING
    bindingTables:
      - bmsql_warehouse, bmsql_customer
      - bmsql_stock, bmsql_district, bmsql_order_line
```

For the data sharding configuration with warehouse id as the sharding key, refer to the appendix of this document.

PostgreSQL JDBC URL parameter recommendations

Adjust the JDBC URL in the configuration file used by BenchmarkSQL, that is, the value of the parameter name conn: - Adding the parameter `defaultRowFetchSize=50` may reduce the number of fetch for multi-row result sets. You need to increase or decrease the number according to actual test results. - Adding the parameter `reWriteBatchedInserts=true` may reduce the time spent on bulk inserts, such as preparing data or bulk inserts for the New Order business. Whether to enable the operation depends on actual test results.

props.pg file excerpt. It is suggested to change the parameter value of conn in line 3.

```
db=postgres
driver=org.postgresql.Driver
conn=jdbc:postgresql://localhost:5432/postgres?defaultRowFetchSize=50&
reWriteBatchedInserts=true
user=benchmarksql
password=PWbmsql
```

ShardingSphere-Proxy server.yaml parameter recommendations

The default value of `proxy-backend-query-fetch-size` is -1. Changing it to about `50` can minimize the number of fetch for multi-row result sets.

The default value of `proxy-frontend-executor-size` is CPU * 2 and can be reduced to about CPU * 0.5 based on actual test results. If NUMA is involved, set this parameter to the number of physical cores per CPU based on actual test results.

`server.yaml` file excerpt:

```
props:
  proxy-backend-query-fetch-size: 50
  # proxy-frontend-executor-size: 32 # 4*32C aarch64
  # proxy-frontend-executor-size: 12 # 2*12C24T x86
```

Appendix

BenchmarkSQL data sharding reference configuration

Adjust pool size according to the actual stress testing process.

```
databaseName: bmsql_sharding
dataSources:
  ds_0:
    url: jdbc:postgresql://db0.ip:5432/bmsql
    username: postgres
    password: postgres
    connectionTimeoutMilliseconds: 3000
```

```yaml
    idleTimeoutMilliseconds: 60000
    maxLifetimeMilliseconds: 1800000
    maxPoolSize: 1000
    minPoolSize: 1000
  ds_1:
    url: jdbc:postgresql://db1.ip:5432/bmsql
    username: postgres
    password: postgres
    connectionTimeoutMilliseconds: 3000
    idleTimeoutMilliseconds: 60000
    maxLifetimeMilliseconds: 1800000
    maxPoolSize: 1000
    minPoolSize: 1000
  ds_2:
    url: jdbc:postgresql://db2.ip:5432/bmsql
    username: postgres
    password: postgres
    connectionTimeoutMilliseconds: 3000
    idleTimeoutMilliseconds: 60000
    maxLifetimeMilliseconds: 1800000
    maxPoolSize: 1000
    minPoolSize: 1000
  ds_3:
    url: jdbc:postgresql://db3.ip:5432/bmsql
    username: postgres
    password: postgres
    connectionTimeoutMilliseconds: 3000
    idleTimeoutMilliseconds: 60000
    maxLifetimeMilliseconds: 1800000
    maxPoolSize: 1000
    minPoolSize: 1000

rules:
  - !SHARDING
    bindingTables:
      - bmsql_warehouse, bmsql_customer
      - bmsql_stock, bmsql_district, bmsql_order_line
    defaultDatabaseStrategy:
      none:
    defaultTableStrategy:
      none:
    keyGenerators:
      snowflake:
        type: SNOWFLAKE
    tables:
      bmsql_config:
        actualDataNodes: ds_0.bmsql_config
```

```yaml
    bmsql_warehouse:
      actualDataNodes: ds_${0..3}.bmsql_warehouse
      databaseStrategy:
        standard:
          shardingColumn: w_id
          shardingAlgorithmName: mod_4

    bmsql_district:
      actualDataNodes: ds_${0..3}.bmsql_district
      databaseStrategy:
        standard:
          shardingColumn: d_w_id
          shardingAlgorithmName: mod_4

    bmsql_customer:
      actualDataNodes: ds_${0..3}.bmsql_customer
      databaseStrategy:
        standard:
          shardingColumn: c_w_id
          shardingAlgorithmName: mod_4

    bmsql_item:
      actualDataNodes: ds_${0..3}.bmsql_item
      databaseStrategy:
        standard:
          shardingColumn: i_id
          shardingAlgorithmName: mod_4

    bmsql_history:
      actualDataNodes: ds_${0..3}.bmsql_history
      databaseStrategy:
        standard:
          shardingColumn: h_w_id
          shardingAlgorithmName: mod_4

    bmsql_oorder:
      actualDataNodes: ds_${0..3}.bmsql_oorder
      databaseStrategy:
        standard:
          shardingColumn: o_w_id
          shardingAlgorithmName: mod_4

    bmsql_stock:
      actualDataNodes: ds_${0..3}.bmsql_stock
      databaseStrategy:
        standard:
          shardingColumn: s_w_id
          shardingAlgorithmName: mod_4
```

```
      bmsql_new_order:
        actualDataNodes: ds_${0..3}.bmsql_new_order
        databaseStrategy:
          standard:
            shardingColumn: no_w_id
            shardingAlgorithmName: mod_4

      bmsql_order_line:
        actualDataNodes: ds_${0..3}.bmsql_order_line
        databaseStrategy:
          standard:
            shardingColumn: ol_w_id
            shardingAlgorithmName: mod_4

    shardingAlgorithms:
      mod_4:
        type: MOD
        props:
          sharding-count: 4
```

BenchmarkSQL 5.0 PostgreSQL statement list

Create tables

```
create table bmsql_config (
  cfg_name    varchar(30) primary key,
  cfg_value   varchar(50)
);

create table bmsql_warehouse (
  w_id        integer   not null,
  w_ytd       decimal(12,2),
  w_tax       decimal(4,4),
  w_name      varchar(10),
  w_street_1  varchar(20),
  w_street_2  varchar(20),
  w_city      varchar(20),
  w_state     char(2),
  w_zip       char(9)
);

create table bmsql_district (
  d_w_id       integer       not null,
  d_id         integer       not null,
  d_ytd        decimal(12,2),
```

```
  d_tax         decimal(4,4),
  d_next_o_id   integer,
  d_name        varchar(10),
  d_street_1    varchar(20),
  d_street_2    varchar(20),
  d_city        varchar(20),
  d_state       char(2),
  d_zip         char(9)
);

create table bmsql_customer (
  c_w_id          integer        not null,
  c_d_id          integer        not null,
  c_id            integer        not null,
  c_discount      decimal(4,4),
  c_credit        char(2),
  c_last          varchar(16),
  c_first         varchar(16),
  c_credit_lim    decimal(12,2),
  c_balance       decimal(12,2),
  c_ytd_payment   decimal(12,2),
  c_payment_cnt   integer,
  c_delivery_cnt  integer,
  c_street_1      varchar(20),
  c_street_2      varchar(20),
  c_city          varchar(20),
  c_state         char(2),
  c_zip           char(9),
  c_phone         char(16),
  c_since         timestamp,
  c_middle        char(2),
  c_data          varchar(500)
);

create sequence bmsql_hist_id_seq;

create table bmsql_history (
  hist_id   integer,
  h_c_id    integer,
  h_c_d_id  integer,
  h_c_w_id  integer,
  h_d_id    integer,
  h_w_id    integer,
  h_date    timestamp,
  h_amount  decimal(6,2),
  h_data    varchar(24)
);
```

```
create table bmsql_new_order (
  no_w_id  integer   not null,
  no_d_id  integer   not null,
  no_o_id  integer   not null
);

create table bmsql_oorder (
  o_w_id       integer      not null,
  o_d_id       integer      not null,
  o_id         integer      not null,
  o_c_id       integer,
  o_carrier_id integer,
  o_ol_cnt     integer,
  o_all_local  integer,
  o_entry_d    timestamp
);

create table bmsql_order_line (
  ol_w_id         integer   not null,
  ol_d_id         integer   not null,
  ol_o_id         integer   not null,
  ol_number       integer   not null,
  ol_i_id         integer   not null,
  ol_delivery_d   timestamp,
  ol_amount       decimal(6,2),
  ol_supply_w_id  integer,
  ol_quantity     integer,
  ol_dist_info    char(24)
);

create table bmsql_item (
  i_id     integer      not null,
  i_name   varchar(24),
  i_price  decimal(5,2),
  i_data   varchar(50),
  i_im_id  integer
);

create table bmsql_stock (
  s_w_id       integer       not null,
  s_i_id       integer       not null,
  s_quantity   integer,
  s_ytd        integer,
  s_order_cnt  integer,
  s_remote_cnt integer,
  s_data       varchar(50),
  s_dist_01    char(24),
  s_dist_02    char(24),
```

```
  s_dist_03    char(24),
  s_dist_04    char(24),
  s_dist_05    char(24),
  s_dist_06    char(24),
  s_dist_07    char(24),
  s_dist_08    char(24),
  s_dist_09    char(24),
  s_dist_10    char(24)
);
```

Create indexes

```
alter table bmsql_warehouse add constraint bmsql_warehouse_pkey
  primary key (w_id);

alter table bmsql_district add constraint bmsql_district_pkey
  primary key (d_w_id, d_id);

alter table bmsql_customer add constraint bmsql_customer_pkey
  primary key (c_w_id, c_d_id, c_id);

create index bmsql_customer_idx1
  on  bmsql_customer (c_w_id, c_d_id, c_last, c_first);

alter table bmsql_oorder add constraint bmsql_oorder_pkey
  primary key (o_w_id, o_d_id, o_id);

create unique index bmsql_oorder_idx1
  on  bmsql_oorder (o_w_id, o_d_id, o_carrier_id, o_id);

alter table bmsql_new_order add constraint bmsql_new_order_pkey
  primary key (no_w_id, no_d_id, no_o_id);

alter table bmsql_order_line add constraint bmsql_order_line_pkey
  primary key (ol_w_id, ol_d_id, ol_o_id, ol_number);

alter table bmsql_stock add constraint bmsql_stock_pkey
  primary key (s_w_id, s_i_id);

alter table bmsql_item add constraint bmsql_item_pkey
  primary key (i_id);
```

New Order business

stmtNewOrderSelectWhseCust

```
UPDATE bmsql_district
    SET d_next_o_id = d_next_o_id + 1
    WHERE d_w_id = ? AND d_id = ?
```

stmtNewOrderSelectDist

```
SELECT d_tax, d_next_o_id
    FROM bmsql_district
    WHERE d_w_id = ? AND d_id = ?
    FOR UPDATE
```

stmtNewOrderUpdateDist

```
UPDATE bmsql_district
    SET d_next_o_id = d_next_o_id + 1
    WHERE d_w_id = ? AND d_id = ?
```

stmtNewOrderInsertOrder

```
INSERT INTO bmsql_oorder (
    o_id, o_d_id, o_w_id, o_c_id, o_entry_d,
    o_ol_cnt, o_all_local)
VALUES (?, ?, ?, ?, ?, ?, ?)
```

stmtNewOrderInsertNewOrder

```
INSERT INTO bmsql_new_order (
    no_o_id, no_d_id, no_w_id)
VALUES (?, ?, ?)
```

stmtNewOrderSelectStock

```
SELECT s_quantity, s_data,
       s_dist_01, s_dist_02, s_dist_03, s_dist_04,
       s_dist_05, s_dist_06, s_dist_07, s_dist_08,
       s_dist_09, s_dist_10
    FROM bmsql_stock
    WHERE s_w_id = ? AND s_i_id = ?
    FOR UPDATE
```

stmtNewOrderSelectItem

```
SELECT i_price, i_name, i_data
    FROM bmsql_item
    WHERE i_id = ?
```

stmtNewOrderUpdateStock

```
UPDATE bmsql_stock
    SET s_quantity = ?, s_ytd = s_ytd + ?,
        s_order_cnt = s_order_cnt + 1,
        s_remote_cnt = s_remote_cnt + ?
    WHERE s_w_id = ? AND s_i_id = ?
```

stmtNewOrderInsertOrderLine

```
INSERT INTO bmsql_order_line (
    ol_o_id, ol_d_id, ol_w_id, ol_number,
    ol_i_id, ol_supply_w_id, ol_quantity,
    ol_amount, ol_dist_info)
VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?)
```

Payment business

stmtPaymentSelectWarehouse

```
SELECT w_name, w_street_1, w_street_2, w_city,
       w_state, w_zip
    FROM bmsql_warehouse
    WHERE w_id = ?
```

stmtPaymentSelectDistrict

```
SELECT d_name, d_street_1, d_street_2, d_city,
       d_state, d_zip
    FROM bmsql_district
    WHERE d_w_id = ? AND d_id = ?
```

stmtPaymentSelectCustomerListByLast

```
SELECT c_id
    FROM bmsql_customer
    WHERE c_w_id = ? AND c_d_id = ? AND c_last = ?
    ORDER BY c_first
```

stmtPaymentSelectCustomer

```
SELECT c_first, c_middle, c_last, c_street_1, c_street_2,
       c_city, c_state, c_zip, c_phone, c_since, c_credit,
       c_credit_lim, c_discount, c_balance
    FROM bmsql_customer
    WHERE c_w_id = ? AND c_d_id = ? AND c_id = ?
    FOR UPDATE
```

stmtPaymentSelectCustomerData

```sql
SELECT c_data
    FROM bmsql_customer
    WHERE c_w_id = ? AND c_d_id = ? AND c_id = ?
```

stmtPaymentUpdateWarehouse

```sql
UPDATE bmsql_warehouse
    SET w_ytd = w_ytd + ?
    WHERE w_id = ?
```

stmtPaymentUpdateDistrict

```sql
UPDATE bmsql_district
    SET d_ytd = d_ytd + ?
    WHERE d_w_id = ? AND d_id = ?
```

stmtPaymentUpdateCustomer

```sql
UPDATE bmsql_customer
    SET c_balance = c_balance - ?,
        c_ytd_payment = c_ytd_payment + ?,
        c_payment_cnt = c_payment_cnt + 1
    WHERE c_w_id = ? AND c_d_id = ? AND c_id = ?
```

stmtPaymentUpdateCustomerWithData

```sql
UPDATE bmsql_customer
    SET c_balance = c_balance - ?,
        c_ytd_payment = c_ytd_payment + ?,
        c_payment_cnt = c_payment_cnt + 1,
        c_data = ?
    WHERE c_w_id = ? AND c_d_id = ? AND c_id = ?
```

stmtPaymentInsertHistory

```sql
INSERT INTO bmsql_history (
    h_c_id, h_c_d_id, h_c_w_id, h_d_id, h_w_id,
    h_date, h_amount, h_data)
VALUES (?, ?, ?, ?, ?, ?, ?, ?)
```

Order Status business

stmtOrderStatusSelectCustomerListByLast

```
SELECT c_id
    FROM bmsql_customer
    WHERE c_w_id = ? AND c_d_id = ? AND c_last = ?
    ORDER BY c_first
```

stmtOrderStatusSelectCustomer

```
SELECT c_first, c_middle, c_last, c_balance
    FROM bmsql_customer
    WHERE c_w_id = ? AND c_d_id = ? AND c_id = ?
```

stmtOrderStatusSelectLastOrder

```
SELECT o_id, o_entry_d, o_carrier_id
    FROM bmsql_oorder
    WHERE o_w_id = ? AND o_d_id = ? AND o_c_id = ?
      AND o_id = (
          SELECT max(o_id)
              FROM bmsql_oorder
              WHERE o_w_id = ? AND o_d_id = ? AND o_c_id = ?
          )
```

stmtOrderStatusSelectOrderLine

```
SELECT ol_i_id, ol_supply_w_id, ol_quantity,
       ol_amount, ol_delivery_d
    FROM bmsql_order_line
    WHERE ol_w_id = ? AND ol_d_id = ? AND ol_o_id = ?
    ORDER BY ol_w_id, ol_d_id, ol_o_id, ol_number
```

Stock level business

stmtStockLevelSelectLow

```
SELECT count(*) AS low_stock FROM (
    SELECT s_w_id, s_i_id, s_quantity
        FROM bmsql_stock
        WHERE s_w_id = ? AND s_quantity < ? AND s_i_id IN (
            SELECT ol_i_id
                FROM bmsql_district
                JOIN bmsql_order_line ON ol_w_id = d_w_id
                 AND ol_d_id = d_id
                 AND ol_o_id >= d_next_o_id - 20
                 AND ol_o_id < d_next_o_id
```

```
                WHERE d_w_id = ? AND d_id = ?
        )
    ) AS L
```

Delivery BG business

stmtDeliveryBGSelectOldestNewOrder

```
SELECT no_o_id
    FROM bmsql_new_order
    WHERE no_w_id = ? AND no_d_id = ?
    ORDER BY no_o_id ASC
```

stmtDeliveryBGDeleteOldestNewOrder

```
DELETE FROM bmsql_new_order
    WHERE no_w_id = ? AND no_d_id = ? AND no_o_id = ?
```

stmtDeliveryBGSelectOrder

```
SELECT o_c_id
    FROM bmsql_oorder
    WHERE o_w_id = ? AND o_d_id = ? AND o_id = ?
```

stmtDeliveryBGUpdateOrder

```
UPDATE bmsql_oorder
    SET o_carrier_id = ?
    WHERE o_w_id = ? AND o_d_id = ? AND o_id = ?
```

stmtDeliveryBGSelectSumOLAmount

```
SELECT sum(ol_amount) AS sum_ol_amount
    FROM bmsql_order_line
    WHERE ol_w_id = ? AND ol_d_id = ? AND ol_o_id = ?
```

stmtDeliveryBGUpdateOrderLine

```
UPDATE bmsql_order_line
SET ol_delivery_d = ?
WHERE ol_w_id = ? AND ol_d_id = ? AND ol_o_id = ?
```

stmtDeliveryBGUpdateCustomer

```
UPDATE bmsql_customer
    SET c_balance = c_balance + ?,
        c_delivery_cnt = c_delivery_cnt + 1
    WHERE c_w_id = ? AND c_d_id = ? AND c_id = ?
```

8.7 Module Test

Provides test engine with each complex modules.

8.7.1 SQL Parser Test

Prepare Data

Not like Integration test, SQL parse test does not need a specific database environment, just define the sql to parse, and the assert data:

SQL Data

As mentioned `sql-case-id` in Integration test, test-case-id could be shared in different module to test, and the file is at `shardingsphere-sql-parser/shardingsphere-sql-parser-test/src/main/resources/sql/supported/${SQL-TYPE}/*.xml`

Assert Data

The assert data is at `shardingsphere-sql-parser/shardingsphere-sql-parser-test/src/main/resources/case/${SQL-TYPE}/*.xml` in that xml file, it could assert against the table name, token or sql condition and so on. For example:

```
<parser-result-sets>
    <parser-result sql-case-id="insert_with_multiple_values">
        <tables>
            <table name="t_order" />
        </tables>
        <tokens>
            <table-token start-index="12" table-name="t_order" length="7" />
        </tokens>
        <sharding-conditions>
            <and-condition>
                <condition column-name="order_id" table-name="t_order" operator=
"EQUAL">
                    <value literal="1" type="int" />
                </condition>
                <condition column-name="user_id" table-name="t_order" operator=
"EQUAL">
                    <value literal="1" type="int" />
                </condition>
            </and-condition>
            <and-condition>
                <condition column-name="order_id" table-name="t_order" operator=
"EQUAL">
```

```
                    <value literal="2" type="int" />
                </condition>
                <condition column-name="user_id" table-name="t_order" operator=
"EQUAL">
                    <value literal="2" type="int" />
                </condition>
            </and-condition>
        </sharding-conditions>
    </parser-result>
</parser-result-sets>
```

When these configs are ready, launch the test engine in `shardingsphere-sql-parser/shardingsphere-sql-parser-test` to test SQL parse.

8.7.2 SQL Rewrite Test

Target

Facing logic databases and tables cannot be executed directly in actual databases. SQL rewrite is used to rewrite logic SQL into rightly executable ones in actual databases, including two parts, correctness rewrite and optimization rewrite. rewrite tests are for these targets.

Test

The rewrite tests are in the test folder under `sharding-core/sharding-core-rewrite` . Followings are the main part for rewrite tests:

- test engine
- environment configuration
- assert data

Test engine is the entrance of rewrite tests, just like other test engines, through Junit Parameterized, read every and each data in the xml file under the target test type in `test\resources`, and then assert by the engine one by one

Environment configuration is the yaml file under test type under `test\resources\yaml`. The configuration file contains dataSources, shardingRule, encryptRule and other info. for example:

```
dataSources:
  db: !!com.zaxxer.hikari.HikariDataSource
    driverClassName: org.h2.Driver
    jdbcUrl: jdbc:h2:mem:db;DB_CLOSE_DELAY=-1;DATABASE_TO_UPPER=false;MODE=MYSQL
    username: sa
    password:

## sharding Rules
```

```yaml
rules:
- !SHARDING
  tables:
    t_account:
      actualDataNodes: db.t_account_${0..1}
      tableStrategy:
        standard:
          shardingColumn: account_id
          shardingAlgorithmName: account_table_inline
      keyGenerateStrategy:
        column: account_id
        keyGeneratorName: snowflake
    t_account_detail:
      actualDataNodes: db.t_account_detail_${0..1}
      tableStrategy:
        standard:
          shardingColumn: order_id
          shardingAlgorithmName: account_detail_table_inline
  bindingTables:
    - t_account, t_account_detail
  shardingAlgorithms:
    account_table_inline:
      type: INLINE
      props:
        algorithm-expression: t_account_${account_id % 2}
    account_detail_table_inline:
      type: INLINE
      props:
        algorithm-expression: t_account_detail_${account_id % 2}
  keyGenerators:
    snowflake:
      type: SNOWFLAKE
```

Assert data are in the xml under test type in `test\resources`. In the xml file, `yaml-rule` means the environment configuration file path, `input` contains the target SQL and parameters, `output` contains the expected SQL and parameters. The `db-type` described the type for SQL parse, default is `SQL92`. For example:

```xml
<rewrite-assertions yaml-rule="yaml/sharding/sharding-rule.yaml">
    <!-- to change SQL parse type, change db-type -->
    <rewrite-assertion id="create_index_for_mysql" db-type="MySQL">
        <input sql="CREATE INDEX index_name ON t_account ('status')" />
        <output sql="CREATE INDEX index_name ON t_account_0 ('status')" />
        <output sql="CREATE INDEX index_name ON t_account_1 ('status')" />
    </rewrite-assertion>
</rewrite-assertions>
```

After set up the assert data and environment configuration, rewrite test engine will assert the corresponding SQL without any Java code modification.

8.8 Scaling Integration Test

8.8.1 Objectives

Verify the functional correctness of data migration and dependency modules.

8.8.2 Test environment

Currently, Native and Docker environments are supported. 1. The Native environment runs directly in the test environment provided by the developer, and users need to start ShardingSphere-Proxy and the corresponding database instance by themselves, which is suitable for debugging scenarios. 2. The Docker environment is run by Maven, which is suitable for cloud compilation environment and ShardingSphere-Proxy testing scenarios, such as GitHub Action.

Currently, you can use MySQL, PostgreSQL and openGuass databases.

8.8.3 User guide

Module path: `shardingsphere-test/shardingsphere-integration-test/shardingsphere-integration-test-scaling`.

Environment setup

`${DOCKER-IMAGE}` refers to the name of a Docker mirror, such as `mysql:8`. `${DATABASE-TYPE}` refers to database types. Directory: `src/test/resources/env-it-env.properties`: the startup parameters of integration testing. - `${DATABASE-TYPE}/server.yaml`: ShardingSphere-Proxy configuration file corresponding to the database. - `${DATABASE-TYPE}/initdb.sql`: The database initializes SQL. - `${DATABASE-TYPE}/*.cnf,*.conf`: Files ending with cnf or conf are database configuration files for Docker mount. - `common/command.xml`: The DistSQL used in the test. - `scenario/`: Store SQL in the test scenarios.

Test case

Currently, all the test cases are directly inherited from `BaseExtraSQLITCase` and indirectly inherited from `BaseITCase`. - `BaseITCase`: Provide generic methods for sub-class. - `BaseExtraSQLITCase`: Provide table creation and CRUD statement execution methods.

Test case example: MySQLGeneralScalingIT. Functions included: - Database-level migration (all tables). - Table-level migration (any number). - Verify migration data consistency. - Stop writing is supported during data migration. - Support restart during data migration. - Support integer primary keys during data migration. - Support string primary keys during data migration. - A non-administrator account can be used to migrate data.

Running the test case

All property values of `it-env.properties` can be introduced by the Maven command line `-D`, and its priority is higher than that of the configuration file.

Native environment setup

The user starts ShardingSphere-Proxy locally in advance, along with dependent configuration centers (such as ZooKeeper) and databases. The port required for ShardingSphere-Proxy is 3307. Take MySQL as an example, `it-env.properties` can be configured as follows:

```
scaling.it.env.type=NATIVE
scaling.it.native.database=mysql
scaling.it.native.mysql.username=root
scaling.it.native.mysql.password=root
scaling.it.native.mysql.port=3306
```

Find the appropriate test case and start it with Junit under the IDE.

Docker environment setup

Step 1: Package mirror.

```
./mvnw -B clean install -am -pl shardingsphere-test/shardingsphere-integration-
test/shardingsphere-integration-test-scaling -Pit.env.docker -DskipTests
```

Running the above command will build a Docker mirror apache/shardingsphere-proxy-test:latest used for integration testing. The mirror sets the port for remote debugging and the default port is 3308. If only the test code is modified, you can reuse the existing test mirror without rebuilding it.

If you need to adjust Docker mirror startup parameters, you can modify the configuration of the ShardingSphereProxyDockerContainer file.

The output log of ShardingSphere-Proxy has the prefix Scaling-Proxy.

Use Maven to run the test cases. Take MySQL as an example:

```
./mvnw -nsu -B install -f shardingsphere-test/shardingsphere-integration-test/
shardingsphere-integration-test-scaling/pom.xml -Dscaling.it.env.type=DOCKER -
Dscaling.it.docker.mysql.version=${image-name}
```

You can also use IDE to run test cases. `it-env.properties` can be configured as follows:

```
scaling.it.env.type=DOCKER
scaling.it.docker.mysql.version=mysql:5.7
```

9 Reference

This chapter contains a section of technical implementation with Apache ShardingSphere, which provide the reference with users and developers.

9.1 Database Compatibility

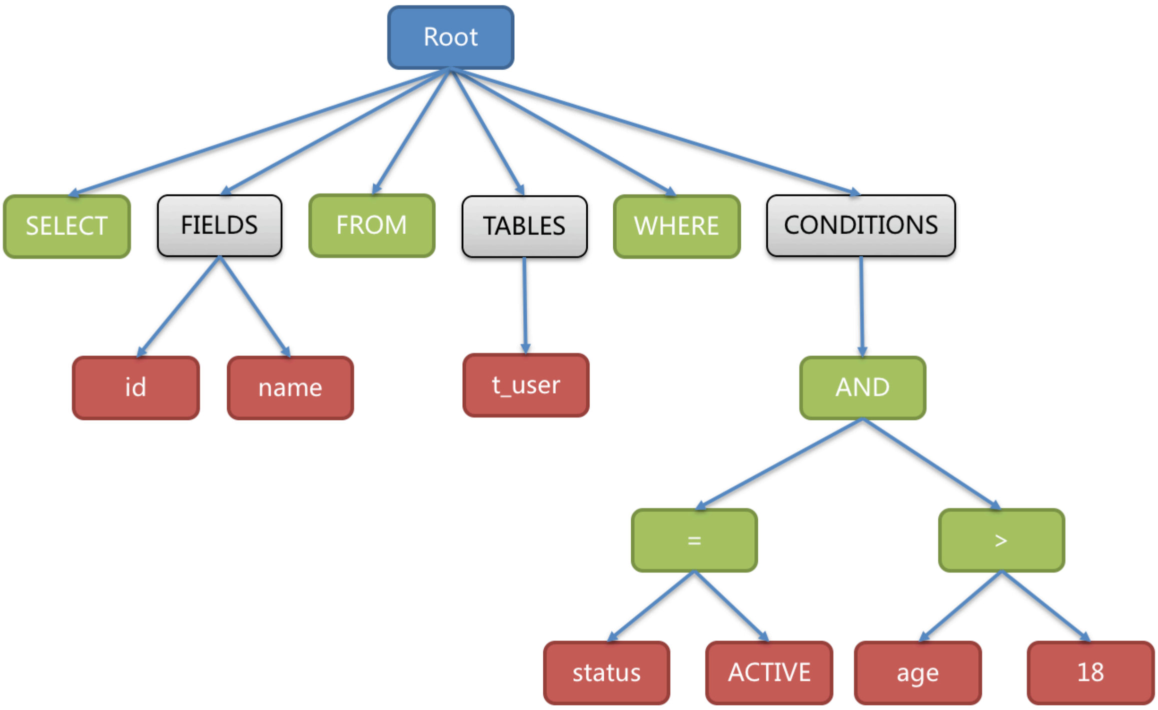


Figure1: The core of SQL parsing: Abstract Syntax Tree (AST)

- SQL compatibility

SQL is the standard language for users to communicate with databases. The SQL parsing engine is responsible for parsing SQL strings into abstract syntax trees so that Apache ShardingSphere can understand and implement its incremental function. ShardingSphere currently supports MySQL, PostgreSQL, SQLServer, Oracle, openGauss, and SQL dialects conforming to the SQL92 standard. Due to the complexity of SQL syntax, a few SQL are not supported for now.

- Database protocol compatibility

Apache ShardingSphere currently implements MySQL and PostgreSQL protocols according to different data protocols.

- Supported features

Apache ShardingSphere provides distributed collaboration capabilities for databases. At the same time, it abstracts some database features to the upper layer for unified management, so as to facilitate users.

Therefore, native SQL will not deliver the features provided uniformly to the database, and a message will be displayed indicating that the operation is not supported. Users can replace it with methods provided by ShardingSphere.

9.2 Database Gateway

Apache ShardingSphere provides the ability for SQL dialect translation to achieve automatic conversion between database dialects. For example, users can use MySQL client to connect ShardingSphere and send SQL based on MySQL dialect. ShardingSphere can automatically identify user protocol and storage node type, automatically complete SQL dialect conversion, and access heterogeneous storage nodes such as PostgreSQL.

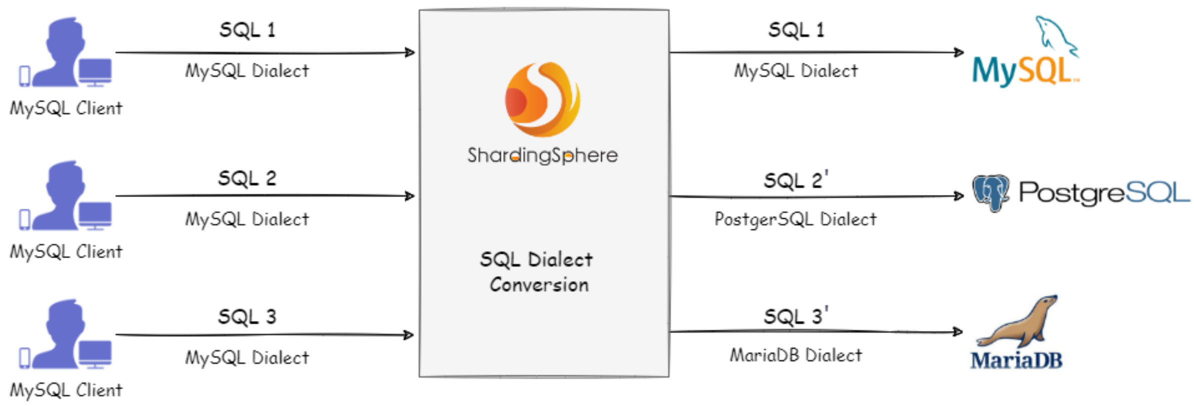


Figure2: Gateway

9.3 Management

9.3.1 Data Structure in Registry Center

Under defined namespace, `rules`, `props` and `metadata` nodes persist in YAML, modifying nodes can dynamically refresh configurations. `nodes` node persist the runtime node of database access object, to distinguish different database access instances.

```
namespace
   ├──rules                                      # Global rule configuration
   ├──props                                      # Properties configuration
   ├──metadata                                   # Metadata configuration
   ├     ├──${databaseName}                      # Logic database name
   ├     ├     ├──schemas                        # Schema list
   ├     ├     ├     ├──${schemaName}            # Logic schema name
   ├     ├     ├     ├     ├──tables            # Table configuration
   ├     ├     ├     ├     ├     ├──${tableName}
   ├     ├     ├     ├     ├     ├──...
   ├     ├     ├     ├──...
   ├     ├     ├──versions                       # Metadata version list
   ├     ├     ├     ├──${versionNumber}         # Metadata version
   ├     ├     ├     ├     ├──dataSources        # Data source configuration
   ├     ├     ├     ├     ├──rules              # Rule configuration
   ├     ├     ├     ├──...
   ├     ├     ├──active_version                 # Active metadata version
   ├     ├──...
   ├──nodes
   ├    ├──compute_nodes
   ├    ├     ├──online
   ├    ├     ├     ├──proxy
   ├    ├     ├     ├     ├──UUID                # Proxy instance identifier
   ├    ├     ├     ├     ├──....
   ├    ├     ├     ├──jdbc
   ├    ├     ├     ├     ├──UUID                # JDBC instance identifier
   ├    ├     ├     ├     ├──....
   ├    ├     ├──status
   ├    ├     ├     ├──UUID
   ├    ├     ├     ├──....
   ├    ├     ├──worker_id
   ├    ├     ├     ├──UUID
   ├    ├     ├     ├──....
   ├    ├     ├──process_trigger
   ├    ├     ├     ├──process_list_id:UUID
   ├    ├     ├     ├──....
   ├    ├     ├──labels
   ├    ├     ├     ├──UUID
   ├    ├     ├     ├──....
   ├    ├──storage_nodes
   ├    ├     ├──${databaseName.groupName.ds}
   ├    ├     ├──${databaseName.groupName.ds}
```

/rules

global rule configurations, including configure the username and password for ShardingSphere-Proxy.

```
- !AUTHORITY
users:
  - root@%:root
  - sharding@127.0.0.1:sharding
provider:
  type: ALL_PERMITTED
```

/props

Properties configuration. Please refer to Configuration Manual for more details.

```
kernel-executor-size: 20
sql-show: true
```

/metadata/$databaseName/versions/${versionNumber}/dataSources

A collection of multiple database connection pools, whose properties (e.g. DBCP, C3P0, Druid and HikariCP) are configured by users themselves.

```
ds_0:
  initializationFailTimeout: 1
  validationTimeout: 5000
  maxLifetime: 1800000
  leakDetectionThreshold: 0
  minimumIdle: 1
  password: root
  idleTimeout: 60000
  jdbcUrl: jdbc:mysql://127.0.0.1:3306/ds_0?serverTimezone=UTC&useSSL=false
  dataSourceClassName: com.zaxxer.hikari.HikariDataSource
  maximumPoolSize: 50
  connectionTimeout: 30000
  username: root
  poolName: HikariPool-1
ds_1:
  initializationFailTimeout: 1
  validationTimeout: 5000
  maxLifetime: 1800000
  leakDetectionThreshold: 0
  minimumIdle: 1
  password: root
  idleTimeout: 60000
  jdbcUrl: jdbc:mysql://127.0.0.1:3306/ds_1?serverTimezone=UTC&useSSL=false
  dataSourceClassName: com.zaxxer.hikari.HikariDataSource
```

```
  maximumPoolSize: 50
  connectionTimeout: 30000
  username: root
  poolName: HikariPool-2
```

/metadata/$databaseName/versions/${versionNumber}/rules

Rule configurations, including sharding, readwrite-splitting, data encryption, shadow DB configurations.

```
- !SHARDING
  xxx

- !READWRITE_SPLITTING
  xxx

- !ENCRYPT
  xxx
```

/metadata/$databaseName/schemas/${schemaName}/tables

Use separate node storage for each table, dynamic modification of metadata content is not supported currently.

```
name: t_order                           # Table name
columns:                                # Columns
  id:                                   # Column name
    caseSensitive: false
    dataType: 0
    generated: false
    name: id
    primaryKey: trues
  order_id:
    caseSensitive: false
    dataType: 0
    generated: false
    name: order_id
    primaryKey: false
indexs:                                 # Index
  t_user_order_id_index:                # Index name
    name: t_user_order_id_index
```

/nodes/compute_nodes

It includes running instance information of database access object, with sub-nodes as the identifiers of currently running instance, which is automatically generated at each startup using UUID. Those identifiers are temporary nodes, which are registered when instances are on-line and cleared when instances are off-line. The registry center monitors the change of those nodes to govern the database access of running instances and other things.

/nodes/storage_nodes

It is able to orchestrate replica database, delete or disable data dynamically.

9.4 Sharding

The figure below shows how sharding works. According to whether query and optimization are needed, it can be divided into the Simple Push Down process and SQL Federation execution engine process. Simple Push Down process consists of `SQL parser => SQL binder => SQL router => SQL rewriter => SQL executor => result merger`, mainly used to deal with SQL execution in standard sharding scenarios. SQL Federation execution engine consists of `SQL parser => SQL binder => logical optimization => physical optimization => data fetcher => operator calculation`. This process performs logical optimization and physical optimization internally, during which the standard kernel procedure is adopted to route, rewrite, execute and merge the optimized logical SQL.

9.4.1 SQL Parser

It is divided into the lexical parser and syntactic parser. SQL is first split into indivisible words through a lexical parser.

The syntactic parser is then used to analyze SQL and ultimately extract the parsing context, which can include tables, options, ordering items, grouping items, aggregation functions, pagination information, query conditions, and placeholders that may be modified.

9.4.2 SQL Route

The sharding strategy configured by the user is matched according to the parsing context and the routing path is generated. Currently, sharding router and broadcast router are supported.

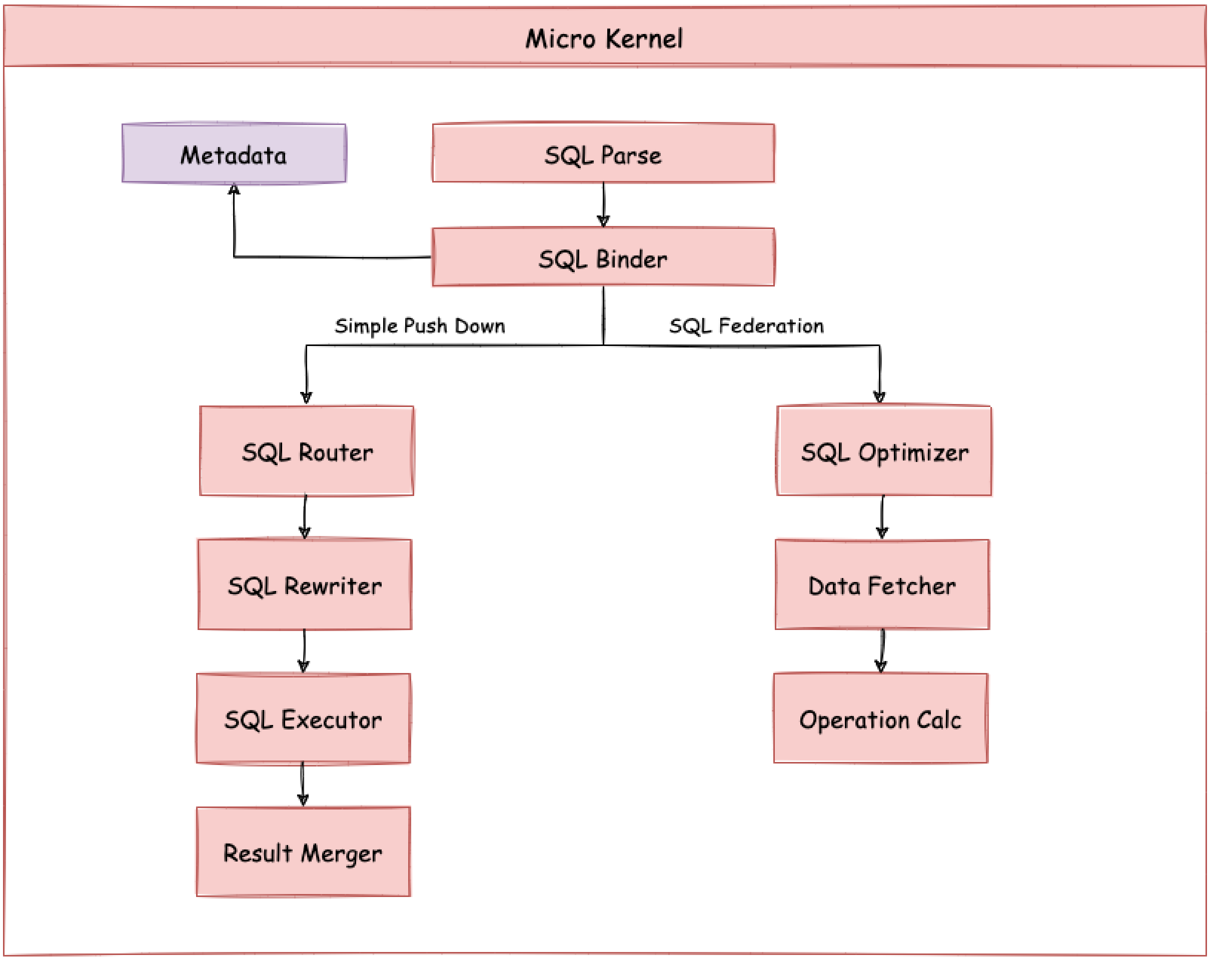


Figure3: Sharding Architecture Diagram

9.4.3 SQL Rewrite

Rewrite SQL into statements that can be executed correctly in a real database. SQL rewriting is divided into rewriting for correctness and rewriting for optimization.

9.4.4 SQL Execution

It executes asynchronously through a multithreaded executor.

9.4.5 Result Merger

It merges multiple execution result sets to achieve output through the unified JDBC interface. The result merger includes the stream merger, memory merger and appended merger using decorator mode.

9.4.6 Query Optimization

Supported by the experimental Federation Execution Engine, it optimizes complex queries such as associated queries and sub-queries and supports distributed queries across multiple database instances. It internally optimizes query plans using relational algebra to query results through optimal plans.

9.4.7 Parse Engine

SQL is relatively simple compared with other programming languages, but it's still a complete programming language. Therefore, there's no essential difference between parsing SQL syntax and parsing other languages (such as Java, C and Go, etc.).

Abstract Syntax Tree

The parsing process is divided into lexical parsing and syntactic parsing. The lexical parser is used to split SQL into indivisible atomic symbols called Tokens.

Tokens are classified into keywords, expressions, literals, and operators based on the dictionaries provided by different database dialects. The syntactic parser is then used to convert the output of the lexical parser into an abstract syntax tree.

For example:

```
SELECT id, name FROM t_user WHERE status = 'ACTIVE' AND age > 18
```

After the above SQL is parsed, its AST (Abstract Syntax Tree) is as follows:

The tokens for keywords in the AST are green, while the tokens for variables are red, and gray ones indicate that further splitting is required.

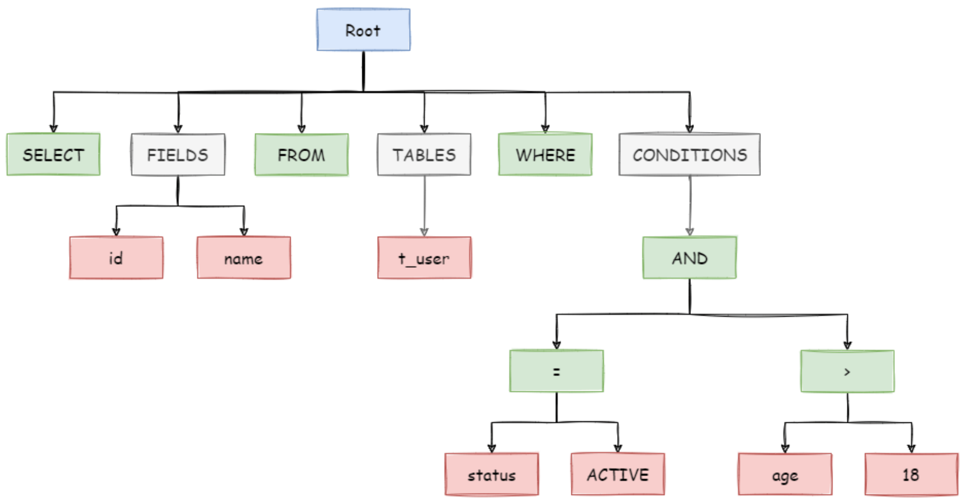


Figure4: SQL AST

Finally, the domain model is traversed through the abstract syntax tree by visitor; the context required for sharding is extracted through the domain model (SQLStatement); and then, mark locations that may need rewriting.

The parsing context for sharding includes select items, table, sharding condition, auto-increment primary key, and Order By, Group By, and pagination information (Limit, Rownum, Top). The SQL parsing process is irreversible.

Each Token is parsed in the original SQL order, providing high performance. Taking the similarities and differences of SQL dialects of various databases into consideration, the SQL dialect dictionary of various databases is provided in the parsing module.

SQL Parser Engine

Iteration

SQL parsing is the core of sharding solutions, and its performance and compatibility are the most important indicators. ShardingSphere's SQL parser has undergone three iterations and upgrades.

To achieve high performance and fast implementation, the first generation of SQL parsers used Druid prior to V1.4.x. In practical tests, its performance far exceeds that of other parsers.

The second generation of SQL parsers started from V1.5.x. ShardingSphere uses a completely self-developed SQL parsing engine. Owing to different purposes, ShardingSphere does not need to convert SQL into a complete abstract syntax tree, nor does it require a second traversal through the accessor pattern. It uses a half-parsing method to extract only the context required by data sharding, thus further improving the performance and compatibility of SQL parsing.

The third generation of SQL parsers, starting with V3.0.x, attempts to use ANTLR as a generator of SQL parsing engines and uses Visit to obtain SQL statements from the AST. Since V5.0.x, the architecture of the parsing engine has been restructured and adjusted. Moreover, the AST obtained from the first parsing is stored in the cache so that the parsing results of the same SQL can be directly obtained next time to improve parsing efficiency. Therefore, it is recommended that you use PreparedStatement, a SQL-precompiled method, to improve performance.

Features

- Independent SQL parsing engine
- The syntax rules can be easily expanded and modified (using `ANTLR`)
- Support multiple dialects

Database	Status
MySQL	perfect supported
PostgreSQL	perfect supported
SQLServer	supported
Oracle	supported
SQL92	supported
openGauss	supported

API Usage

- Introducing Maven dependency

```
<dependency>
    <groupId>org.apache.shardingsphere</groupId>
    <artifactId>shardingsphere-sql-parser-engine</artifactId>
    <version>${project.version}</version>
</dependency>
<!-- According to the needs, introduce the parsing module of the specified dialect
(take MySQL as an example), you can add all the supported dialects, or just what
you need -->
<dependency>
    <groupId>org.apache.shardingsphere</groupId>
    <artifactId>shardingsphere-sql-parser-mysql</artifactId>
    <version>${project.version}</version>
</dependency>
```

- Obtain AST

```
CacheOption cacheOption = new CacheOption(128, 1024L);
SQLParserEngine parserEngine = new SQLParserEngine(sql, cacheOption);
ParseASTNode parseASTNode = parserEngine.parse(sql, useCache);
```

- Obtain SQLStatement

```
CacheOption cacheOption = new CacheOption(128, 1024L);
SQLParserEngine parserEngine = new SQLParserEngine(sql, cacheOption);
ParseASTNode parseASTNode = parserEngine.parse(sql, useCache);
SQLVisitorEngine sqlVisitorEngine = new SQLVisitorEngine(sql, "STATEMENT",
useCache, new Properties());
SQLStatement sqlStatement = sqlVisitorEngine.visit(parseASTNode);
```

- SQL Formatting

```
ParseASTNode parseASTNode = parserEngine.parse(sql, useCache);
SQLVisitorEngine sqlVisitorEngine = new SQLVisitorEngine(sql, "FORMAT", useCache,
new Properties());
String result = sqlVisitorEngine.visit(parseASTNode);
```

Example:

Original SQL	Formatted SQL
select a+1 as b, name n from table1 join table2 where id=1 and name= 'lu' ;	SELECT a + 1 AS b, name nFROM table1 JOIN table2WHERE id = 1 and name = 'lu' ;
select id, name, age, sex, ss, yy from table1 where id=1;	SELECT id , name , age , sex , ss , yy FROM table1WHERE id = 1;
select id, name, age, count(*) as n, (select id, name, age, sex from table2 where id=2) as sid, yyyy from table1 where id=1;	SELECT id , name , age , COUNT(*) AS n, (SELECT id , name , age , sex FROM table2 WHERE id = 2) AS sid, yyyy FROM table1WHERE id = 1;
select id, name, age, sex, ss, yy from table1 where id=1 and name=1 and a=1 and b=2 and c=4 and d=3;	SELECT id , name , age , sex , ss , yy FROM table1WHERE id = 1 and name = 1 and a = 1 and b = 2 and c = 4 and d = 3;
ALTER TABLE t_order ADD column4 DATE, ADD column5 DATETIME, engine ss max_rows 10,min_rows 2, ADD column6 TIMESTAMP, ADD column7 TIME;	ALTER TABLE t_order ADD column4 DATE, ADD column5 DATETIME, ENGINE ss MAX_ROWS 10, MIN_ROWS 2, ADD column6 TIMESTAMP, ADD column7 TIME
CREATE TABLE IF NOT EXISTS runoob_tbl(runoob_id INT UNSIGNED AUTO_INCREMENT,runoob_title VARCHAR(100) NOT NULL,runoob_author VARCHAR(40) NOT NULL,runoob_test NATIONAL CHAR(40),submission_date DATE,PRIMARY KEY (runoob_id))ENGINE=InnoDB DEFAULT CHARSET=utf8;	CREATE TABLE IF NOT EXISTS runoob_tbl (runoob_id INT UNSIGNED AUTO_INCREMENT, runoob_title VARCHAR(100) NOT NULL, runoob_author VARCHAR(40) NOT NULL , runoob_test NATIONAL CHAR(40), submission_date DATE, PRIMARY KEY (runoob_id)) ENGINE = InnoDB DEFAULT CHARSET = utf8;
INSERT INTO t_order_item(order_id, user_id, status, creation_date) values (1, 1, 'insert' , '2017-08-08'), (2, 2, 'insert' , '2017-08-08') ON DUPLICATE KEY UPDATE status = 'init' ;	INSERT INTO t_order_item (order_id , user_id , status , creatio n_date)VALUES (1, 1, 'insert' , '2017-08-08'), (2, 2, 'insert' , '2017-08-08')ON DUPLICATE KEY UPDATE status = 'init' ;
INSERT INTO t_order SET order_id = 1, user_id = 1, status = convert(to_base64(aes_encrypt(1, 'key')) USING utf8) ON DUPLICATE KEY UPDATE status = VALUES(status);	INSERT INTO t_order SET order_id = 1, user_id = 1, status = CONVERT(to_ base64(aes_encrypt(1 , 'key')) USING utf8)ON DUPLICATE KEY UPDATE status = VALUES(status);
INSERT INTO t_order (order_id, user_id, status) SELECT order_id, user_id, status FROM t_order WHERE order_id = 1;	INSERT INTO t_order (order_id , user_id , status) SELECT order_id , user_id , status FROM t_orderWHERE order_id = 1;

9.4.8 Route Engine

Sharding strategies for databases and tables are matched based on the parsing context, and routing paths are generated. SQL with shard keys can be divided into the single-shard router (the shard key operator is equal), multi-shard router (the shard key operator is IN), and range router (the shard key operator is BETWEEN). SQL that does not carry shard keys adopts broadcast routing.

Sharding strategies can usually be configured either by the built-in database or by the user. The built-in database scheme is relatively simple, and the built-in sharding strategy can be roughly divided into mantissa modulo, hash, range, label, time, etc.

The sharding strategies configured by the user are more flexible. You can customize the compound sharding strategy based on the user's requirements. If it is used with automatic data migration, users do not need to work on the sharding strategies.

Sharding and data balancing can be automatically achieved by the middle layer of the database, and distributed databases can achieve elastic scalability. In the planning of ShardingSphere, the elastic scaling function will be available at V4.x.

Sharding Route

The scenario that is routed based on shard keys is divided into three types: direct route, standard route, and Cartesian route.

Direct Route

The requirement for direct route is relatively harsh. It needs to be sharded by Hint (using HintAPI to specify routes to databases and tables), and it can avoid SQL parsing and subsequent result merge on the premise of having database shards but not table shards.

Therefore, it is the most compatible one and can execute any SQL in complex scenarios including sub-queries and custom functions. The direct route can also be used when shard keys are not in SQL. For example, set the key for database sharding to 3,

```
hintManager.setDatabaseShardingValue(3);
```

If the routing algorithm is `value % 2`, when a logical database `t_order` corresponds to two physical databases`t_order_0` and `t_order_1`, the SQL will be executed on `t_order_1` after routing. The following is a sample code using the API.

```
String sql = "SELECT * FROM t_order";
try (
        HintManager hintManager = HintManager.getInstance();
        Connection conn = dataSource.getConnection();
        PreparedStatement pstmt = conn.prepareStatement(sql)) {
    hintManager.setDatabaseShardingValue(3);
    try (ResultSet rs = pstmt.executeQuery()) {
        while (rs.next()) {
```

```
            //...
        }
    }
}
```

Standard Route

The standard route is the most recommended sharding method, and it is applicable to SQL that does not contain an associated query or only contains the associated query between binding tables.

When the sharding operator is equal, the routing result will fall into a single database (table). When the sharding operator is BETWEEN or IN, the routing result will not necessarily fall into a unique database (table).

Therefore, logical SQL may eventually be split into multiple real SQL to be executed. For example, if the data sharding is carried out according to the odd and even numbers of order_id, the SQL for a single table query is as follows:

```sql
SELECT * FROM t_order WHERE order_id IN (1, 2);
```

Then the routing result should be:

```sql
SELECT * FROM t_order_0 WHERE order_id IN (1, 2);
SELECT * FROM t_order_1 WHERE order_id IN (1, 2);
```

An associated query for a binding table is as complex as a single table query and they have the same performance. For example, if the SQL of an associated query that contains binding tables is as follows:

```sql
SELECT * FROM t_order o JOIN t_order_item i ON o.order_id=i.order_id  WHERE order_
id IN (1, 2);
```

Then the routing result should be:

```sql
SELECT * FROM t_order_0 o JOIN t_order_item_0 i ON o.order_id=i.order_id  WHERE
order_id IN (1, 2);
SELECT * FROM t_order_1 o JOIN t_order_item_1 i ON o.order_id=i.order_id  WHERE
order_id IN (1, 2);
```

As you can see, the number of SQL splits is consistent with that of a single table.

Cartesian Route

The Cartesian route is the most complex one because it cannot locate sharding rules according to the relationship between binding tables, so associated queries between unbound tables need to be disassembled and executed as cartesian product combinations. If the SQL in the previous example was not configured with binding table relationships, the routing result would be:

```
SELECT * FROM t_order_0 o JOIN t_order_item_0 i ON o.order_id=i.order_id  WHERE
order_id IN (1, 2);
SELECT * FROM t_order_0 o JOIN t_order_item_1 i ON o.order_id=i.order_id  WHERE
order_id IN (1, 2);
SELECT * FROM t_order_1 o JOIN t_order_item_0 i ON o.order_id=i.order_id  WHERE
order_id IN (1, 2);
SELECT * FROM t_order_1 o JOIN t_order_item_1 i ON o.order_id=i.order_id  WHERE
order_id IN (1, 2);
```

The Cartesian route query has low performance, so think carefully when you use it.

Broadcast Route

For SQL that does not carry shard keys, broadcast routes are used. According to the SQL type, it can be further divided into five types: full database and table route, full database route, full instance route, unicast route, and block route.

Full database and table route

The full database table route is used to handle operations on all real tables related to its logical tables in the database, including DQL and DML without shard keys, as well as DDL, etc. For example:

```
SELECT * FROM t_order WHERE good_prority IN (1, 10);
```

All tables in all databases will be traversed, matching logical tables and real table names one by one. The table that can be matched will be executed. The routing result would be:

```
SELECT * FROM t_order_0 WHERE good_prority IN (1, 10);
SELECT * FROM t_order_1 WHERE good_prority IN (1, 10);
SELECT * FROM t_order_2 WHERE good_prority IN (1, 10);
SELECT * FROM t_order_3 WHERE good_prority IN (1, 10);
```

Full database route

The full database route is used to handle operations on the database, including database management commands of type SET for database settings and transaction control statements such as TCL.

In this case, all real database matching names are traversed based on the logical database name, and the command is executed in the real database. For example:

```
SET autocommit=0;
```

If the command is executed in `t_order`, `t_order` which has two real databases, it is actually executed on both `t_order_0` and `t_order_1`.

Full instance route

Full instance route is used for DCL operations, and authorized statements are used for database instances.

No matter how many schemas are contained in an instance, each database instance is executed only once. For example:

```
CREATE USER customer@127.0.0.1 identified BY '123';
```

This command will be executed on all real database instances to ensure that users can access each instance.

Unicast Route

The unicast route is used to obtain the information of a real table. It only needs to obtain data from any real table in any database. For example:

```
DESCRIBE t_order;
```

`t_order_0` and `t_order_1`, the two real tables of `t_order`, have the same description structure, so this command is executed only once on any real table.

Block Route

Block route is used to block SQL operations on the database, for example:

```
USE order_db;
```

This command will not be executed in a real database because ShardingSphere uses the logical Schema and there is no need to send the Schema shift command to the database.

The overall structure of the routing engine is as follows.

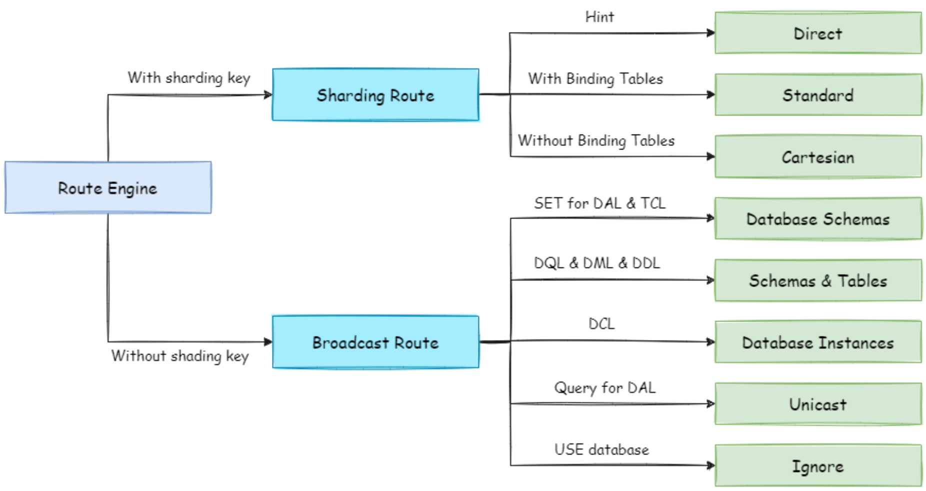


Figure5: Route Engine

9.4.9 Rewrite Engine

SQL written by engineers for logical databases and tables cannot be directly executed in real databases.

SQL rewriting is used to rewrite logical SQL into SQL that can be executed correctly in real databases. It includes rewriting for correctness and rewriting for optimization.

Rewriting for Correctness

In a scenario with table shards, you need to rewrite the logical table name in the table shards configuration to the real table name obtained after routing.

Only database shards do not require rewriting table names. Additionally, it also includes column derivation and pagination information correction.

Identifier Rewriting

The identifiers that need to be overwritten include table names, index names, and Schema names.

Rewriting table names is the process of finding the location of the logical table in the original SQL and rewriting it into a real table.

Table name rewriting is a typical scenario that requires SQL parsing. For example, if logical SQL is:

```
SELECT order_id FROM t_order WHERE order_id=1;
```

Assume that the SQL is configured with the shard key `order_id` and `order_id=1`, it will be routed to shard table 1. Then the rewritten SQL should be:

```
SELECT order_id FROM t_order_1 WHERE order_id=1;
```

In the simplest SQL scenario, it doesn't seem to matter whether or not the SQL is parsed into an abstract syntax tree.

SQL can be rewritten correctly only by finding and replacing strings. However, it is impossible to achieve the same effect in the following scenarios.

```
SELECT order_id FROM t_order WHERE order_id=1 AND remarks=' t_order xxx';
```

The correct rewritten SQL would be:

```
SELECT order_id FROM t_order_1 WHERE order_id=1 AND remarks=' t_order xxx';
```

Instead of:

```
SELECT order_id FROM t_order_1 WHERE order_id=1 AND remarks=' t_order_1 xxx';
```

Because there may be characters similar to the table name, you cannot rewrite SQL simply by replacing strings.

Let's look at a more complex scenario:

```
SELECT t_order.order_id FROM t_order WHERE t_order.order_id=1 AND remarks=' t_order
xxx';
```

The above SQL uses the table name as an identifier of the field, so it needs to be modified when SQL is rewritten:

```
SELECT t_order_1.order_id FROM t_order_1 WHERE t_order_1.order_id=1 AND remarks='
t_order xxx';
```

If a table alias is defined in SQL, the alias does not need to be modified, even if it is the same as the table name. For example:

```
SELECT t_order.order_id FROM t_order AS t_order WHERE t_order.order_id=1 AND
remarks=' t_order xxx';
```

Rewriting the table name is enough for SQL rewriting.

```
SELECT t_order.order_id FROM t_order_1 AS t_order WHERE t_order.order_id=1 AND
remarks=' t_order xxx';
```

The index name is another identifier that can be rewritten. In some databases (such as MySQL and SQLServer), indexes are created in the dimension of tables.

Indexes in different tables can have the same name. In other databases (such as PostgreSQL and Oracle), indexes are created in the dimension of databases, and even indexes on different tables should have unique names.

In ShardingSphere, schemas are managed in the same way as tables. Logical Schemas are used to manage a set of data sources.

Therefore, ShardingSphere needs to replace the logical Schema written by the user in SQL with the real database Schema.

Currently, ShardingSphere does not support the use of Schema in DQL and DML statements. It only supports the use of Schema in database management statements. For example:

```
SHOW COLUMNS FROM t_order FROM order_ds;
```

Schema rewriting refers to the rewriting of a logical Schema using unicast routing to a correct and real Schema that is randomly found.

Column Derivation

There are two cases that need to complement columns in a query statement. In the first case, ShardingSphere needs to get the data during the result merge, but the data is not returned by the queried SQL.

In this case, it mainly applies to GROUP BY and ORDER BY. When merging the results, you need to group and order the field items according to `GROUP BY` and `ORDER BY`, but if the original SQL does not contain grouping or ordering items in the selections, you need to rewrite the original SQL. Let' s look at a scenario where the original SQL has the required information for result merge.

```
SELECT order_id, user_id FROM t_order ORDER BY user_id;
```

Since `user_id` is used for sorting, the data of `user_id` needs to be retrieved in the result merge. And the above SQL can obtain the data of `user_id`, so there is no need to add columns.

If the selection does not contain the columns required to merge the results, you need to fill the columns, as in the following SQL:

```
SELECT order_id FROM t_order ORDER BY user_id;
```

Since the original SQL does not contain the `user_id` required in the result merge, you need to fill in and rewrite the SQL. Then SQL would be:

```
SELECT order_id, user_id AS ORDER_BY_DERIVED_0 FROM t_order ORDER BY user_id;
```

It should be noted that only missing columns are complemented instead of all columns. And SQL that contains * in the SELECT statement will also selectively complement columns based on the metadata information of the table. Here is a relatively complex column derivation scenario of SQL:

```
SELECT o.* FROM t_order o, t_order_item i WHERE o.order_id=i.order_id ORDER BY
user_id, order_item_id;
```

We assume that only the table `t_order_item` contains the column `order_item_id`. According to the metadata information of the table, when the result is merged, the `user_id` in the ordering items exists on the table `t_order`, so there is no need to add columns. `order_item_id` is not in `t_order`, so column derivation is required. Then SQL would become:

```
SELECT o.*, order_item_id AS ORDER_BY_DERIVED_0 FROM t_order o, t_order_item i
WHERE o.order_id=i.order_id ORDER BY user_id, order_item_id;
```

The second case of column derivation is the use of AVG aggregate functions. In distributed scenarios, using (avg1 + avg2 + avg3)/3 to calculate the average is incorrect and should be rewritten as (sum1 + sum2 + sum3) /(count1 + count2 + count3). In this case, rewriting the SQL containing AVG to SUM and COUNT is required, and recalculating the average when the results are merged. For example:

```
SELECT AVG(price) FROM t_order WHERE user_id=1;
```

The above SQL should be rewritten as:

```
SELECT COUNT(price) AS AVG_DERIVED_COUNT_0, SUM(price) AS AVG_DERIVED_SUM_0 FROM t_
order WHERE user_id=1;
```

Then you can calculate the average correctly by merging the results.

The last type of column derivation is the one that does not need to write the primary key field if the database auto-increment primary key is used during executing an INSERT SQL statement. However, the auto-increment primary key of the database cannot meet the unique primary key in distributed scenarios. Therefore, ShardingSphere provides the generation strategy of the distributed auto-increment primary key. Users can replace the existing auto-increment primary key transparently with the distributed auto-increment primary key without changing the existing code through column derivation. The generation strategy for distributed auto-increment primary keys is described below, and here only SQL rewriting is illustrated. For example, if the primary key of table `t_order` is `order_id`, the original SQL would be:

```
INSERT INTO t_order (`field1`, `field2`) VALUES (10, 1);
```

As you can see, the above SQL does not contain the auto-increment primary key, which requires the database itself to fill. After ShardingSphere is configured with the auto-increment primary key, SQL will be rewritten as:

```sql
INSERT INTO t_order (`field1`, `field2`, order_id) VALUES (10, 1, xxxxx);
```

The rewritten SQL will add column names of the primary key and auto-increment primary key values generated automatically at the end of the INSERT FIELD and INSERT VALUE. The xxxxx in the above SQL represents the auto-increment primary key value generated automatically.

If the INSERT SQL does not contain the column name of the table, ShardingSphere can also compare the number of parameters and the number of columns in the table meta information and automatically generate auto-increment primary keys. For example, the original SQL is:

```sql
INSERT INTO t_order VALUES (10, 1);
```

The rewritten SQL will simply add the auto-increment primary key in the column order in which the primary key locates:

```sql
INSERT INTO t_order VALUES (xxxxx, 10, 1);
```

If you use placeholders to write SQL, you only need to rewrite the parameter list, not the SQL itself.

Pagination Correction

The scenario of acquiring pagination data from multiple databases is different from that of one single database. If every 10 pieces of data are taken as one page, the user wants to take the second page of data. It is not correct to acquire `LIMIT 10, 10` under sharding situations, or take out the first 10 pieces of data according to sorting conditions after merging. For example, if SQL is:

```sql
SELECT score FROM t_score ORDER BY score DESC LIMIT 1, 2;
```

The following picture shows the pagination execution results without SQL rewriting.

As shown in the picture, if you want to acquire the second and the third piece of data sorted by score in both tables, and they are supposed to be `95` and `90`.

Since executed SQL can only acquire the second and the third piece of data from each table, i.e., `90` and `80` from `t_score_0`, `85` and `75` from `t_score_1`. When merging results, it can only merge from `90`, `80`, `85` and `75` already acquired, so the right result cannot be acquired anyway.

The right way is to rewrite pagination conditions as `LIMIT 0, 3`, take out all the data from the first two pages and calculate the right data based on sorting conditions. The following picture shows the execution results of pagination after SQL rewrite.

The latter the offset position is, the lower the efficiency of using `LIMIT` pagination will be. There are many ways to avoid using `LIMIT` as pagination method, such as constructing a secondary index to the number of line records and line offsets or using the end ID of the last pagination data as a condition for the next query.

When revising pagination information, if the users use the placeholder to write SQL, they only need to rewrite the parameter list rather than SQL itself.

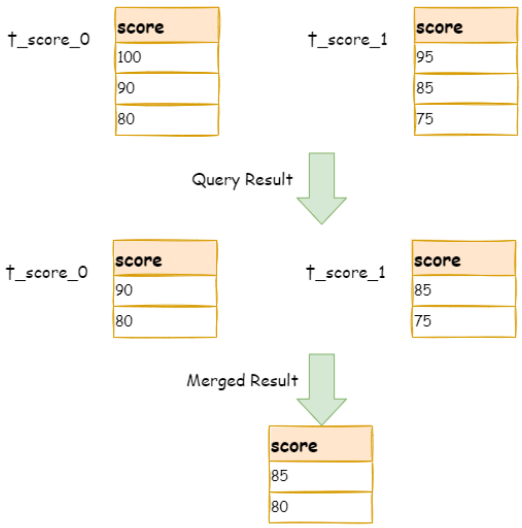


Figure6: Pagination without rewrite

Batch Split

When using bulk inserted SQL, if the inserted data crosses shards, the SQL needs to be rewritten to prevent excess data from being written to the database.

The insertion operation differs from the query operation in that the query statement does not affect the data even if it uses the shard key that does not exist in the current shard. In contrast, insertion operations must remove excess shard keys. For example, see the following SQL:

```
INSERT INTO t_order (order_id, xxx) VALUES (1, 'xxx'), (2, 'xxx'), (3, 'xxx');
```

If the database is still divided into two parts according to the odd and even number of order_id, this SQL will be executed after its table name is revised. Then, both shards will be written with the same record.

Though only the data that satisfies sharding conditions can be retrieved from the query statement, it is not reasonable for the schema to have excessive data. So SQL should be rewritten as:

```
INSERT INTO t_order_0 (order_id, xxx) VALUES (2, 'xxx');
INSERT INTO t_order_1 (order_id, xxx) VALUES (1, 'xxx'), (3, 'xxx');
```

IN query is similar to batch insertion, but IN operation will not lead to wrong data query result. Through rewriting IN query, the query performance can be further improved. See the following SQL:

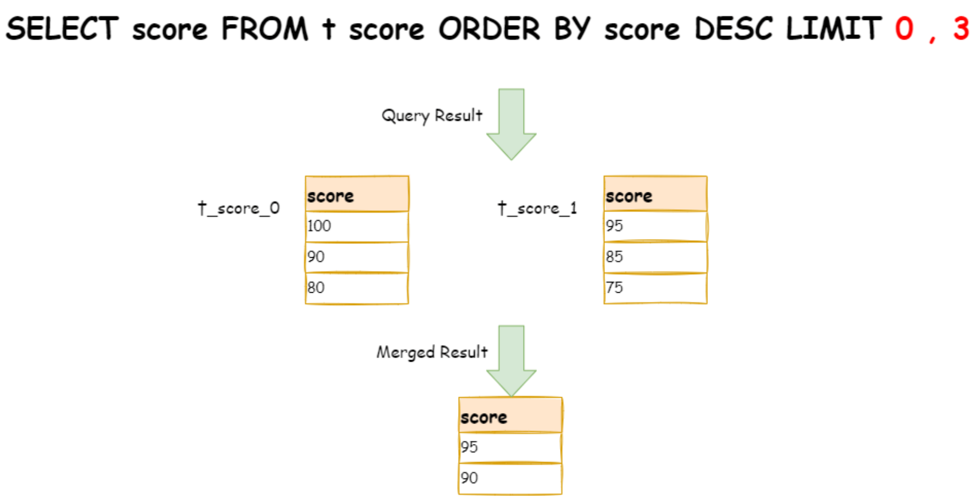


Figure7: Pagination with rewrite

```
SELECT * FROM t_order WHERE order_id IN (1, 2, 3);
```

The SQL is rewritten as:

```
SELECT * FROM t_order_0 WHERE order_id IN (2);
SELECT * FROM t_order_1 WHERE order_id IN (1, 3);
```

The query performance will be further improved. For now, ShardingSphere has not realized this rewrite strategy, so the current rewrite result is:

```
SELECT * FROM t_order_0 WHERE order_id IN (1, 2, 3);
SELECT * FROM t_order_1 WHERE order_id IN (1, 2, 3);
```

Though the execution result of SQL is right, it did not achieve the highest query efficiency.

Rewriting for Optimization

Its purpose is to effectively improve performance without influencing the correctness of the query. It can be divided into single node optimization and stream merger optimization.

Single Node Optimization

It refers to the optimization that stops the SQL rewrite from the route to the single node. After acquiring one route result, if it is routed to a single data node, there is no need to involve result merger, as well as rewrites such as column derivation and pagination information correction.

In particular, there is no need to read from the first piece of information, which reduces the pressure on the database to a large extent and saves meaningless consumption of the network bandwidth.

Stream Merger Optimization

It only adds `ORDER BY` and ordering items and sorting orders identical with grouping items to SQL that contains `GROUP BY`. And it is used to transfer memory merger to stream merger. Stream merger and memory merger will be explained in detail in the result merger section.

The overall structure of the rewrite engine is shown in the following picture.

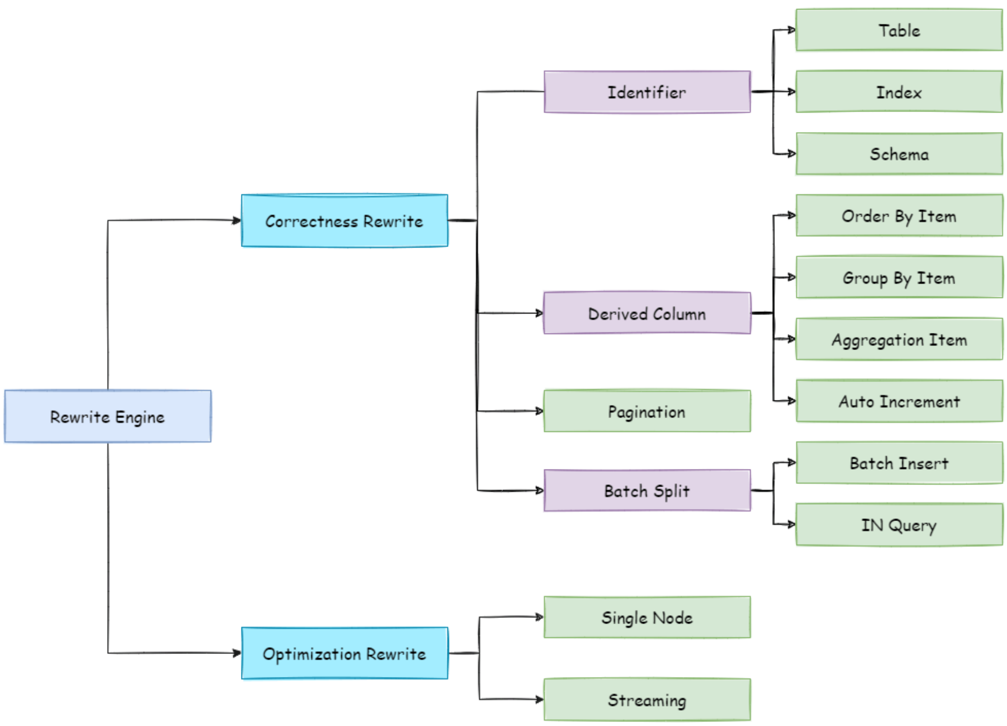


Figure8: Rewrite Engine

9.4.10 Execute Engine

ShardingSphere uses an automated execution engine to safely and efficiently send the real SQL, which has been routed and rewritten, to the underlying data source for execution.

It does not simply send SQL directly to the data source for execution via JDBC, nor are execution requests placed directly into a thread pool for concurrent execution.

It focuses more on the creation of a balanced data source connection, the consumption generated by the memory usage, and the maximum utilization of the concurrency. The objective of the execution engine is to automatically balance resource control with execution efficiency.

Connection Mode

From the perspective of resource control, the connection number a business can make to the database should be limited. It can effectively prevent certain business operations from occupying excessive resources, exhausting database connection resources, and influencing the normal access of other businesses.

Especially when one database instance contains many sub-tables, a logical SQL that does not contain any shard key will produce a large number of real SQLs that fall into different tables in one database. If each real SQL takes an independent connection, a query will undoubtedly take up excessive resources.

From the perspective of execution efficiency, maintaining an independent database connection for each shard query can make more effective use of multi-thread to improve execution efficiency.

Creating a separate thread for each database connection allows I/O consumption to be processed in parallel. Maintaining a separate database connection for each shard also prevents premature loading of query result data into memory.

It is enough for independent database connections to maintain result set quotation and cursor position, and move the cursor when acquiring corresponding data.

Merging the result set by moving down its cursor is called the stream merger. It does not need to load all the query results into the memory, which can effectively save memory resources effectively and reduce the frequency of garbage collection.

If each shard query cannot be guaranteed to have an independent database connection, the current query result set needs to be loaded into memory before reusing the database connection to obtain the query result set of the next shard table. Therefore, though the stream merger can be used, it will also degenerate into the memory merger in this scenario.

On the one hand, we need to control and protect database connection resources; on the other hand, it is important to save middleware memory resources by adopting a better merging mode. How to deal with the relationship between the two is a problem that the ShardingSphere execution engine needs to solve. Specifically, if an SQL is sharded through the ShardingSphere, it needs to operate on 200 tables under a database instance. So, should we choose to create 200 connections in parallel, or one connection in sequence? How to choose between efficiency and resource control? For the above scenario, ShardingSphere provides a solution. It introduces the concept of Connection Mode, which is divided into MEMORY_STRICTLY and CONNECTION_STRICTLY.

MEMORY_STRICTLY Mode

The prerequisite to using this mode is that ShardingSphere does not restrict the connection number of one operation. If the actual executed SQL needs to operate 200 tables in some database instance, it will create a new database connection for each table and deal with them concurrently through multi-thread to maximize the execution efficiency. When SQL meets the conditions, stream merger is preferred to avoid memory overflow or frequent garbage recycling.

CONNECTION_STRICTLY Mode

The prerequisite to using this mode is that ShardingSphere strictly restricts the connection consumption number of one operation. If the SQL to be executed needs to operate 200 tables in a database instance, it will create one database connection and operate them serially. If shards exist in different databases, it will still adopt multi-thread operations for different databases, but with only one database connection being created for each operation in each database. It prevents the problem of consuming too many database connections for one request. The mode chooses memory merger all the time.

The MEMORY_STRICTLY mode applies to OLAP operation and can increase the system throughput by removing database connection restrictions. It is also applicable to OLTP operation, which usually has shard keys and can be routed to a single shard. So it is a wise choice to control database connections strictly to make sure that database resources in an online system can be used by more applications.

Automatic Execution Engine

ShardingSphere initially leaves the decision of which mode to use up to the users and they can choose to use MEMORY_STRICTLY mode or CONNECTION_STRICTLY mode according to their actual business scenarios.

This solution gives users the right to choose, who must understand the pros and cons of the two modes and make a choice based on the requirements of the business scenarios. No doubt, it is not the best solution as it increases users' learning and use costs.

This dichotomy solution, which leaves the switching of the two modes to static initialization, lacks flexibility. In practical scenarios, the routing result varies with SQL and placeholder indexes. This means that some operations may need to use memory merger, while others may prefer stream merger. Connection modes should not be set by the user before ShardingSphere is started, but should be determined dynamically based on the SQL and placeholder indexes scenarios.

In order to reduce the usage cost for users and achieve a dynamic connection mode, ShardingSphere has extracted the concept of the automatic execution engine to eliminate the connection mode concept internally. The user does not need to know what the MEMORY_STRICTLY mode and CONNECTION_STRICTLY mode are, but the execution engine automatically selects the best execution scheme according to the current scenario.

The automatic execution engine chooses the connection mode based on each SQL operation. For each SQL request, the automatic execution engine will do real-time calculations and evaluations according to its route result and execute the appropriate connection mode automatically to strike the optimal balance between resource control and efficiency. For the automatic execution engine, users only need to configure `maxConnectionSizePerQuery`, which represents the maximum connection number allowed by each database for one query.

The execution engine is divided into two phases: preparation and execution.

Preparation Phrase

As indicated by its name, this phrase is used to prepare the data to be executed. It can be divided into two steps: result set grouping and unit creation.

Result set grouping is the key to realizing the internal connection model concept. According to the configuration items of `maxConnectionSizePerQuery`, the execution engine will choose an appropriate connection mode based on the current route result.

Detailed steps are as follow:

1. Group SQL route results according to data source names.
2. As we can see in the following formula, users can acquire the SQL route result set to be executed by each database instance within the `maxConnectionSizePerQuery` permission range and calculate the optimal connection mode of this request.

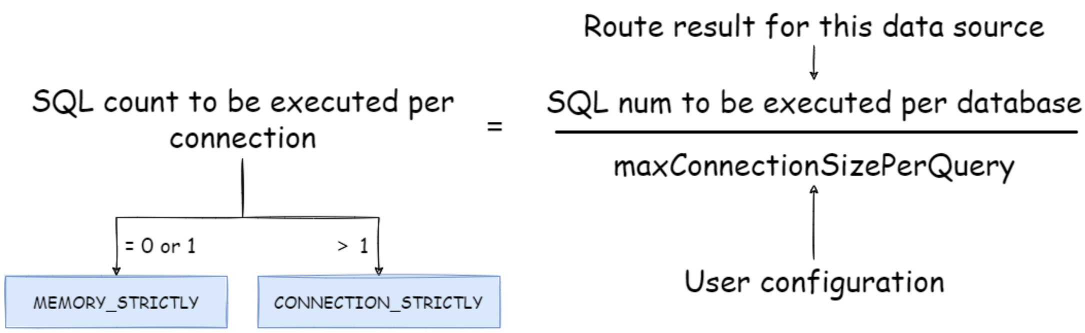


Figure9: Connection mode calculate formula

Within the scope of the maxConnectionSizePerQuery allowed, when the request number that one connection needs to execute is more than 1, the current database connection cannot hold the corresponding data result set, so it must use memory merger. On the contrary, when the number equals 1, the current database connection can hold the corresponding data result set, and it can use stream merger.

Each connection mode selection is specific to each physical database. That is, if you route to more than one database in the same query, the connection mode of each database may not be the same, and they may be mixed. Users can use the route grouping result acquired from the last step to create the execution unit. When the data source uses technologies, such as the database connection pool, to control database connection numbers, there is a chance that a deadlock will occur if concurrency is not handled properly while retrieving database connections. As multiple requests wait for each other to release database connection resources, starvation occurs, causing the crossing deadlock.

For example, suppose that a query requires obtaining two database connections at a data source and routing queries to two sub-tables of the same database. It is possible that query A has obtained one database connection from this data source and is waiting to obtain another database connection.

Query B has also acquired a database connection at the data source and is also waiting for another database connection to be acquired. If the maximum number of connections allowed in the database connection pool is 2, then the two query requests will wait forever. The following diagram depicts a deadlock situation.

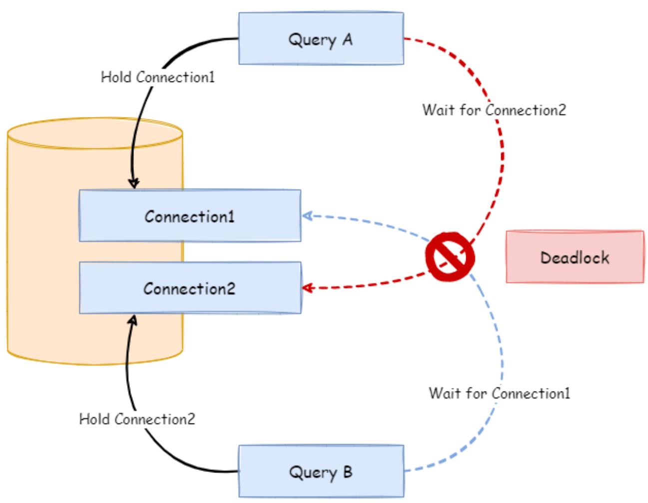


Figure10: Dead lock

ShardingSphere synchronizes database connections to avoid deadlocks. When it creates the execution unit, it atomically obtains all the database connections required by the SQL request at one time, eliminating the possibility of obtaining partial resources in each query request.

Because the operation on the database is very frequent, locking a database connection each time when acquiring it will reduce the concurrency of ShardingSphere. Therefore, ShardingSphere has improved two aspects here:

1. Locking can be avoided and only one database connection needs to be obtained each time. Because under this circumstance, two requests waiting for each other will not happen, so there is no need for locking. Most OLTP operations use shard keys to route to the unique data node, which makes the system completely unlocked and further improves the concurrency efficiency. In addition to routing to a single shard, read/write-splitting also belongs to this category.
2. Locking resources only happens in MEMORY_STRICTLY mode. When using CONNECTION_STRICTLY mode, all the query result sets will release database connection resources after loading them to the memory, so deadlock wait will not appear.

Execution Phrase

This stage is used to actually execute SQL and is divided into two steps: group execution and merger result generation.

Group execution can distribute execution unit groups generated in the preparation phase to the underlying concurrency engine and send events for each key step during the execution process, such as starting, successful and failed execution events. The execution engine only focuses on sending events rather than subscribers to the event. Other ShardingSphere modules, such as distributed transactions, call linked tracing and so on, will subscribe to the events of interest and process them accordingly.

ShardingSphere generates memory merger result sets or stream merger result sets through the connection mode acquired in the preparation phase. And then it passes the result set to the result merger engine for the next step.

The overall structure of the execution engine is divided as shown below.

9.4.11 Merger Engine

Result merger refers to merging multi-data result sets acquired from all the data nodes as one result set and returning it to the requesting client correctly.

The result merger supported by ShardingSphere can be divided into five functional types: traversal, order-by, group-by, pagination and aggregation, which are combined rather than mutually exclusive. From the perspective of structure, it can be divided into stream merger, memory merger and decorator merger, among which stream merger and memory merger are mutually exclusive, and decorator merger can be further processed based on stream merger and memory merger.

Since the result set is returned from the database one by one instead of being loaded to the memory all at a time, the method of merging the result sets returned from the database can greatly reduce memory consumption and is the preferred method of merging.

Stream merger means that each time the data is obtained from the result set is able to return the correct single piece of data line by line. It is the best fit with the native method of returning the result set of the database. Traversal, order-by, and stream group-by are all examples of the stream merger.

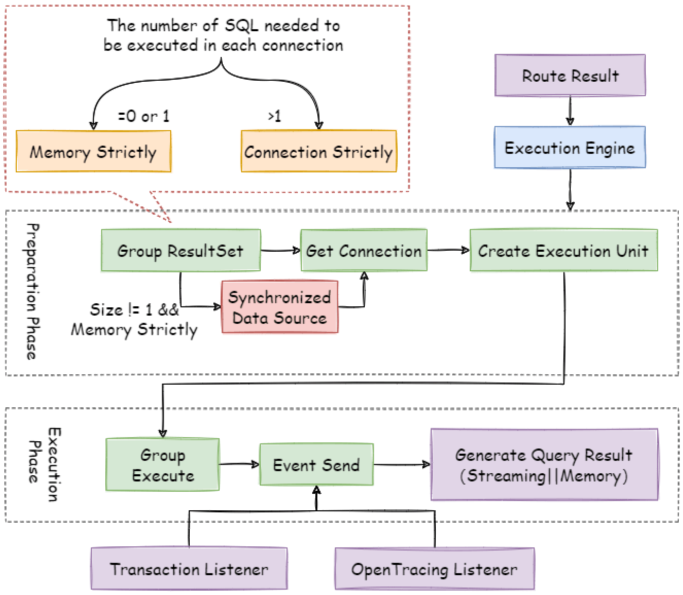


Figure11: Execute engine architecture

Memory merger needs to traverse all the data in the result set and store it in the memory first. After unified grouping, ordering, aggregation and other calculations, the data is packaged into the data result set accessed one by one and returned.

Decorator merger merges and reinforces all the result sets function uniformly. Currently, decorator merger has two types: pagination merger and aggregation merger.

Traversal Merger

As the simplest merger method, traversal merger only requires the combination of multiple data result sets into a one-way linked table. After traversing current data result sets in the linked table, it only needs to move the elements of the linked table back one bit and continue traversing the next data result set.

Order-by Merger

Because there is an `ORDER BY` statement in SQL, each data result has its own order. So it only needs to sort data value that the result set cursor currently points to, which is equal to sorting multiple ordered arrays. Therefore, order-by merger is the most suitable sorting algorithm in this scenario.

When merging ordered queries, ShardingSphere will compare current data values in each result set (which is realized by the Java Comparable interface) and put them into the priority queue. Each time when acquiring the next piece of data, it only needs to move down the result set cursor at the top of the queue, reenter the priority order according to the new cursor and relocate its own position.

Here is an instance to explain ShardingSphere' s order-by merger. The following picture is an illustration of ordering by the score. Data result sets returned by 3 tables are shown in the example and each of them has already been ordered according to the score, but there is no order between the 3 data result sets. Order the data value that the result set cursor currently points to in these 3 result sets. Then put them into the priority queue. The first data value of `t_score_0` is the biggest, followed by that of `t_score_2` and `t_score_1` in sequence. Thus, the priority queue is ordered by the sequence of `t_score_0`, `t_score_2` and `t_score_1`.

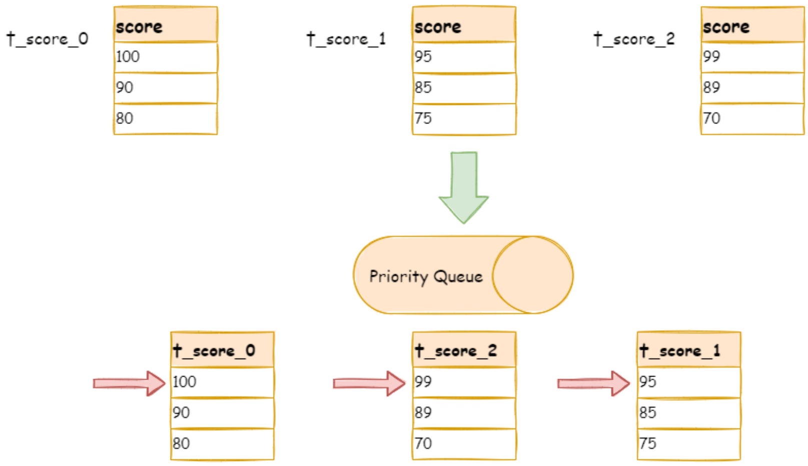


Figure12: Order by merger example 1

The following diagram illustrates how the order-by merger works when using next call. We can see from the diagram that when using the `next` call, `t_score_0` at the first of the queue will be popped out. After returning the data value currently pointed by the cursor (i.e., 100) to the requesting client, the cursor will be moved down and `t_score_0` will be put back into the queue.

While the priority queue will also be ordered according to the `t_score_0` data value (90 here) pointed by the cursor of the current data result set. According to the current value, `t_score_0` is at the end of the queue, and the data result set of `t_score_2`, originally in the second place of the queue, automatically moves to the first place of the queue.

In the second `next` call, `t_score_2` in the first place is popped out. Its value pointed by the cursor of the data result set is returned to the client end, with its cursor moved down to rejoin the queue, and the following will be the same way. If there is no data in the result set, it will not rejoin the queue.

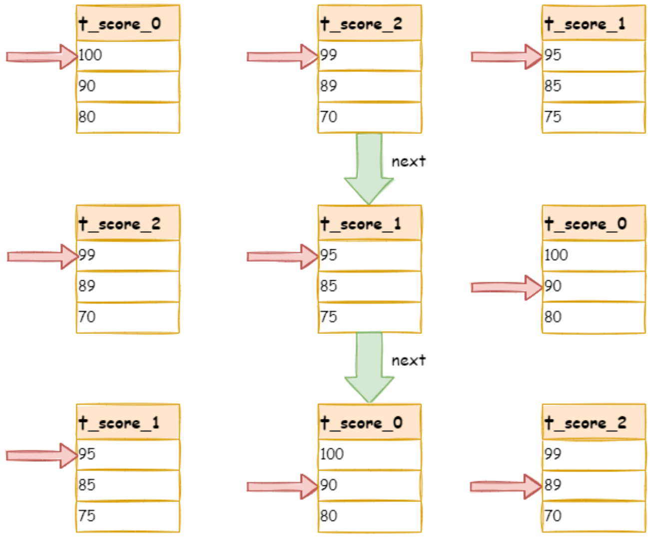


Figure13: Order by merger example 2

It can be seen that when data in each result set is ordered, but multiple result sets are disordered, ShardingSphere can still order them with no need to upload all the data to the memory. In the stream merger method, each `next` operation only acquires the right piece of data each time, which saves memory consumption to a large extent.

On the other hand, the order-by merger has maintained the orderliness on the horizontal axis and vertical axis of the data result set. Naturally ordered, the vertical axis refers to each data result set itself, which is acquired by SQL with `ORDER BY`. The horizontal axis refers to the current value pointed by each data result set, and its order needs to be maintained by the priority queue. Each time when the current cursor moves down, it requires putting the result set in the priority order again, which means only the cursor of the first data result set can be moved down.

Group-by Merger

Group-by merger is the most complex one and can be divided into stream group-by merger and memory group-by merger. Stream group-by merger requires that the SQL's ordering items must be consistent with the field and ordering types (ASC or DESC) of the group-by item; otherwise, data correctness can only be guaranteed by memory merger.

For instance, if it is sharded based on subject, the table structure contains the examinees' name (to simplify, name repetition is not taken into consideration) and score. The following SQL is used to acquire each examinee's total score:

```
SELECT name, SUM(score) FROM t_score GROUP BY name ORDER BY name;
```

When order-by item and group-by item are totally consistent, the data obtained is continuous. The data required by group-by is all stored in the data value that the data result set cursor currently points to. Thus, stream group-by merger can be used, as illustrated by the diagram:

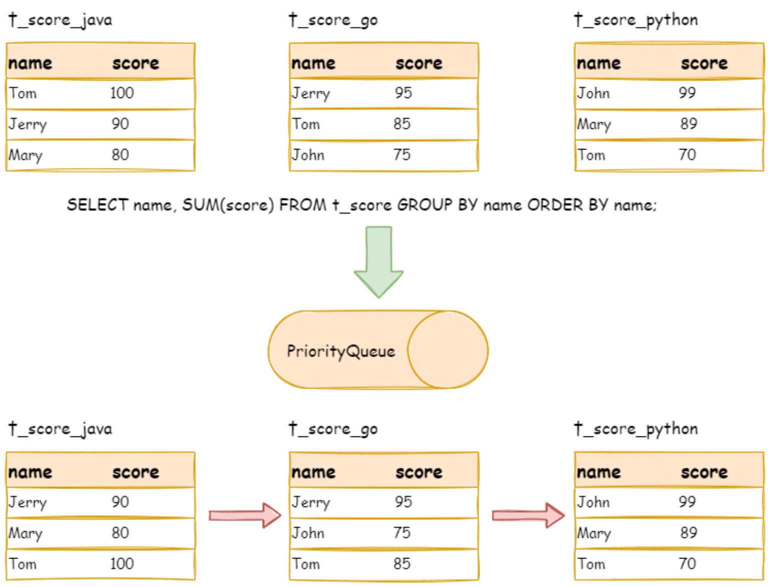


Figure14: Group by merger example 1

The merging logic is similar to that of order-by merger. The following picture shows how the stream group-by merger works in the `next` call.

We can see from the picture that, in the first `next` call, `t_score_java` in the first place will be popped out of the queue, along with other result set data having the same grouping value of "Jerry". After acquiring all the students' scores with the name of "Jerry", the accumulation operation will proceed. Hence, after the first `next` call is finished, the result set acquired is the sum of Jerry's scores. At the same time, all the cursors in data result sets will be moved down to a different data value next to "Jerry" and reordered according to the current result set value. Thus, the data that contains the second name "John" will be put at the beginning of the queue.

Stream group-by merger is different from order-by merger only in two aspects:

1. It will take out all the data with the same group item from multiple data result sets at once.
2. It carried out the aggregation calculation according to the aggregation function type.

For the inconsistency between the grouping item and ordering item, it requires uploading all the data to the memory to group and aggregate, since the relevant data value needed to acquire group information

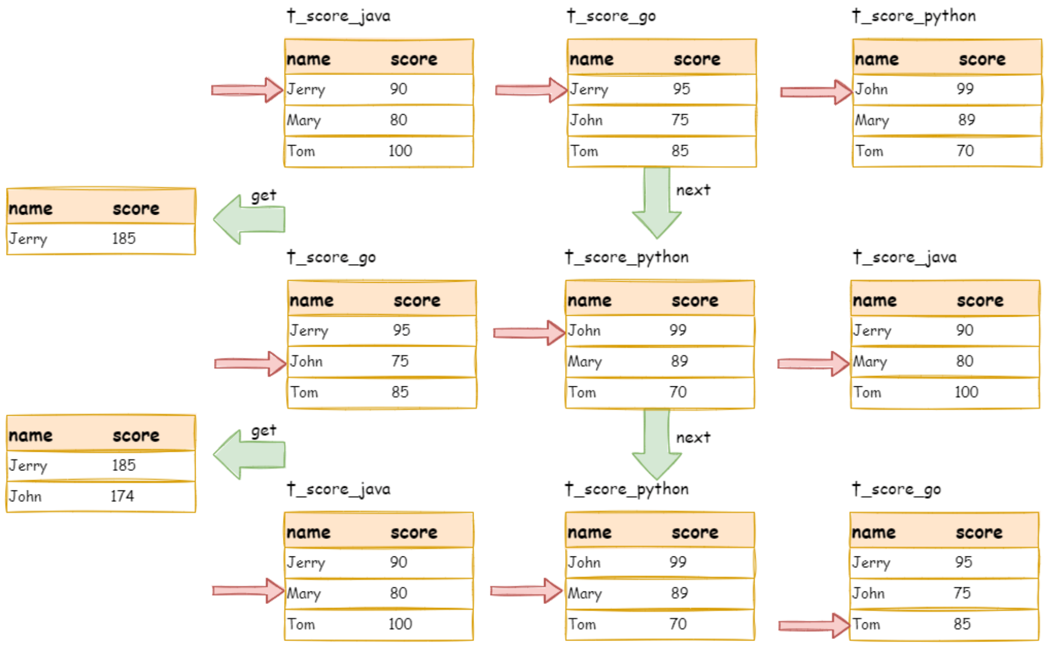


Figure15: Group by merger example 2

is not continuous, and stream merger is not available. For example, acquire each examinee' s total score through the following SQL and order them from the highest to the lowest:

```
SELECT name, SUM(score) FROM t_score GROUP BY name ORDER BY score DESC;
```

Then, stream merger is not able to use, for the data taken out from each result set is the same as the original data of the order-by merger diagram in the upper half part structure.

When SQL only contains the group-by statement, according to different database implementations, its sorting order may not be the same as the group order. The lack of an ordering statement indicates the order is not important in this SQL. Therefore, through the optimization of SQL rewriting, Sharding-Sphere can automatically add the ordering item the same as the grouping item, converting it from the memory merger that consumes memory to the stream merger.

Aggregation Merger

Whether it is stream group-by merger or memory group-by merger, they process the aggregation function in the same way. In addition to grouped SQL, ungrouped SQL can also use aggregate functions. Therefore, aggregation merger is an additional merging ability based on what has been introduced above, i.e., the decorator mode. The aggregation function can be categorized into three types: comparison, sum and average.

The comparison aggregation function refers to `MAX` and `MIN`. They need to compare all the result set data of each group and simply return the maximum or minimum value.

The sum aggregation function refers to `SUM` and `COUNT`. They need to sum up all the result set data of each group.

The average aggregation function refers only to `AVG`. It must be calculated through `SUM` and `COUNT` rewritten by SQL, which has been mentioned in the SQL rewriting section.

Pagination Merger

All the merger types above can be paginated. Pagination is the decorator added to other kinds of mergers. ShardingSphere strengthens its ability to paginate the data result set through decorator mode. The pagination merger is responsible for filtering unnecessary data.

ShardingSphere's pagination function can be misleading to users in that they may think it will take a large amount of memory. In distributed scenarios, it can only guarantee the data correctness by rewriting `LIMIT 10000000, 10` to `LIMIT 0, 10000010`. Users can easily misunderstand that ShardingSphere uploads a large amount of meaningless data to the memory and has the risk of memory overflow. Actually, it can be known from the principle of stream merger that only memory group-by merger will upload all the data to the memory. Generally speaking, SQL used for OLAP grouping, is often applied to massive calculations or small result generation, and it won't generate vast result data. Except for memory group-by merger, other scenarios all use stream merger to acquire data result set. So ShardingSphere would skip unnecessary data through the next call method in the result set, rather than storing it in the memory.

But it should be noted that pagination with `LIMIT` is not the best practice, because a large amount of data still needs to be transmitted to ShardingSphere's memory space for ordering. `LIMIT` cannot query data by index, so paginating with ID is a better solution if ID continuity can be guaranteed. For example:

```
SELECT * FROM t_order WHERE id > 100000 AND id <= 100010 ORDER BY id;
```

Or query the next page through the ID of the last query result, for example:

```
SELECT * FROM t_order WHERE id > 10000000 LIMIT 10;
```

The overall structure of the merger engine is shown in the following diagram:

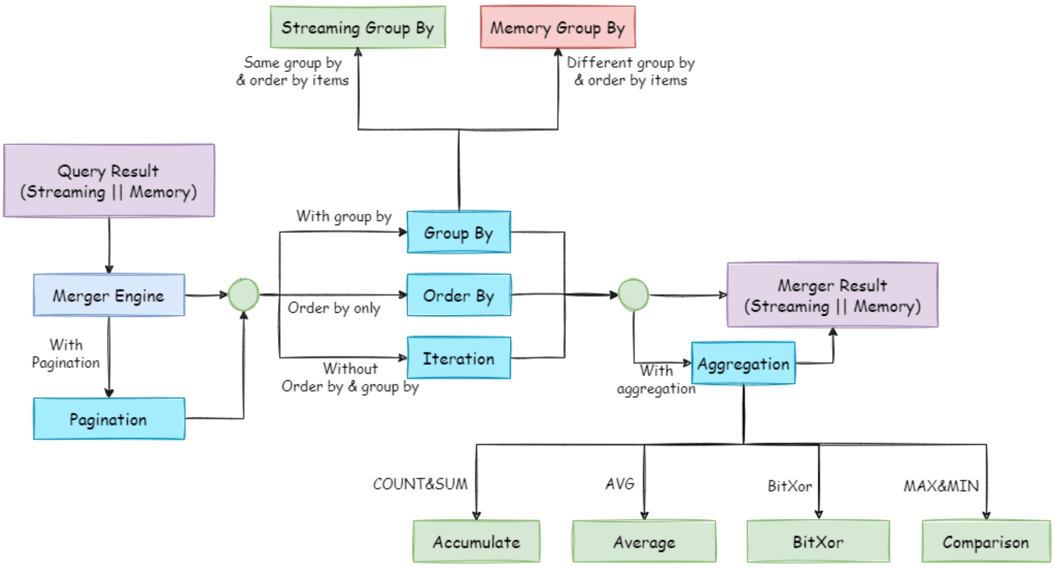


Figure16: Merge Architecture

9.5 Transaction

9.5.1 Navigation

This chapter mainly introduces the principles of the distributed transactions:

- 2PC transaction with XA
- BASE transaction with Seata

9.5.2 XA Transaction

`XAShardingSphereTransactionManager` is XA transaction manager of Apache ShardingSphere. Its main responsibility is manage and adapt multiple data sources, and sent corresponding transactions to concrete XA transaction manager.

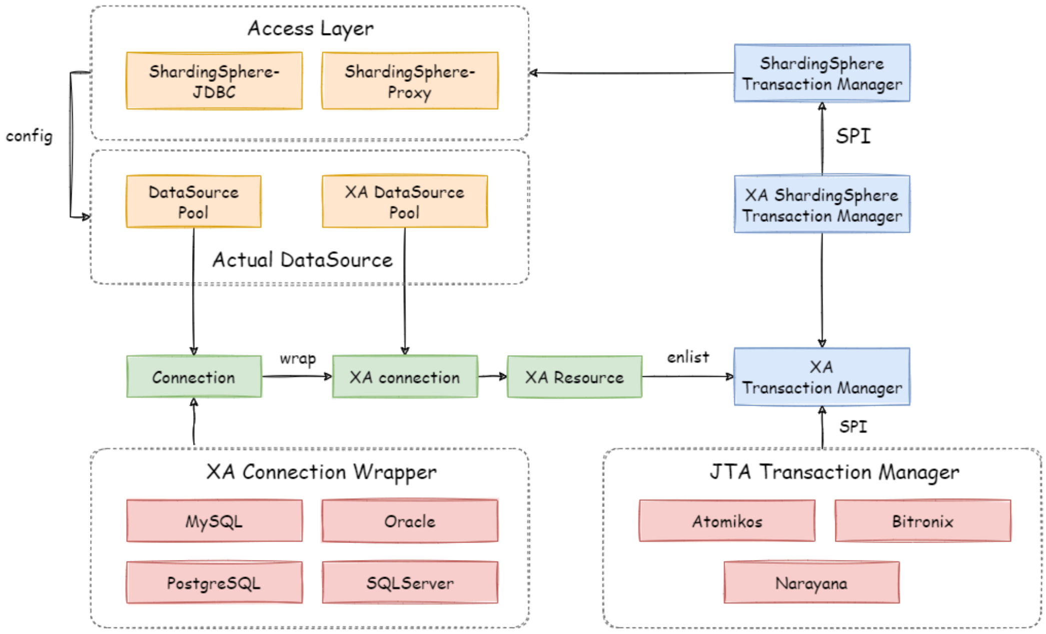


Figure17: Principle of ShardingSphere transaction XA

Transaction Begin

When receiving `set autoCommit=0` from client, `XAShardingSphereTransactionManager` will use XA transaction managers to start overall XA transactions, which is marked by XID.

Execute actual sharding SQL

After `XAShardingSphereTransactionManager` register the corresponding XAResource to the current XA transaction, transaction manager will send `XAResource.start` command to databases. After databases received `XAResource.end` command, all SQL operator will mark as XA transaction.

For example:

```
XAResource1.start              ## execute in the enlist phase
statement.execute("sql1");
statement.execute("sql2");
XAResource1.end                ## execute in the commit phase
```

`sql1` and `sql2` in example will be marked as XA transaction.

Commit or Rollback

After `XAShardingSphereTransactionManager` receives the commit command in the access, it will delegate it to the actual XA manager. It will collect all the registered XAResource in the thread, before sending `XAResource.end` to mark the boundary for the XA transaction. Then it will send prepare command one by one to collect votes from XAResource. If all the XAResource feedback is OK, it will send commit command to finally finish it; If there is any No XAResource feedback, it will send rollback command to roll back. After sending the commit command, all XAResource exceptions will be submitted again according to the recovery log to ensure the atomicity and high consistency.

For example:

```
XAResource1.prepare           ## ack: yes
XAResource2.prepare           ## ack: yes
XAResource1.commit
XAResource2.commit

XAResource1.prepare           ## ack: yes
XAResource2.prepare           ## ack: no
XAResource1.rollback
XAResource2.rollback
```

9.5.3 Seata BASE transaction

When integrating Seata AT transaction, we need to integrate TM, RM and TC component into ShardingSphere transaction manager. Seata have proxied `DataSource` interface in order to RPC with TC. Similarly, Apache ShardingSphere faced to `DataSource` interface to aggregate data sources too. After Seata `DataSource` encapsulation, it is easy to put Seata AT transaction into Apache ShardingSphere sharding ecosystem.

Init Seata Engine

When an application containing `ShardingSphereTransactionBaseSeataAT` startup, the user-configured DataSource will be wrapped into seata `DataSourceProxy` through `seata.conf`, then registered into RM.

Transaction Begin

TM controls the boundaries of global transactions. TM obtains the global transaction ID by sending Begin instructions to TC. All branch transactions participate in the global transaction through this global transaction ID. The context of the global transaction ID will be stored in the thread local variable.

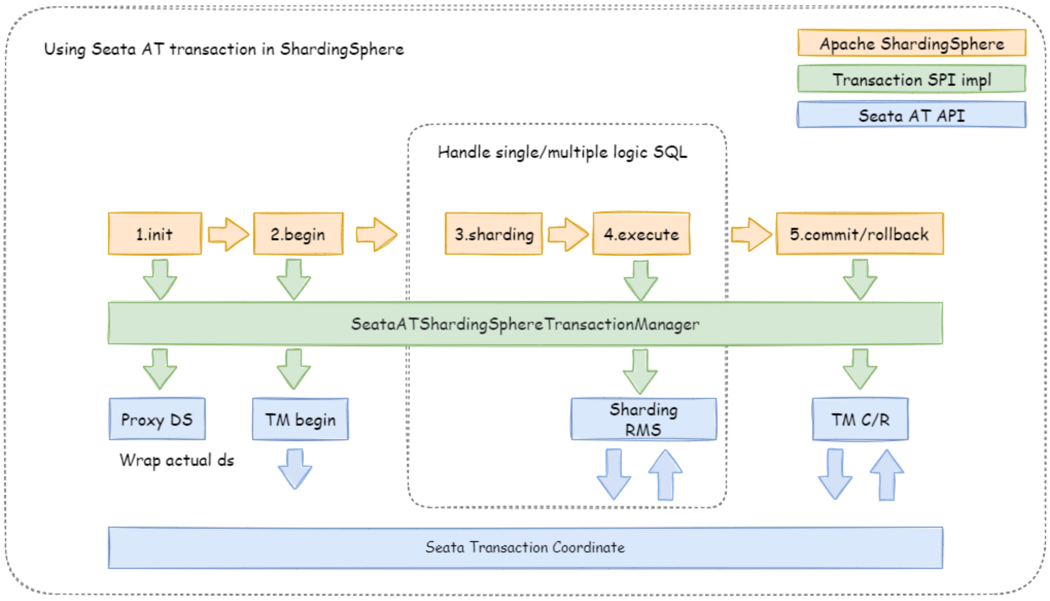


Figure18: Seata BASE transaction

Execute actual sharding SQL

Actual SQL in Seata global transaction will be intercepted to generate undo snapshots by RM and sends participate instructions to TC to join global transaction. Since actual sharding SQLs executed in multi-threads, global transaction context should transfer from main thread to child thread, which is exactly the same as context transfer between services.

Commit or Rollback

When submitting a seata transaction, TM sends TC the commit and rollback instructions of the global transaction. TC coordinates all branch transactions for commit and rollback according to the global transaction ID.

9.6 Data Migration

9.6.1 Explanation

The current data migration solution uses a completely new database cluster as the migration target.

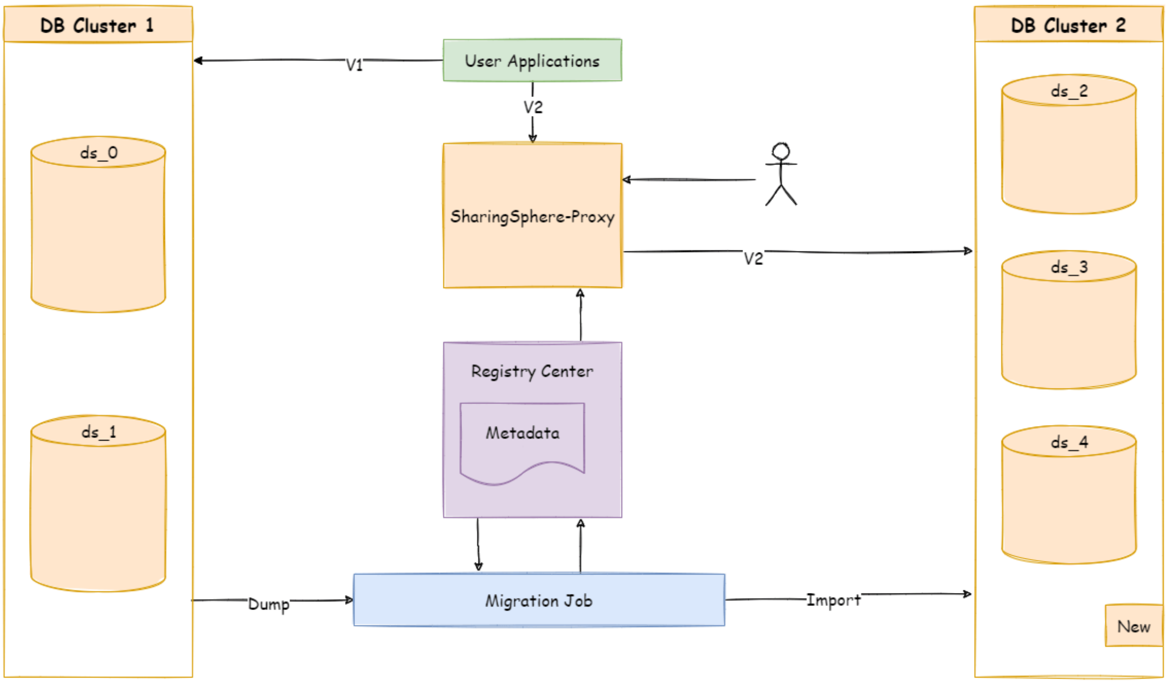


Figure19: Migration Overview

This implementation has the following advantages:

1. No impact on the original data during migration.
2. No risk in case of migration failure.
3. Free from sharding policy limitations.

The implementation has the following disadvantages:

1. Redundant servers can exist for a certain period of time.
2. All data needs to be moved.

A single data migration mainly consists of the following phases:

1. Preparation.
2. Stock data migration.
3. The synchronization of incremental data.

4. Traffic switching .

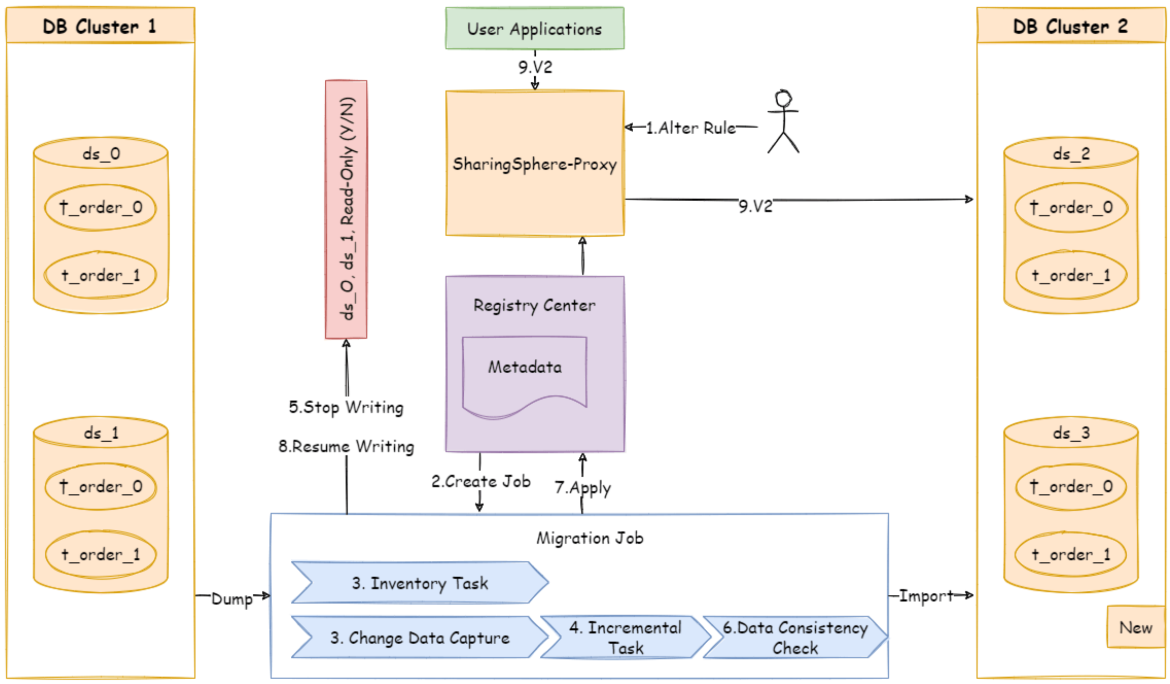


Figure20: Illustration

9.6.2 Execution Stage Explained

Preparation

In the preparation stage, the data migration module verifies data source connectivity and permissions, counts stock data statistics, records the log and finally shards the tasks according to data volume and parallelism set by the users.

Stock data migration

Execute the stock data migration tasks that have been sharded during preparation stage. The stock migration stage uses JDBC queries to read data directly from the source and write into the target based on the sharding rules and other configurations.

The Synchronization of incremental data

Since the duration of stock data migration depends on factors such as data volume and parallelism, it is necessary to synchronize the data added to the business operations during this period. Different databases differ in technical details, but in general they are all based on replication protocols or WAL logs to achieve the capture of changed data.

- MySQL: subscribe and parse binlog
- PostgreSQL: uses official logical replication test_decoding.

These incremental data captured are also written into the new data nodes by the data migration modules. When synchronization of incremental data is basically completed (the incremental data flow is not interrupted since the business system is still in function), you can then move to the traffic switching stage.

Traffic Switching

During this stage, there may be a read-only period of time, where data in the source data nodes is allowed to be in static mode for a short period of time to ensure that the incremental synchronization can be fully completed. Users can set this by using ShardingSphere's stop-write feature or by shifting the database to read-only status or by controlling the traffic flow generated from the source.

The length of this read-only window depends on whether users need to perform consistency checks on the data and the exact amount of data in this scenario. Once confirmed, the data migration is complete.

Users can then switch the read traffic or write traffic to Apache ShardingSphere.

9.6.3 References

Configurations of data migration

9.7 Encryption

Apache ShardingSphere parses the SQL entered by users and rewrites the SQL according to the encryption rules provided by users, to encrypt the source data and store the source data (optional) and ciphertext data in the underlying database.

When a user queries data, it only retrieves ciphertext data from the database, decrypts it, and finally returns the decrypted source data to the user. Apache ShardingSphere achieves a transparent and automatic data encryption process. Users can use encrypted data as normal data without paying attention to the implementation details of data encryption.

9.7.1 Overall Architecture

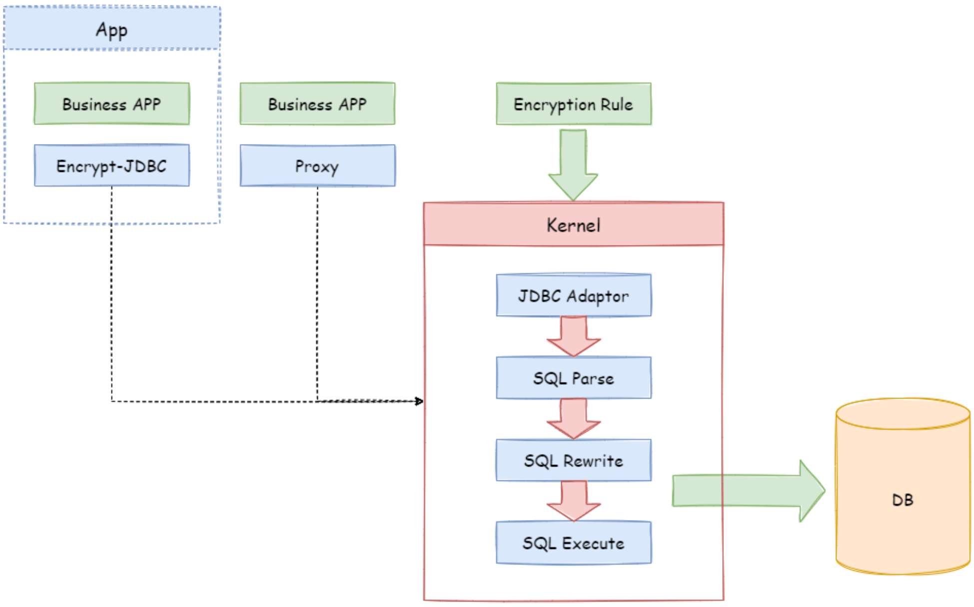


Figure21: 1

The encrypted module intercepts the SQL initiated by the user and parses and understands the SQL behavior through the SQL syntactic parser. Then it finds out the fields to be encrypted and the encryption and decryption algorithm according to the encryption rules introduced by the user and interacts with the underlying database.

Apache ShardingSphere will encrypt the plaintext requested by users and store it in the underlying database. When the user queries, the ciphertext is extracted from the database, decrypted, and returned to the terminal user. By shielding the data encryption process, users do not need to operate the SQL parsing process, data encryption, and data decryption.

9.7.2 Encryption Rules

Before explaining the whole process, we need to understand the encryption rules and configuration. Encryption configuration is mainly divided into four parts: data source configuration, encryptor configuration, encryption table configuration, and query attribute configuration, as shown in the figure below:

Data source configuration: the configuration of the data source.

Encryptor configuration: refers to the encryption algorithm used for encryption and decryption. Currently, ShardingSphere has three built-in encryption and decryption algorithms: AES, MD5, and RC4.

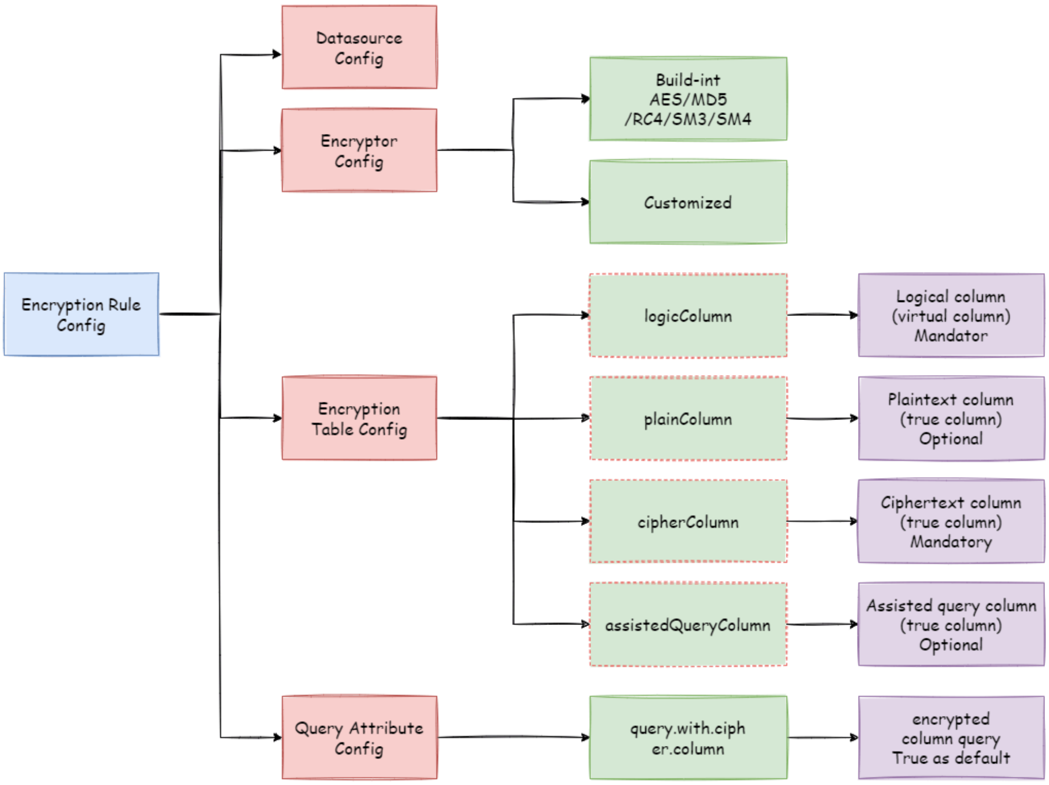


Figure22: 2

Users can also implement a set of encryption and decryption algorithms by implementing the interfaces provided by ShardingSphere.

Encryption table configuration: it is used to tell ShardingSphere which column in the data table is used to store ciphertext data (`cipherColumn`), which column is used to store plaintext data (`plainColumn`), and which column the user would like to use for SQL writing (`logicColumn`).

> What does it mean by "which column the user would like to use for SQL writing (logicColumn)"? We have to know first why the encrypted module exists. The goal of the encrypted module is to shield the underlying data encryption process, which means we don't want users to know how data is encrypted and decrypted, and how to store plaintext data into `plainColumn` and ciphertext data into `cipherColumn`. In other words, we don't want users to know there is a `plainColumn` and `cipherColumn` or how they are used. Therefore, we need to provide the user with a conceptual column that can be separated from the real column in the underlying database. It may or may not be a real column in the database table so that users can change the column names of `plainColumn` and `cipherColumn` of the underlying database at will. Or we can delete `plainColumn` and never store plaintext, only ciphertext. The only thing we have to ensure is that the user's SQL is written towards the logical column, and the correct mapping relation between `logicColumn`, `plainColumn`, and `cipherColumn` can be seen in the encryption rules.

Query attribute configuration: if both plaintext and ciphertext data are stored in the underlying database table, this attribute can be used to determine whether to query the plaintext data in the database table and return it directly, or query the ciphertext data and return it after decryption through Apache ShardingSphere. This attribute can be configured at the table level and the entire rule level. The table-level has the highest priority.

9.7.3 Encryption Process

For example, if there is a table named `t_user` in the database, and they' re two fields in the table: `pwd_plain` for storing plaintext data and `pwd_cipher` for storing ciphertext data, and logicColumn is defined as pwd, then users should write SQL for `logicColumn`, that is `INSERT INTO t_user SET pwd = '123'`. Apache ShardingSphere receives the SQL and finds that the pwd is the `logicColumn` based on the encryption configuration provided by the user. Therefore, it encrypts the logical column and its corresponding plaintext data.

Apache ShardingSphere transforms the column names and data encryption mapping between the logical columns facing users and the plain and cipher columns facing the underlying database. As shown in the figure below:

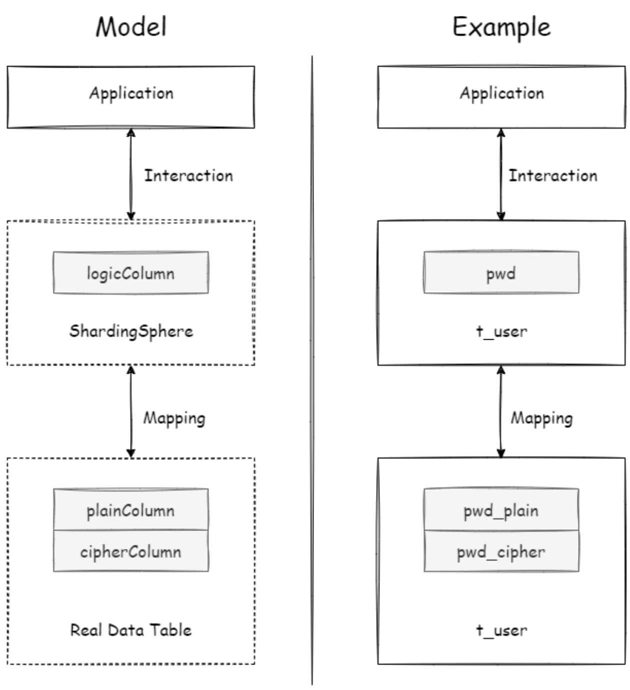


Figure23: 3

The user' s SQL is separated from the underlying data table structure according to the encryption rules provided by the user so that the user' s SQL writing does not depend on the real database table structure.

The connection, mapping, and transformation between the user and the underlying database are handled by Apache ShardingSphere.

The picture below shows the processing flow and conversion logic when the encryption module is used to add, delete, change and check, as shown in the figure below.

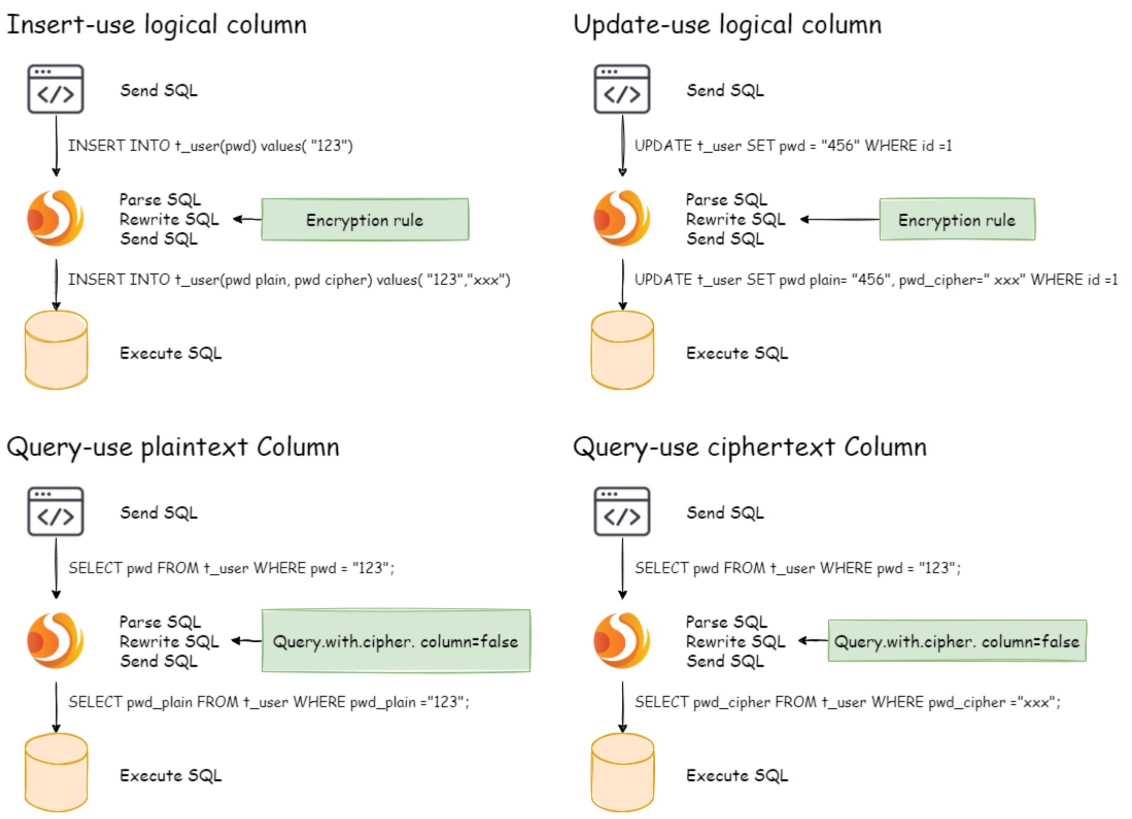


Figure24: 4

Detailed Solution

After understanding Apache ShardingSphere' s encryption process, you can combine the encryption configuration and encryption process according to your scenario. The entire design & development was conceived to address the pain points encountered in business scenarios. So, how to use Apache ShardingSphere to meet the business requirements mentioned before?

9.7.4 New Business

Business scenario analysis: the newly launched business is relatively simple because it starts from scratch and there' s no need to clean up historical data.

Solution description: after selecting the appropriate encryption algorithm, such as AES, you only need to configure the logical column (write SQL for users) and the ciphertext column (the data table stores the ciphertext data). The logical columns and ciphertext columns can also be different. The following configurations are recommended (in YAML format):

```yaml
-!ENCRYPT
 encryptors:
  aes_encryptor:
   type: AES
   props:
    aes-key-value: 123456abc
 tables:
  t_user:
   columns:
    pwd:
     cipherColumn: pwd_cipher
     encryptorName: aes_encryptor
     assistedQueryColumn: pwd_assisted_query
     assistedQueryEncryptorName: pwd_assisted_query_cipher
     queryWithCipherColumn: true
```

With the above configuration, Apache ShardingSphere only needs to convert `logicColumn`, `cipherColumn`, and `assistedQueryColumn`.

The underlying data table does not store plaintext, and only ciphertext is stored, which is also the requirement of the security audit. If you want to store both plaintext and ciphertext in the database, add the `plainColumn` configuration. The overall processing flow is shown in the figure below:

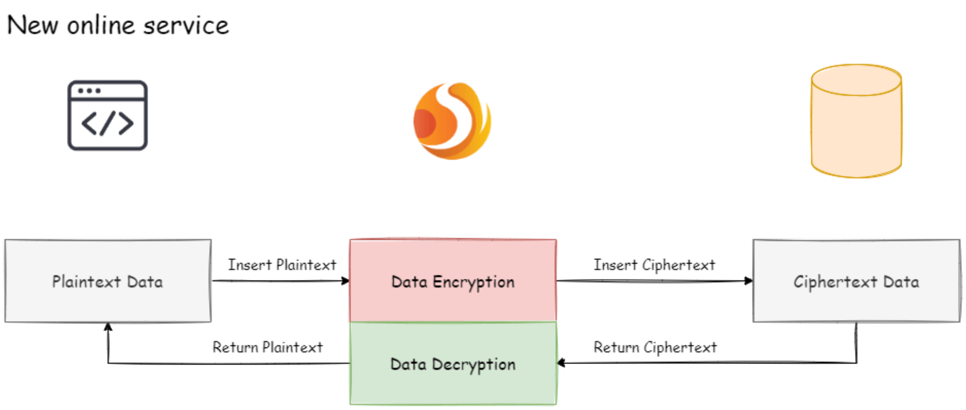


Figure25: 5

9.7.5 Online Business Transformation

Business scenario analysis: as the business is already running, the database will already have stored a large amount of plaintext historical data. The current challenges are how to encrypt and clean up the historical data, how to encrypt and process the incremental data, and how to seamlessly and transparently migrate business between the old and new data systems.

Solution Description: before coming up with a solution, let' s brainstorm.

First, since it is an old business that needs to be encrypted and transformed, it must have stored very important and sensitive information, which is valuable and related to critical businesses. Therefore, it is impossible to suspend business immediately, prohibit writing new data, encrypt and clean all historical data with an encryption algorithm. And then deploy and launch the reconstructed code to encrypt and decrypt the stock and incremental data online. Such a complex solution will definitely not work.

Another relatively safe solution is to build a set of pre-released environments exactly the same as the production environment, and then encrypt the stock original data of the production environment and store it in the pre-released environment through migration and data cleansing tools.

The new data is encrypted and stored in the database of the pre-released environment through tools such as MySQL primary/secondary replication and self-developed ones by the business side. The reconfigurable code that can be encrypted and decrypted is deployed to the pre-released environment. This way, the production environment takes plaintext as the core used for queries and modifications.

The pre-released environment is a ciphertext-based environment for encrypted and decrypted queries and modifications. After comparison, the production flow can be transferred to the pre-released environment by nighttime operation. This method is relatively safe and reliable, but time consuming,labor and capital intensive, mainly including building a pre-released environment, modifying production code, developing auxiliary tools, etc.

The most popular solutions for developers are to reduce the capital cost, not change the business code, and be able to migrate the system safely and smoothly. Thus, the encryption function module of ShardingSphere was created. It can be divided into three steps:

1. Before system migration

Assuming that the system needs to encrypt the pwd field of `t_user`, the business side uses Apache ShardingSphere to replace the standardized JDBC interface, which basically requires no additional modification (we also provide Spring Boot Starter, Spring Namespace, YAML and other access methods to meet different business requirements). In addition, we would like to demonstrate a set of encryption configuration rules, as follows:

```
-!ENCRYPT
  encryptors:
    aes_encryptor:
      type: AES
      props:
        aes-key-value: 123456abc
  tables:
    t_user:
```

```
      columns:
        pwd:
          plainColumn: pwd
          cipherColumn: pwd_cipher
          encryptorName: aes_encryptor
          assistedQueryColumn: pwd_assisted_query
          assistedQueryEncryptorName: pwd_assisted_query_cipher
          queryWithCipherColumn: false
```

According to the above encryption rules, we need to add a field called `pwd_cipher`, namely `cipherColumn`, in the `t_user` table, which is used to store ciphertext data.

At the same time, we set `plainColumn` to pwd, which is used to store plaintext data, and `logicColumn` is also set to pwd.

Because the previous SQL was written using pwd, the SQL was written for logical columns, and the business code does not need to be changed. Through Apache ShardingSphere, for the incremental data, the plaintext will be written to the pwd column and be encrypted and stored in the `pwd_cipher` column.

At this time, because `queryWithCipherColumn` is set to `false`, for business applications, the plaintext column of pwd is still used for query and storage, but the ciphertext data of the new data is additionally stored on the underlying database table `pwd_cipher`. The processing flow is shown below:

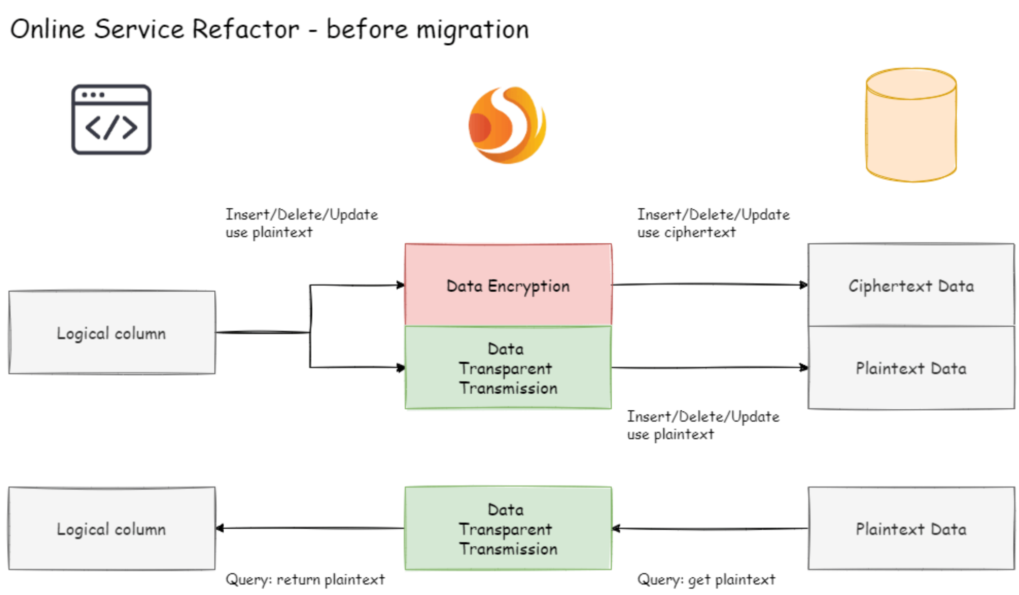


Figure26: 6

When the new data is inserted, it is encrypted as ciphertext data by Apache ShardingSphere and stored in the `cipherColumn`. Now you need to deal with the historical plaintext stock data. Apache ShardingSphere currently does not provide a migration and data cleansing tool, so you need to encrypt the plaintext data in the pwd and store it in the `pwd_cipher`.

2. During system migration

The new ciphertext data is stored in the `cipherColumn` and the new plaintext one is stored in the `plainColumn` by Apache ShardingSphere. After the historical data is encrypted and cleaned by the business side, its ciphertext is also stored in the `cipherColumn`. In other words, the current database stores both plaintext and ciphertext.

Owing to the configuration item `queryWithCipherColumn = false`, the ciphertext is never used. Now we need to set `queryWithCipherColumn` in the encryption configuration to true in order for the system to query ciphertext data.

After restarting the system, we found that all system businesses are normal, but Apache ShardingSphere has started to take out and decrypt the cipherColumn data from the database and returned those data to the user. In terms of users' requirements of addition, deletion and modification, the original data is still stored in the `plainColumn`, and the encrypted ciphertext data is stored in the `cipherColumn`.

Although the business system has taken out the data in the `cipherColumn` and returned it after decryption, it will still save a copy of the original data to the `plainColumn`. Why? The answer is: to enable system rollback.

Because as long as the ciphertext and plaintext always exist at the same time, we can freely switch the business query to `cipherColumn` or `plainColumn` through the configuration of the switch item.

In other words, if the system is switched to the ciphertext column for query, the system reports an error and needs to be rolled back. Then we only need to set `queryWithCipherColumn = false`, and Apache ShardingSphere will restore and start using `plainColumn` to query again. The processing flow is shown in the following figure:

3. After system migration

As required by security audit teams, it is generally impossible for the business system to permanently synchronize the plaintext column and ciphertext column of the database, so we need to delete the plaintext column data after the system is stable.

That is, we need to delete plainColumn (i.e.pwd) after system migration. The problem is that now the business code is written for pwd SQL, and we delete the pwd that stores plaintext in the underlying data table and use the `pwd_cipher` to decrypt the original data.

Does that mean that the business side needs to change all SQL, to not use the pwd column to be deleted? No. Remember the core concept of Apache ShardingSphere?

> That is exactly the core concept of Apache ShardingSphere's encryption module. According to the encryption rules provided by the user, the user SQL is separated from the underlying database table structure, so that the user's SQL writing no longer depends on the actual database table structure. The connection, mapping, and conversion between the user and the underlying database are handled by ShardingSphere.

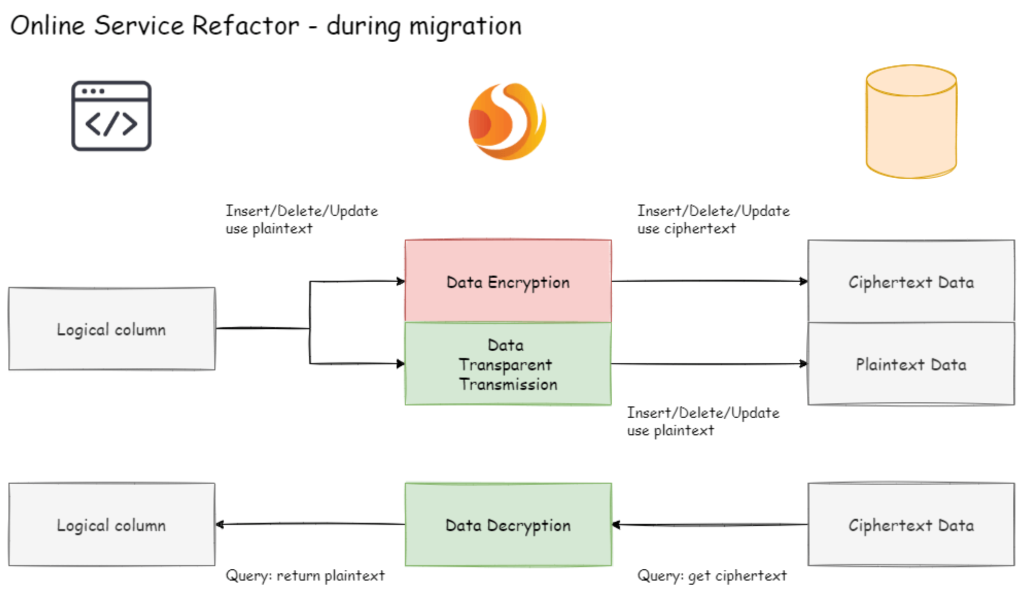


Figure27: 7

The existence of the `logicColumn` means that users write SQL for this virtual column. Apache ShardingSphere can map this logical column and the ciphertext column in the underlying data table. So the encryption configuration after the migration is:

```
-!ENCRYPT
  encryptors:
    aes_encryptor:
      type: AES
      props:
        aes-key-value: 123456abc
  tables:
    t_user:
      columns:
        pwd: # pwd and pwd_cipher transformation mapping
          cipherColumn: pwd_cipher
          encryptorName: aes_encryptor
          assistedQueryColumn: pwd_assisted_query
          assistedQueryEncryptorName: pwd_assisted_query_cipher
          queryWithCipherColumn: true
```

The processing flow is as follows:

4. System migration completed

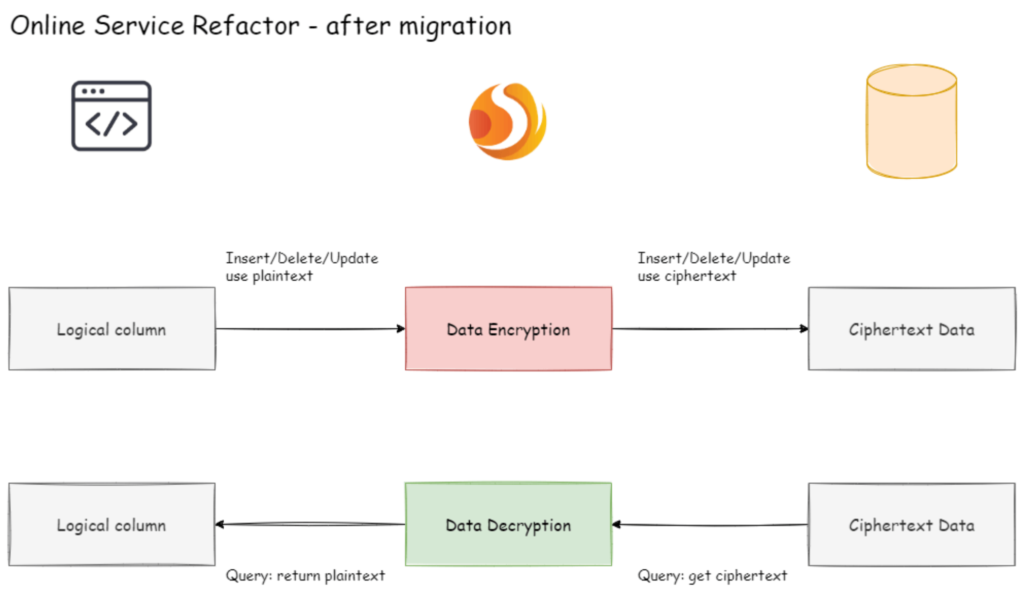


Figure28: 8

As required by security audit teams, the business system needs to periodically trigger key modifications or through some emergency events. We need to perform migration data cleansing again, which means using the old key to decrypt and then use the new key to encrypt.

The problem persists. The plaintext column data has been deleted, and the amount of data in the database table is tens of millions. Additionally, the migration and cleansing take a certain amount of time, during which the cipher column changes.

Under these circumstances, the system still needs to provide services correctly. What can we do? The answer lies in the auxiliary query column. Because auxiliary query columns generally use algorithms such as irreversible MD5 and SM3. Queries based on auxiliary columns are performed correctly by the system even during the migration and data cleansing process.

So far, the encryption rectification solution for the released business has been completely demonstrated. We provide Java, YAML, Spring Boot Starter, and Spring namespace for users to choose and access to meet different business requirements. This solution has been continuously verified by enterprise users such as JD Technology.

The advantages of Middleware encryption service

1. Automatic and transparent data encryption process. Encryption implementation details are no longer a concern for users.
2. It provides a variety of built-in and third-party (AKS) encryption algorithms, which are available through simple configurations.
3. It provides an encryption algorithm API interface. Users can implement the interface to use a custom encryption algorithm for data encryption.
4. It can switch among different encryption algorithms.
5. For businesses already launched, it is possible to store plaintext data and ciphertext data synchronously. And you can decide whether to use plaintext or ciphertext columns for query through configuration. Without changing the business query SQL, the released system can safely and transparently migrate data before and after encryption.

Solution

Apache ShardingSphere provides an encryption algorithm for data encryption, namely `EncryptAlgorithm`.

On the one hand, Apache ShardingSphere provides users with built-in implementation classes for encryption and decryption, which are available through configurations by users.

On the other hand, in order to be applicable to different scenarios, we also opened the encryption and decryption interfaces, and users can provide specific implementation classes according to these two types of interfaces.

After simple configuration, Apache ShardingSphere can call user-defined encryption and decryption schemes for data encryption.

9.7.6 EncryptAlgorithm

The solution provides two methods, `encrypt()` and `decrypt()`, to encrypt or decrypt data. When users perform `INSERT`, `DELETE` and `UPDATE` operations, ShardingSphere will parse, rewrite and route SQL according to the configuration.

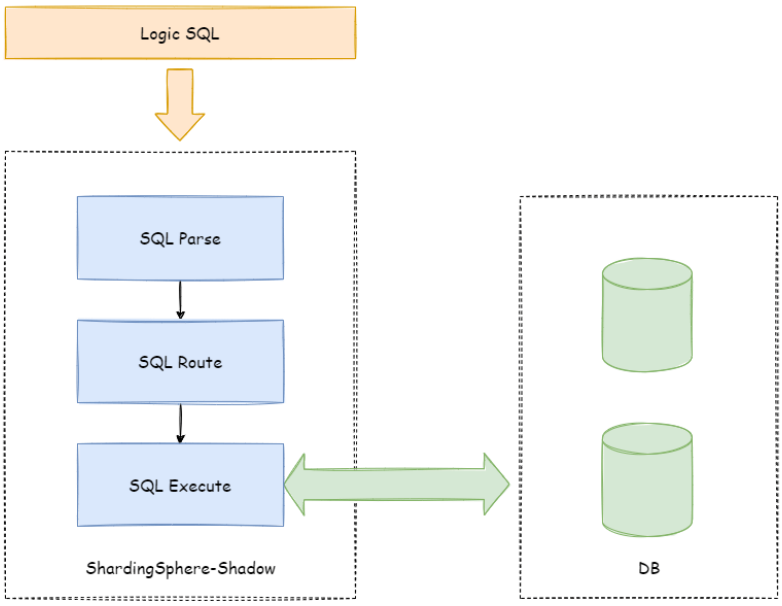
It will also use `encrypt()` to encrypt data and store them in the database. When using SELECT, they will decrypt sensitive data from the database with `decrypt()` and finally return the original data to users.

Currently, Apache ShardingSphere provides five types of implementations for this kind of encryption solution, including MD5 (irreversible), AES (reversible), RC4 (reversible), SM3 (irreversible) and SM4 (reversible), which can be used after configuration.

9.8 Shadow

9.8.1 How it works

Apache ShardingSphere determines the incoming SQL via shadow by parsing the SQL and routing it to the production or shadow database based on the shadow rules set by the user in the configuration file.



In the example of an INSERT statement, when writing data, Apache ShardingSphere parses the SQL and then constructs a routing chain based on the rules in the configuration file. In the current version, the shadow feature is at the last execution unit in the routing chain, i.e. if other rules exist that require routing, such as sharding, Apache ShardingSphere will first route to a particular database according to the sharding rules, and then run the shadow routing determination process to determine that the execution SQL meets the configuration set by shadow rules. Then data is routed to the corresponding shadow database, while the production data remains unchanged.

DML sentence

Two algorithms are supported. Shadow determination first determines whether the execution SQL-related table intersects with the configured shadow table. If the result is positive, the shadow algorithm within the part of intersection associated with the shadow table will be determined sequentially. If any of the determination is successful, the SQL statement is routed to the shadow library. If there is no intersection or the shadow algorithm determination is unsuccessful, the SQL statement is routed to the production database.

DDL sentence

Only supports shadow algorithm with comments attached. In stress testing scenarios, DDL statements are generally not required for testing, and are used mainly when initializing or modifying shadow tables in the shadow database. The shadow determination will first determine whether the execution SQL contains comments or not. If the result is a yes, the HINT shadow algorithm configured in the shadow rules determines them in order. The SQL statement is routed to the shadow database if any of the determinations are successful. If the execution SQL does not contain comments or the HINT shadow algorithm determination is unsuccessful, the SQL statements are routed to the production database.

9.8.2 References

JAVA API: shadow database configuration

YAMLconfiguration: shadow database

Spring Boot Starter: shadow database configuration

Spring namespace: shadow database configuration

9.9 Oberservability

9.9.1 How it works

ShardingSphere-Agent module provides an observable framework for ShardingSphere, which is implemented based on Java Agent.

Metrics, tracing and logging functions are integrated into the agent through plugins, as shown in the following figure:

- The Metrics plugin is used to collect and display statistical indicators for the entire cluster. Apache ShardingSphere supports Prometheus by default.
- The tracing plugin is used to obtain the link trace information of SQL parsing and SQL execution. Apache ShardingSphere provides support for Jaeger, OpenTelemetry, OpenTracing(SkyWalking) and Zipkin by default. It also supports users developing customized tracing components through plugin.
- The default logging plugin shows how to record additional logs in ShardingSphere. In practical applications, users need to explore according to their own needs.

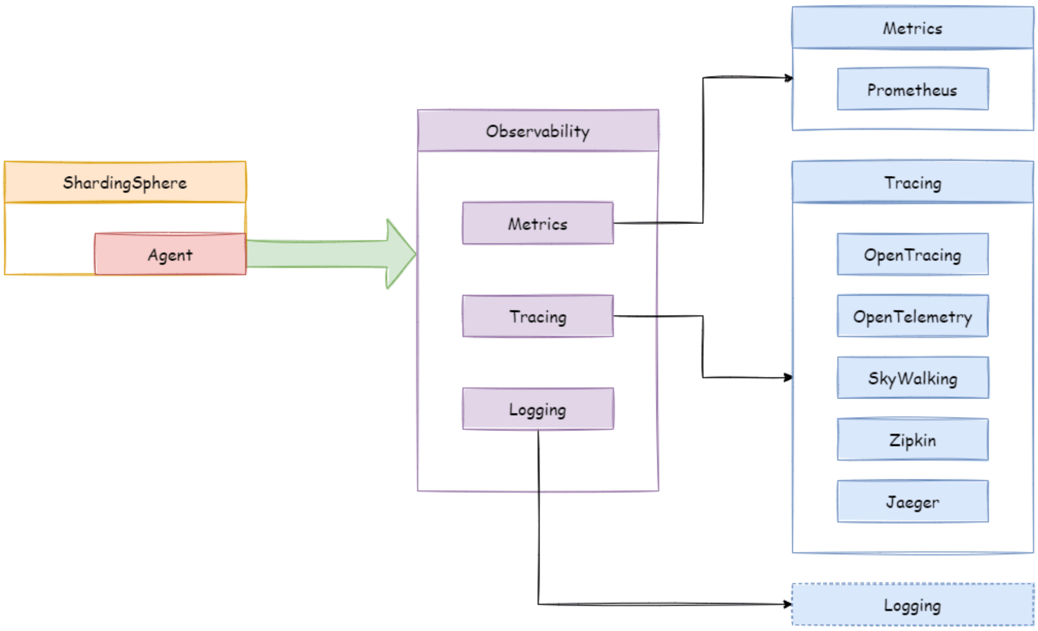


Figure29: Overview

9.10 DistSQL

This chapter will introduce the detailed syntax of DistSQL.

9.10.1 Syntax

This chapter describes the syntax of DistSQL in detail, and introduces use of DistSQL with practical examples.

RDL Syntax

RDL (Resource & Rule Definition Language) responsible for definition of resources/rules.

Resource Definition

This chapter describes the syntax of resource definition.

ADD RESOURCE

Description

The ADD RESOURCE syntax is used to add resources for the currently selected database.

Syntax

```
AddResource ::=
  'ADD' 'RESOURCE' resourceDefinition (',' resourceDefinition)*

resourceDefinition ::=
  resourceName '(' ( 'HOST' '=' hostName ',' 'PORT' '=' port ',' 'DB' '=' dbName  |
 'URL' '=' url  ) ',' 'USER' '=' user (',' 'PASSWORD' '=' password )?  (','
proerties)?')'

resourceName ::=
  identifier

hostname ::=
  string
```

```
port ::=
  int

dbName ::=
  string

url ::=
  string

user ::=
  string

password ::=
  string

proerties ::=
  PROPERTIES '(' property ( ',' property )* ')'

property ::=
  key '=' value

key ::=
  string

value ::=
  string
```

Supplement

- Before adding resources, please confirm that a database has been created in Proxy, and execute the `use` command to successfully select a database;
- Confirm that the added resource can be connected normally, otherwise it will not be added successfully;
- `resourceName` is case-sensitive;
- `resourceName` needs to be unique within the current database;
- `resourceName` name only allows letters, numbers and _, and must start with a letter;
- `poolProperty` is used to customize connection pool parameters, `key` must be the same as the connection pool parameter name, `value` supports int and String types;
- When `password` contains special characters, it is recommended to use the string form; For example, the string form of `password@123` is `"password@123"`.

Example

- Add resource using standard mode

```
ADD RESOURCE ds_0 (
    HOST=127.0.0.1,
    PORT=3306,
    DB=db_0,
    USER=root,
    PASSWORD=root
);
```

- Add resource and set connection pool parameters using standard mode

```
ADD RESOURCE ds_1 (
    HOST=127.0.0.1,
    PORT=3306,
    DB=db_1,
    USER=root,
    PASSWORD=root,
    PROPERTIES("maximumPoolSize"=10)
);
```

- Add resource and set connection pool parameters using URL patterns

```
ADD RESOURCE ds_2 (
    URL="jdbc:mysql://127.0.0.1:3306/db_2?serverTimezone=UTC&useSSL=false",
    USER=root,
    PASSWORD=root,
    PROPERTIES("maximumPoolSize"=10,"idleTimeout"="30000")
);
```

Reserved word

ADD, RESOURCE, HOST, PORT, DB, USER, PASSWORD, PROPERTIES, URL

Related links

- Reserved word

ALTER RESOURCE

Description

The `ALTER RESOURCE` syntax is used to alter resources for the currently selected database.

Syntax

```
AlterResource ::=
  'ALTER' 'RESOURCE' resourceDefinition (',' resourceDefinition)*

resourceDefinition ::=
  resourceName '(' ( 'HOST' '=' hostName ',' 'PORT' '=' port ',' 'DB' '=' dbName  |
 'URL' '=' url  ) ',' 'USER' '=' user (',' 'PASSWORD' '=' password )?  (','
proerties)?')'

resourceName ::=
  identifier

hostname ::=
  string

port ::=
  int

dbName ::=
  string

url ::=
  string

user ::=
  string

password ::=
  string

proerties ::=
  PROPERTIES '(' property ( ',' property )* ')'

property ::=
  key '=' value
```

```
key ::=
  string

value ::=
  string
```

Supplement

- Before altering the resources, please confirm that a database exists in Proxy, and execute the `use` command to successfully select a database;
- `ALTER RESOURCE` is not allowed to change the real data source associated with this resource;
- `ALTER RESOURCE` will switch the connection pool. This operation may affect the ongoing business, please use it with caution;
- `resourceName` is case-sensitive;
- `resourceName` needs to be unique within the current database;
- `resourceName` name only allows letters, numbers and _, and must start with a letter;
- `poolProperty` is used to customize connection pool parameters, `key` must be the same as the connection pool parameter name, `value` supports int and String types;
- When `password` contains special characters, it is recommended to use the string form; for example, the string form of `password@123` is `"password@123"`.

Example

- Alter resource using standard mode

```
ALTER RESOURCE ds_0 (
    HOST=127.0.0.1,
    PORT=3306,
    DB=db_0,
    USER=root,
    PASSWORD=root
);
```

- Alter resource and set connection pool parameters using standard mode

```
ALTER RESOURCE ds_1 (
    HOST=127.0.0.1,
    PORT=3306,
    DB=db_1,
    USER=root,
    PASSWORD=root
```

```
    PROPERTIES("maximumPoolSize"=10)
);
```

- Alter resource and set connection pool parameters using URL patterns

```
ALTER RESOURCE ds_2 (
    URL="jdbc:mysql://127.0.0.1:3306/db_2?serverTimezone=UTC&useSSL=false",
    USER=root,
    PASSWORD=root,
    PROPERTIES("maximumPoolSize"=10,"idleTimeout"="30000")
);
```

Reserved word

ALTER, RESOURCE, HOST, PORT, DB, USER, PASSWORD, PROPERTIES, URL

Related links

- Reserved word

DROP RESOURCE

Description

The DROP RESOURCE syntax is used to drop resources from the current database

Syntax

```
DropResource ::=
  'DROP' 'RESOURCE' ( 'IF' 'EXISTS' )? resourceName  ( ',' resourceName )* (
'IGNORE' 'SINGLE' 'TABLES' )?

resourceName ::=
  identifier
```

Supplement

- DROP RESOURCE will only drop resources in Proxy, the real data source corresponding to the resource will not be dropped;
- Unable to drop resources already used by rules. Resources are still in used. will be prompted when removing resources used by rules;
- The resource need to be removed only contains SINGLE TABLE RULE, and when the user confirms that this restriction can be ignored, the IGNORE SINGLE TABLES keyword can be added to remove the resource.

Example

- Drop a resource

```sql
DROP RESOURCE ds_0;
```

- Drop multiple resources

```sql
DROP RESOURCE ds_1, ds_2;
```

- Ignore single table rule remove resource

```sql
DROP RESOURCE ds_1 IGNORE SINGLE TABLES;
```

- Drop the resource if it exists

```sql
DROP RESOURCE IF EXISTS ds_2;
```

Reserved word

DROP, RESOURCE, IF, EXISTS, IGNORE, SINGLE, TABLES

Related links

- Reserved word

Rule Definition

This chapter describes the syntax of rule definition.

Database Discovery

This chapter describes the syntax of database discovery.

CREATE DB_DISCOVERY RULE

Description

The `CREATE DB_DISCOVERY RULE` syntax is used to create a database discovery rule.

Syntax

```
CreateDatabaseDiscoveryRule ::=
  'CREATE' 'DB_DISCOVERY' 'RULE' ( databaseDiscoveryDefinition |
databaseDiscoveryConstruction ) ( ',' ( databaseDiscoveryDefinition |
databaseDiscoveryConstruction ) )*

databaseDiscoveryDefinition ::=
    ruleName '(' 'RESOURCES' '(' resourceName ( ',' resourceName )* ')' ',' 'TYPE'
'(' 'NAME' '=' typeName ( ',' 'PROPERTIES' 'key' '=' 'value' ( ',' 'key' '=' 'value
' )* )? ',' 'HEARTBEAT' '(' 'key' '=' 'value' ( ',' 'key' '=' 'value' )* ')' ')'

databaseDiscoveryConstruction ::=
    ruleName '(' 'RESOURCES' '(' resourceName ( ',' resourceName )* ')' ',' 'TYPE'
'=' discoveryTypeName ',' 'HEARTBEAT' '=' discoveryHeartbeatName ')'

ruleName ::=
  identifier

resourceName ::=
  identifier

typeName ::=
  identifier

discoveryHeartbeatName ::=
  identifier
```

Supplement

- discoveryType specifies the database discovery service type, ShardingSphere has built-in support for MySQL.MGR;
- Duplicate ruleName will not be created;
- The discoveryType and discoveryHeartbeat being used cannot be deleted;
- Names with - need to use " " when changing。

Example

When creating a discoveryRule, create both discoveryType and discoveryHeartbeat

```
CREATE
DB_DISCOVERY RULE db_discovery_group_0 (
    RESOURCES(ds_0, ds_1, ds_2),
    TYPE(NAME='MySQL.MGR',PROPERTIES('group-name'='92504d5b-6dec')),
    HEARTBEAT(PROPERTIES('keep-alive-cron'='0/5 * * * * ?'))
);
```

Use the existing discoveryType and discoveryHeartbeat to create a discoveryRule

```
CREATE
DB_DISCOVERY RULE db_discovery_group_1 (
    RESOURCES(ds_0, ds_1, ds_2),
    TYPE=db_discovery_group_1_mgr,
    HEARTBEAT=db_discovery_group_1_heartbeat
);
```

Reserved word

CREATE, DB_DISCOVERY, RULE, RESOURCES, TYPE, NAME, PROPERTIES, HEARTBEAT

Related links

- Reserved word

Encrypt

This chapter describes the syntax of encrypt.

CREATE ENCRYPT RULE

Description

The `CREATE READWRITE_SPLITTING RULE` syntax is used to create a readwrite splitting rule.

Syntax

```
CreateEncryptRule ::=
  'CREATE' 'ENCRYPT' 'RULE' encryptDefinition ( ',' encryptDefinition )*

encryptDefinition ::=
  tableName '(' 'COLUMNS' '(' columnDefinition ( ',' columnDefinition )*  ')' ','
'QUERY_WITH_CIPHER_COLUMN' '=' queryWithCipherColumn ')'

columnDefinition ::=
    'NAME' '=' columnName ',' ( 'PLAIN' '=' plainColumnName )? 'CIPHER' '='
cipherColumnName ','  'TYPE' '(' 'NAME' '=' encryptAlgorithmType ( ',' 'PROPERTIES'
'(' 'key' '=' 'value' ( ',' 'key' '=' 'value' )* ')' )? ')'

tableName ::=
  identifier

queryWithCipherColumn ::=
  identifier

columnName ::=
  identifier

plainColumnName ::=
  identifier

cipherColumnName ::=
  identifier

encryptAlgorithmType ::=
  identifier
```

Supplement

- PLAIN specifies the plain column, CIPHER specifies the cipher column
- encryptAlgorithmType specifies the encryption algorithm type, please refer to Encryption Algorithm
- Duplicate tableName will not be created
- queryWithCipherColumn support uppercase or lowercase true or false

Example

Create a encrypt rule

```
CREATE ENCRYPT RULE t_encrypt (
COLUMNS(
(NAME=user_id,PLAIN=user_plain,CIPHER=user_cipher,TYPE(NAME='AES',PROPERTIES('aes-key-value'='123456abc'))),
(NAME=order_id, CIPHER =order_cipher,TYPE(NAME='MD5'))
),QUERY_WITH_CIPHER_COLUMN=true),
t_encrypt_2 (
COLUMNS(
(NAME=user_id,PLAIN=user_plain,CIPHER=user_cipher,TYPE(NAME='AES',PROPERTIES('aes-key-value'='123456abc'))),
(NAME=order_id, CIPHER=order_cipher,TYPE(NAME='MD5'))
), QUERY_WITH_CIPHER_COLUMN=FALSE);
```

Reserved word

CREATE, ENCRYPT, RULE, COLUMNS, NAME, CIPHER, PLAIN, QUERY_WITH_CIPHER_COLUMN, TYPE, TRUE, FALSE

Related links

- Reserved word

Readwrite-Splitting

This chapter describes the syntax of readwrite splitting.

CREATE READWRITE_SPLITTING RULE

Description

The `CREATE READWRITE_SPLITTING RULE` syntax is used to create a readwrite splitting rule.

Syntax

```
CreateReadwriteSplittingRule ::=
  'CREATE' 'READWRITE_SPLITTING' 'RULE' readwriteSplittingDefinition ( ','
readwriteSplittingDefinition )*

readwriteSplittingDefinition ::=
  ruleName '(' ( staticReadwriteSplittingDefinition |
dynamicReadwriteSplittingDefinition ) ( ',' loadBanlancerDefinition )? ')'

staticReadwriteSplittingDefinition ::=
    'WRITE_RESOURCE' '=' writeResourceName ',' 'READ_RESOURCES' '(' ruleName (','
ruleName)* ')'

dynamicReadwriteSplittingDefinition ::=
    'AUTO_AWARE_RESOURCE' '=' resourceName ( ',' 'WRITE_DATA_SOURCE_QUERY_ENABLED'
'=' ('TRUE' | 'FALSE') )?

loadBanlancerDefinition ::=
    'TYPE' '(' 'NAME' '=' loadBanlancerType ( ',' 'PROPERTIES' '(' 'key' '=' 'value
' ( ',' 'key' '=' 'value' )* ')' )? ')'

ruleName ::=
  identifier

writeResourceName ::=
  identifier

resourceName ::=
  identifier
```

Supplement

- Support the creation of static readwrite-splitting rules and dynamic readwrite-splitting rules;
- Dynamic readwrite-splitting rules rely on database discovery rules;
- `loadBanlancerType` specifies the load balancing algorithm type, please refer to Load Balance Algorithm;
- Duplicate `ruleName` will not be created.

Example

Create a statics readwrite splitting rule

```
CREATE READWRITE_SPLITTING RULE ms_group_0 (
    WRITE_RESOURCE=write_ds,
    READ_RESOURCES(read_ds_0,read_ds_1),
    TYPE(NAME="random")
);
```

Create a dynamic readwrite splitting rule

```
CREATE READWRITE_SPLITTING RULE ms_group_1 (
    AUTO_AWARE_RESOURCE=group_0,
    WRITE_DATA_SOURCE_QUERY_ENABLED=false,
    TYPE(NAME="random",PROPERTIES("read_weight"="2:1"))
);
```

Reserved word

`CREATE`, `READWRITE_SPLITTING`, `RULE`, `WRITE_RESOURCE`, `READ_RESOURCES`, `AUTO_AWARE_RESOURCE` , `WRITE_DATA_SOURCE_QUERY_ENABLED`, `TYPE`, `NAME`, `PROPERTIES`, `TRUE`, `FALSE`

Related links

- Reserved word

Shadow

This chapter describes the syntax of shadow.

CREATE SHADOW RULE

Description

The `CREATE SHADOW RULE` syntax is used to create a default shadow rule.

Syntax

```
CreateShadowRule ::=
  'CREATE' 'SHADOW' 'RULE' shadowDefinition ( ',' shadowDefinition )*

shadowDefinition ::=
  ruleName '(' resourceMapping shadowTableRule ( ',' shadowTableRule )* ')'

resourceMapping ::=
    'SOURCE' '=' resourceName ',' 'SHADOW' '=' resourceName

shadowTableRule ::=
    tableName '(' shadowAlgorithm ( ',' shadowAlgorithm )* ')'

shadowAlgorithm ::=
    ( algorithmName ',' )? 'TYPE' '('  'NAME' '=' shadowAlgorithmType ','
'PROPERTIES' '(' 'key' '=' 'value' ( ',' 'key' '=' 'value' ) ')'

ruleName ::=
  identifier

resourceName ::=
  identifier

tableName ::=
  identifier

algorithmName ::=
  identifier

shadowAlgorithmType ::=
  identifier
```

Supplement

- Duplicate `ruleName` cannot be created;
- `resourceMapping` specifies the mapping relationship between the `source` database and the shadow library. You need to use the resource managed by RDL, please refer to resource;
- `shadowAlgorithm` can act on multiple `shadowTableRule` at the same time;
- If `algorithmName` is not specified, it will be automatically generated according to `ruleName`, `tableName` and `shadowAlgorithmType`;
- `shadowAlgorithmType` currently supports `VALUE_MATCH`, `REGEX_MATCH` and `SIMPLE_HINT`.

Example

Create a shadow rule

```
CREATE SHADOW RULE shadow_rule(
  SOURCE=demo_ds,
  SHADOW=demo_ds_shadow,
  t_order((simple_hint_algorithm, TYPE(NAME="SIMPLE_HINT", PROPERTIES("shadow"=
"true", "foo"="bar"))),(TYPE(NAME="REGEX_MATCH", PROPERTIES("operation"="insert",
"column"="user_id", "regex"='[1]')))),
  t_order_item((TYPE(NAME="VALUE_MATCH", PROPERTIES("operation"="insert","column"=
"user_id", "value"='1'))))
);
```

Reserved word

CREATE, SHADOW, RULE, SOURCE, SHADOW, TYPE, NAME, PROPERTIES

Related links

- Reserved word

Sharding

This chapter describes the syntax of sharding.

CREATE SHARDING TABLE RULE

Description

The CREATE SHARDING TABLE RULE syntax is used to add sharding table rule for the currently selected database

Syntax

```
CreateShardingTableRule ::=
  'CREATE' 'SHARDING' 'TABLE' 'RULE' ( tableDefinition | autoTableDefinition ) ( ',
' ( tableDefinition | autoTableDefinition ) )*

tableDefinition ::=
   tableName '(' 'DATANODES' '(' dataNode ( ',' dataNode )* ')'  ( ','  'DATABASE_
STRATEGY' '(' strategyDefinition ')' )?  ( ','  'TABLE_STRATEGY' '('
strategyDefinition ')' )?  ( ','  'KEY_GENERATE_STRATEGY' '('
```

```
keyGenerateStrategyDefinition ')' )? ')'

autoTableDefinition ::=
    tableName '(' 'RESOURCES' '(' resourceName ( ',' resourceName )*  ')' ','
'SHARDING_COLUMN' '=' columnName ',' algorithmDefinition ( ','  'KEY_GENERATE_
STRATEGY' '(' keyGenerateStrategyDefinition ')' )?')'

strategyDefinition ::=
  'TYPE' '=' strategyType ',' ( 'SHARDING_COLUMN' | 'SHARDING_COLUMNS' ) '='
columnName ',' algorithmDefinition

keyGenerateStrategyDefinition ::=
  'KEY_GENERATE_STRATEGY' '(' 'COLUMN' '=' columnName ',' ( 'KEY_GENERATOR' '='
algorihtmName | algorithmDefinition ) ')'

algorithmDefinition ::=
  ('SHARDING_ALGORITHM' '=' algorithmName | 'TYPE' '(' 'NAME' '=' algorithmType (
',' 'PROPERTIES'  '(' propertyDefinition  ')' )?')'  )

propertyDefinition ::=
  ( key  '=' value ) ( ',' key  '=' value )*

tableName ::=
  identifier

resourceName ::=
  identifier

columnName ::=
  identifier

algorithmName ::=
  identifier
```

Supplement

- `tableDefinition` is defined for standard sharding table rule; `autoTableDefinition` is defined for auto sharding table rule. For standard sharding rules and auto sharding rule, refer to Data Sharding;
- use standard sharding table rule:
  - `DATANODES` can only use resources that have been added to the current database, and can only use INLINE expressions to specify required resources;
  - `DATABASE_STRATEGY`, `TABLE_STRATEGY` are the database sharding strategy and the table sharding strategy, which are optional, and the default strategy is used when not configured;
  - The attribute `TYPE` in `strategyDefinition` is used to specify the type of Sharding Algo-

rithm, currently only supports STANDARD, COMPLEX. Using COMPLEX requires specifying multiple sharding columns with SHARDING_COLUMNS.

- use auto sharding table rule:
  - RESOURCES can only use resources that have been added to the current database, and the required resources can be specified by enumeration or INLINE expression;
  - Only auto sharding algorithm can be used, please refer to Auto Sharding Algorithm.
- algorithmType is the sharding algorithm type, please refer to Sharding Algorithm;
- The auto-generated algorithm naming rule is tableName _ strategyType _ shardingAlgorithmType;
- The auto-generated primary key strategy naming rule is tableName _ strategyType;
- KEY_GENERATE_STRATEGY is used to specify the primary key generation strategy, which is optional. For the primary key generation strategy, please refer to Distributed Primary Key.

Example

1.Standard sharding table rule

- **Create standard sharding table rule by specifying sharding algorithms**

```
-- create sharding algorithms
CREATE SHARDING ALGORITHM database_inline (
    TYPE(NAME="inline", PROPERTIES("algorithm-expression"="t_order_${user_id % 2}
"))
), table_inline (
    TYPE(NAME="inline", PROPERTIES("algorithm-expression"="t_order_${order_id % 2}
"))
);

-- create a sharding rule by specifying sharding algorithms
CREATE SHARDING TABLE RULE t_order (
    DATANODES("ds_${0..1}.t_order_${0..1}"),
    DATABASE_STRATEGY(TYPE="standard", SHARDING_COLUMN=user_id, SHARDING_
ALGORITHM=database_inline),
    TABLE_STRATEGY(TYPE="standard", SHARDING_COLUMN=order_id, SHARDING_
ALGORITHM=table_inline)
);
```

- **Use the default sharding database strategy, create standard sharding table rule by specifying a sharding algorithm**

```
-- create sharding algorithms
CREATE SHARDING ALGORITHM database_inline (
    TYPE(NAME="inline", PROPERTIES("algorithm-expression"="t_order_${user_id % 2}
"))
), table_inline (
    TYPE(NAME="inline", PROPERTIES("algorithm-expression"="t_order_${order_id % 2}
"))
);

-- create a default sharding database strategy
CREATE DEFAULT SHARDING DATABASE STRATEGY (
    TYPE="standard", SHARDING_COLUMN=order_id, SHARDING_ALGORITHM=database_inline
);

-- create a sharding table rule by specifying a sharding algorithm
CREATE SHARDING TABLE RULE t_order (
    DATANODES("ds_${0..1}.t_order_${0..1}"),
    TABLE_STRATEGY(TYPE="standard", SHARDING_COLUMN=order_id, SHARDING_
ALGORITHM=table_inline)
);
```

- **Use both the default sharding and the default sharding strategy, create standard sharding table rule**

```
-- create sharding algorithms
CREATE SHARDING ALGORITHM database_inline (
    TYPE(NAME="inline", PROPERTIES("algorithm-expression"="t_order_${user_id % 2}
"))
), table_inline (
    TYPE(NAME="inline", PROPERTIES("algorithm-expression"="t_order_${order_id % 2}
"))
);

-- create a default sharding database strategy
CREATE DEFAULT SHARDING DATABASE STRATEGY (
    TYPE="standard", SHARDING_COLUMN=order_id, SHARDING_ALGORITHM=database_inline
);

-- create a default sharding table strategy
CREATE DEFAULT SHARDING TABLE STRATEGY (
    TYPE="standard", SHARDING_COLUMN=order_id, SHARDING_ALGORITHM=table_inline
);

-- create a sharding table rule
```

```
CREATE SHARDING TABLE RULE t_order (
    DATANODES("ds_${0..1}.t_order_${0..1}")
);
```

- **Create standard sharding table rule and sharding algorithms at the same time**

```
CREATE SHARDING TABLE RULE t_order (
    DATANODES("ds_${0..1}.t_order_${0..1}"),
    DATABASE_STRATEGY(TYPE="standard", SHARDING_COLUMN=user_id, SHARDING_
ALGORITHM(TYPE(NAME="inline", PROPERTIES("algorithm-expression"="ds_${user_id % 2}
")))),
    TABLE_STRATEGY(TYPE="standard", SHARDING_COLUMN=user_id, SHARDING_
ALGORITHM(TYPE(NAME="inline", PROPERTIES("algorithm-expression"="ds_${order_id % 2}
"))))
);
```

2.Auto sharding table rule

- **create auto sharding table rule**

```
CREATE SHARDING TABLE RULE t_order (
    RESOURCES(ds_0, ds_1),
    SHARDING_COLUMN=order_id, TYPE(NAME="MOD", PROPERTIES("sharding-count"="4"))
);
```

Reserved word

CREATE, SHARDING, TABLE, RULE, DATANODES, DATABASE_STRATEGY, TABLE_STRATEGY, KEY_GENERATE_STRATEGY, RESOURCES, SHARDING_COLUMN, TYPE, SHARDING_COLUMN, KEY_GENERATOR, SHARDING_ALGORITHM, COLUMN, NAME, PROPERTIES

Related links

- Reserved word
- CREATE SHARDING ALGORITHM
- CREATE DEFAULT_SHARDING STRATEGY

ALTER SHARDING TABLE RULE

Description

The ALTER SHARDING TABLE RULE syntax is used to alter sharding table rule for the currently selected database

Syntax

```
AlterShardingTableRule ::=
  'ALTER' 'SHARDING' 'TABLE' 'RULE' ( tableDefinition | autoTableDefinition ) ( ','
( tableDefinition | autoTableDefinition ) )*

tableDefinition ::=
   tableName '(' 'DATANODES' '(' dataNode ( ',' dataNode )* ')'  ( ','  'DATABASE_
STRATEGY' '(' strategyDefinition ')' )?  ( ','  'TABLE_STRATEGY' '('
strategyDefinition ')' )?  ( ','  'KEY_GENERATE_STRATEGY' '('
keyGenerateStrategyDefinition ')' )? ')'

autoTableDefinition ::=
    tableName '(' 'RESOURCES' '(' resourceName ( ',' resourceName )*  ')' ','
'SHARDING_COLUMN' '=' columnName ',' algorithmDefinition ( ','  'KEY_GENERATE_
STRATEGY' '(' keyGenerateStrategyDefinition ')' )?')'

strategyDefinition ::=
  'TYPE' '=' strategyType ',' ( 'SHARDING_COLUMN' | 'SHARDING_COLUMNS' ) '='
columnName ',' algorithmDefinition

keyGenerateStrategyDefinition ::=
  'KEY_GENERATE_STRATEGY' '(' 'COLUMN' '=' columnName ',' ( 'KEY_GENERATOR' '='
algorihtmName | algorithmDefinition ) ')'

algorithmDefinition ::=
  ('SHARDING_ALGORITHM' '=' algorithmName | 'TYPE' '(' 'NAME' '=' algorithmType (
',' 'PROPERTIES'  '(' propertyDefinition  ')' )?')'  )

propertyDefinition ::=
  ( key  '=' value ) ( ',' key  '=' value )*

tableName ::=
  identifier

resourceName ::=
  identifier

columnName ::=
  identifier
```

```
algorithmName ::=
  identifier
```

Supplement

- `tableDefinition` is defined for standard sharding table rule; `autoTableDefinition` is defined for auto sharding table rule. For standard sharding rules and auto sharding rule, refer to Data Sharding;
- use standard sharding table rule:
  - `DATANODES` can only use resources that have been added to the current database, and can only use INLINE expressions to specify required resources;
  - `DATABASE_STRATEGY`, `TABLE_STRATEGY` are the database sharding strategy and the table sharding strategy, which are optional, and the default strategy is used when not configured;
  - The attribute `TYPE` in `strategyDefinition` is used to specify the type of Sharding Algorithm, currently only supports `STANDARD`, `COMPLEX`. Using `COMPLEX` requires specifying multiple sharding columns with `SHARDING_COLUMNS`.
- use auto sharding table rule:
  - `RESOURCES` can only use resources that have been added to the current database, and the required resources can be specified by enumeration or INLINE expression;
  - Only auto sharding algorithm can be used, please refer to Auto Sharding Algorithm.
- `algorithmType` is the sharding algorithm type, please refer to Sharding Algorithm;
- The auto-generated algorithm naming rule is `tableName _ strategyType _ shardingAlgorithmType`;
- The auto-generated primary key strategy naming rule is `tableName _ strategyType`;
- `KEY_GENERATE_STRATEGY` is used to specify the primary key generation strategy, which is optional. For the primary key generation strategy, please refer to Distributed Primary Key.

Example

1.Standard sharding table rule

- **Alter standard sharding table rule to the specified sharding algorithms being altered**

```sql
-- alter sharding algorithms
ALTER SHARDING ALGORITHM database_inline (
    TYPE(NAME="inline", PROPERTIES("algorithm-expression"="t_order_${user_id % 4}
"))
), table_inline (
```

```
    TYPE(NAME="inline", PROPERTIES("algorithm-expression"="t_order_${order_id % 4}
"))
);

-- alter a sharding rule to the specified sharding algorithms being altered
ALTER SHARDING TABLE RULE t_order (
    DATANODES("resource_${0..3}.t_order_item${0..3}"),
    DATABASE_STRATEGY(TYPE="standard", SHARDING_COLUMN=user_id, SHARDING_
ALGORITHM=database_inline),
    TABLE_STRATEGY(TYPE="standard", SHARDING_COLUMN=order_id, SHARDING_
ALGORITHM=table_inline)
);
```

- **Use the altered default sharding database strategy, alter standard sharding table rule to the specified sharding algorithm being altered**

```
-- alter sharding algorithms
ALTER SHARDING ALGORITHM database_inline (
    TYPE(NAME="inline", PROPERTIES("algorithm-expression"="t_order_${user_id % 4}
"))
), table_inline (
    TYPE(NAME="inline", PROPERTIES("algorithm-expression"="t_order_${order_id % 4}
"))
);

-- alter a default sharding database strategy
ALTER DEFAULT SHARDING DATABASE STRATEGY (
    TYPE="standard", SHARDING_COLUMN=order_id, SHARDING_ALGORITHM=database_inline
);

-- alter a sharding table rule to the specified sharding algorithm being altered
ALTER SHARDING TABLE RULE t_order (
    DATANODES("resource_${0..3}.t_order_item${0..3}"),
    TABLE_STRATEGY(TYPE="standard", SHARDING_COLUMN=order_id, SHARDING_
ALGORITHM=table_inline)
);
```

- **Use both the altered default sharding and the altered default sharding strategy, alter standard sharding table rule**

```
-- alter sharding algorithms
ALTER SHARDING ALGORITHM database_inline (
    TYPE(NAME="inline", PROPERTIES("algorithm-expression"="t_order_${user_id % 4}
"))
), table_inline (
```

```
    TYPE(NAME="inline", PROPERTIES("algorithm-expression"="t_order_${order_id % 4}
"))
);

-- alter a default sharding database strategy
ALTER DEFAULT SHARDING DATABASE STRATEGY (
    TYPE="standard", SHARDING_COLUMN=order_id, SHARDING_ALGORITHM=database_inline
);

-- alter a default sharding table strategy
ALTER DEFAULT SHARDING TABLE STRATEGY (
    TYPE="standard", SHARDING_COLUMN=order_id, SHARDING_ALGORITHM=table_inline
);

-- alter a sharding table rule
ALTER SHARDING TABLE RULE t_order (
    DATANODES("resource_${0..3}.t_order_item${0..3}")
);
```

- **Alter standard sharding table rule and create sharding algorithms at the same time**

```
ALTER SHARDING TABLE RULE t_order (
    DATANODES("ds_${0..1}.t_order_${0..1}"),
    DATABASE_STRATEGY(TYPE="standard", SHARDING_COLUMN=user_id, SHARDING_
ALGORITHM(TYPE(NAME="inline", PROPERTIES("algorithm-expression"="ds_${user_id % 2}
")))),
    TABLE_STRATEGY(TYPE="standard", SHARDING_COLUMN=user_id, SHARDING_
ALGORITHM(TYPE(NAME="inline", PROPERTIES("algorithm-expression"="ds_${order_id % 2}
"))))
);
```

2.Auto sharding table rule

- **alter auto sharding table rule**

```
ALTER SHARDING TABLE RULE t_order (
    RESOURCES(ds_0, ds_1),
    SHARDING_COLUMN=order_id, TYPE(NAME="MOD", PROPERTIES("sharding-count"="4"))
);
```

Reserved word

ALTER, SHARDING, TABLE, RULE, DATANODES, DATABASE_STRATEGY, TABLE_STRATEGY, KEY_GENERATE_STRATEGY, RESOURCES, SHARDING_COLUMN, TYPE, SHARDING_COLUMN, KEY_GENERATOR, SHARDING_ALGORITHM, COLUMN, NAME, PROPERTIES

Related links

- Reserved word
- ALTER SHARDING ALGORITHM
- ALTER DEFAULT_SHARDING STRATEGY

CREATE SHARDING ALGORITHM

Description

The CREATE SHARDING ALGORITHM syntax is used to create a sharding algorithm for the currently selected database.

Syntax

```
CreateShardingAlgorithm ::=
  'CREATE' 'SHARDING' 'ALGORITHM' shardingAlgorithmName '(' algorithmDefinition ')'

algorithmDefinition ::=
  'TYPE' '(' 'NAME' '=' algorithmType ( ',' 'PROPERTIES'  '(' propertyDefinition
')' )?')'

propertyDefinition ::=
  ( key  '=' value ) ( ',' key  '=' value )*

shardingAlgorithmName ::=
  identifier

algorithmType ::=
  identifier
```

Supplement

- algorithmType is the sharding algorithm type. For detailed sharding algorithm type information, please refer to Sharding Algorithm.

Example

1.Create sharding algorithms

```
-- create a sharding algorithm of type INLINE
CREATE SHARDING ALGORITHM inline_algorithm (
    TYPE(NAME="inline", PROPERTIES("algorithm-expression"="t_order_${user_id % 2}
"))
);

-- create a sharding algorithm of type AUTO_INTERVAL
CREATE SHARDING ALGORITHM interval_algorithm (
    TYPE(NAME="auto_interval", PROPERTIES("datetime-lower"="2022-01-01 00:00:00",
"datetime-upper"="2022-01-03 00:00:00", "sharding-seconds"="86400"))
);
```

Reserved word

CREATE, SHARDING, ALGORITHM, TYPE, NAME, PROPERTIES

Related links

- Reserved word

CREATE DEFAULT SHARDING STRATEGY

Description

The CREATE DEFAULT SHARDING STRATEGY syntax is used to create a default sharding strategy

Syntax

```
CreateDefaultShardingStrategy ::=
  'CREATE' 'DEFAULT' 'SHARDING' ('DATABASE' | 'TABLE') 'STRATEGY' '('
shardingStrategy ')'

shardingStrategy ::=
  'TYPE' '=' strategyType ',' ( 'SHARDING_COLUMN' '=' columnName  | 'SHARDING_
COLUMNS' '=' columnNames ) ',' ( 'SHARDING_ALGORITHM' '=' algorithmName |
algorithmDefinition )

algorithmDefinition ::=
  'TYPE' '(' 'NAME' '=' algorithmType ( ',' 'PROPERTIES'  '(' propertyDefinition
')' )?')'

columnNames ::=
  columnName (',' columnName)+

columnName ::=
  identifier

algorithmName ::=
  identifier

algorithmType ::=
  identifier
```

Supplement

- When using the complex sharding algorithm, multiple sharding columns need to be specified using `SHARDING_COLUMNS`;
- `algorithmType` is the sharding algorithm type. For detailed sharding algorithm type information, please refer to Sharding Algorithm.

Example

1.Create a default sharding strategy by using an existing sharding algorithm

```
-- create a sharding algorithm
CREATE SHARDING ALGORITHM database_inline (
    TYPE(NAME="inline", PROPERTIES("algorithm-expression"="t_order_${order_id % 2}
"))
);

-- create a default sharding database strategy
```

```
CREATE DEFAULT SHARDING DATABASE STRATEGY (
    TYPE="standard", SHARDING_COLUMN=user_id, SHARDING_ALGORITHM=database_inline
);
```

2.Create sharding algorithm and default sharding table strategy at the same time

```
-- create a default sharding table strategy
CREATE DEFAULT SHARDING TABLE STRATEGY (
    TYPE="standard", SHARDING_COLUMN=user_id, SHARDING_ALGORITHM(TYPE(NAME="inline
", PROPERTIES("algorithm-expression"="t_order_${user_id % 2}")))
);
```

Reserved word

CREATE, DEFAULT, SHARDING, DATABASE, TABLE, STRATEGY, TYPE, SHARDING_COLUMN, SHARDING_COLUMNS, SHARDING_ALGORITHM, NAME, PROPERTIES

Related links

- Reserved word
- CREATE SHARDING ALGORITHM

CREATE SHARDING BINDING TABLE RULE

Description

The `CREATE SHARDING BINDING TABLE RULE` syntax is used to add binding relationships and create binding table rules for tables with sharding table rules

Syntax

```
CreateBindingTableRule ::=
  'CREATE' 'SHARDING' 'BINDING' 'TABLE' 'RULES'  bindingTableDefinition  (','
bindingTableDefinition )*

bindingTableDefinition ::=
  '(' tableName (',' tableName)* ')'

tableName ::=
  identifier
```

Supplement

- Creating binding relationships rules can only use sharding tables;
- A sharding table can only have one binding relationships;
- The sharding table for creating binding relationships needs to use the same resources and the same actual tables. For example `ds_${0..1}.t_order_${0..1}` 与 `ds_${0..1}.t_order_item_${0..1}`;
- The sharding table for creating binding relationships needs to use the same sharding algorithm for the sharding column. For example `t_order_{order_id % 2}` and `t_order_item_{order_item_id % 2}`;
- Only one binding rule can exist, but can contain multiple binding relationships, so can not execute `CREATE SHARDING BINDING TABLE RULE` more than one time. When a binding table rule already exists but a binding relationship needs to be added, you need to use `ALTER SHARDING BINDING TABLE RULE` to modify the binding table.

Example

1.Create a binding table rule

```
-- Before creating a binding table rule, you need to create sharding table rules t_
order, t_order_item
CREATE SHARDING BINDING TABLE RULES (t_order,t_order_item);
```

2.Create multiple binding table rules

```
-- Before creating binding table rules, you need to create sharding table rules t_
order, t_order_item, t_product, t_product_item
CREATE SHARDING BINDING TABLE RULES (t_order,t_order_item),(t_product,t_product_
item);
```

Reserved word

`CREATE`, `SHARDING`, `BINDING`, `TABLE`, `RULES`

Related links

- Reserved word
- CREATE SHARDING TABLE RULE

CREATE SHARDING BROADCAST TABLE RULE

Description

The `CREATE SHARDING BROADCAST TABLE RULE` syntax is used to create broadcast table rules for tables that need to be broadcast (broadcast tables)

Syntax

```
CreateBroadcastTableRule ::=
  'CREATE' 'SHARDING' 'BROADCAST' 'TABLE' 'RULES' '(' tableName (',' tableName)* ')
'

tableName ::=
  identifier
```

Supplement

- `tableName` can use an existing table or a table that will be created;
- Only one broadcast rule can exist, but can contain multiple broadcast tables, so can not execute `CREATE SHARDING BROADCAST TABLE RULE` more than one time. When the broadcast table rule already exists but the broadcast table needs to be added, you need to use `ALTER BROADCAST TABLE RULE` to modify the broadcast table rule.

Example

Create sharding broadcast table rule

```sql
-- Add t_province, t_city to broadcast table rules
CREATE SHARDING BROADCAST TABLE RULES (t_province, t_city);
```

Reserved word

CREATE, SHARDING, BROADCAST, TABLE, RULES

Related links

- Reserved word

CREATE SHARDING KEY GENERATOR

Description

The CREATE SHARDING KEY GENERATOR syntax is used to add a distributed primary key generator for the currently selected logic database

Syntax

```
CreateShardingAlgorithm ::=
  'CREATE' 'SHARDING' 'KEY' 'GENERATOR' keyGeneratorName '(' algorithmDefinition ')
'

algorithmDefinition ::=
  'TYPE' '(' 'NAME' '=' algorithmType ( ',' 'PROPERTIES'  '(' propertyDefinition
')' )?')'

propertyDefinition ::=
  ( key  '=' value ) ( ',' key  '=' value )*

keyGeneratorName ::=
  identifier

algorithmType ::=
  identifier
```

Supplement

- algorithmType is the key generate algorithm type. For detailed key generate algorithm type information, please refer to KEY GENERATE ALGORITHM.

Example

Create a distributed primary key generator

```
CREATE SHARDING KEY GENERATOR snowflake_key_generator (
    TYPE(NAME="SNOWFLAKE", PROPERTIES("max-vibration-offset"="3"))
);
```

Reserved word

CREATE, SHARDING, KEY, GENERATOR, TYPE, NAME, PROPERTIES

Related links

- Reserved word

Single Table

This chapter describes the syntax of single table.

CREATE DEFAULT SINGLE TABLE RULE

Description

The `CREATE DEFAULT SINGLE TABLE RULE` syntax is used to create a default single table rule.

Syntax

```
CreateDefaultSingleTableRule ::=
  'CREATE' 'DEFAULT' 'SINGLE' 'TABLE' 'RULE' singleTableDefinition

singleTableDefinition ::=
  'RESOURCE' '=' resourceName

resourceName ::=
  identifier
```

Supplement

- RESOURCE needs to use data source resource managed by RDL.

Example

Create a default single table rule

```
CREATE DEFAULT SINGLE TABLE RULE RESOURCE = ds_0;
```

Reserved word

CREATE, SHARDING, SINGLE, TABLE, RULE, RESOURCE

Related links

- Reserved word

RQL Syntax

RQL (Resource & Rule Query Language) responsible for resources/rules query.

Resource Query

This chapter describes the syntax of resource query.

SHOW RESOURCE

Description

The SHOW RESOURCE syntax is used to query the resources that have been added to the specified database.

Syntax

```
ShowResource ::=
  'SHOW' 'DATABASE' 'RESOURCES' ('FROM' databaseName)?

databaseName ::=
  identifier
```

Supplement

- When databaseName is not specified, the default is the currently used DATABASE; if DATABASE is not used, it will prompt No database selected.

Return Value Description

Column	Description
name	Data source name
type	Data source type
host	Data source host
port	Data source port
db	Database name
attribute	Data source attribute

Example

- Query resources for the specified database

```sql
SHOW DATABASE RESOURCES FROM sharding_db;
```

```
+------+-------+-----------+------+------+---------------------------------+-------
--------------------+---------------------------+---------------+---------------+--
---------+-------------------------------------------------------------------------
-----------------------------------------------------------------------------------
-----------------------------------------------------------------------------------
-----------------------------------------------------------------------------------
-----------------------------------------------------------------------------------
-----------------------------------------------------------------------------------
-----------------------------------------------------------------------------------
---------------------------------------------------------+
| name | type  | host      | port | db   | connection_timeout_milliseconds | idle_
timeout_milliseconds | max_lifetime_milliseconds | max_pool_size | min_pool_size |
read_only | other_attributes







                                                   |
+------+-------+-----------+------+------+---------------------------------+-------
--------------------+---------------------------+---------------+---------------+--
---------+-------------------------------------------------------------------------
-----------------------------------------------------------------------------------
-----------------------------------------------------------------------------------
-----------------------------------------------------------------------------------
-----------------------------------------------------------------------------------
```

```
-----------------------------------------------------------------------------------
-----------------------------------------------------------------------------------
----------------------------------------------------------+
| ds_0 | MySQL | 127.0.0.1 | 3306 | db_0 | 30000                           | 60000
                   | 1800000                     | 50              | 1             |
false      | {"dataSourceProperties":{"cacheServerConfiguration":"true",
"elideSetAutoCommits":"true","useServerPrepStmts":"true","cachePrepStmts":"true",
"rewriteBatchedStatements":"true","cacheResultSetMetadata":"false",
"useLocalSessionState":"true","maintainTimeStats":"false","prepStmtCacheSize":
"200000","tinyInt1isBit":"false","prepStmtCacheSqlLimit":"2048",
"zeroDateTimeBehavior":"round","netTimeoutForStreamingResults":"0"},
"healthCheckProperties":{},"initializationFailTimeout":1,"validationTimeout":5000,
"leakDetectionThreshold":0,"registerMbeans":false,"allowPoolSuspension":false,
"autoCommit":true,"isolateInternalQueries":false} |
| ds_1 | MySQL | 127.0.0.1 | 3306 | db_1 | 30000                           | 60000
                   | 1800000                     | 50              | 1             |
false      | {"dataSourceProperties":{"cacheServerConfiguration":"true",
"elideSetAutoCommits":"true","useServerPrepStmts":"true","cachePrepStmts":"true",
"rewriteBatchedStatements":"true","cacheResultSetMetadata":"false",
"useLocalSessionState":"true","maintainTimeStats":"false","prepStmtCacheSize":
"200000","tinyInt1isBit":"false","prepStmtCacheSqlLimit":"2048",
"zeroDateTimeBehavior":"round","netTimeoutForStreamingResults":"0"},
"healthCheckProperties":{},"initializationFailTimeout":1,"validationTimeout":5000,
"leakDetectionThreshold":0,"registerMbeans":false,"allowPoolSuspension":false,
"autoCommit":true,"isolateInternalQueries":false} |
+------+-------+-----------+------+------+---------------------------------+-------
--------------------+---------------------------+----------------+----------------+--
---------+-------------------------------------------------------------------------
-----------------------------------------------------------------------------------
-----------------------------------------------------------------------------------
-----------------------------------------------------------------------------------
-----------------------------------------------------------------------------------
-----------------------------------------------------------------------------------
-----------------------------------------------------------------------------------
----------------------------------------------------------+
2 rows in set (0.26 sec)
```

- Query resources for the current database

```sql
SHOW DATABASE RESOURCES;
```

```
+------+-------+-----------+------+------+---------------------------------+-------
--------------------+---------------------------+----------------+----------------+--
---------+-------------------------------------------------------------------------
-----------------------------------------------------------------------------------
-----------------------------------------------------------------------------------
-----------------------------------------------------------------------------------
-----------------------------------------------------------------------------------
```

```
----------------------------------------------------------------------------------
----------------------------------------------------------------------------------
---------------------------------------------------------+
| name | type  | host      | port | db   | connection_timeout_milliseconds | idle_
timeout_milliseconds | max_lifetime_milliseconds | max_pool_size | min_pool_size |
read_only | other_attributes









                                                   |
+-------+-------+-----------+------+------+---------------------------------+-------
--------------------+---------------------------+---------------+---------------+--
---------+------------------------------------------------------------------------
----------------------------------------------------------------------------------
----------------------------------------------------------------------------------
----------------------------------------------------------------------------------
----------------------------------------------------------------------------------
----------------------------------------------------------------------------------
----------------------------------------------------------------------------------
---------------------------------------------------------+
| ds_0 | MySQL | 127.0.0.1 | 3306 | db_0 | 30000                           | 60000
                   | 1800000                   | 50            | 1             |
false     | {"dataSourceProperties":{"cacheServerConfiguration":"true",
"elideSetAutoCommits":"true","useServerPrepStmts":"true","cachePrepStmts":"true",
"rewriteBatchedStatements":"true","cacheResultSetMetadata":"false",
"useLocalSessionState":"true","maintainTimeStats":"false","prepStmtCacheSize":
"200000","tinyInt1isBit":"false","prepStmtCacheSqlLimit":"2048",
"zeroDateTimeBehavior":"round","netTimeoutForStreamingResults":"0"},
"healthCheckProperties":{},"initializationFailTimeout":1,"validationTimeout":5000,
"leakDetectionThreshold":0,"registerMbeans":false,"allowPoolSuspension":false,
"autoCommit":true,"isolateInternalQueries":false} |
| ds_1 | MySQL | 127.0.0.1 | 3306 | db_1 | 30000                           | 60000
                   | 1800000                   | 50            | 1             |
false     | {"dataSourceProperties":{"cacheServerConfiguration":"true",
"elideSetAutoCommits":"true","useServerPrepStmts":"true","cachePrepStmts":"true",
"rewriteBatchedStatements":"true","cacheResultSetMetadata":"false",
"useLocalSessionState":"true","maintainTimeStats":"false","prepStmtCacheSize":
"200000","tinyInt1isBit":"false","prepStmtCacheSqlLimit":"2048",
"zeroDateTimeBehavior":"round","netTimeoutForStreamingResults":"0"},
"healthCheckProperties":{},"initializationFailTimeout":1,"validationTimeout":5000,
"leakDetectionThreshold":0,"registerMbeans":false,"allowPoolSuspension":false,
"autoCommit":true,"isolateInternalQueries":false} |
+-------+-------+-----------+------+------+---------------------------------+-------
--------------------+---------------------------+---------------+---------------+--
---------+------------------------------------------------------------------------
```

```
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
---------------------------------------------------------+
2 rows in set (0.26 sec)
```

SHOW UNUSED RESOURCE

Description

The `SHOW UNUSED RESOURCE` syntax is used to query resources in the specified database that have not been referenced by rules.

Syntax

```
ShowUnusedResource ::=
  'SHOW' 'UNUSED' 'DATABASE'? 'RESOURCES' ('FROM' databaseName)?

databaseName ::=
  identifier
```

Supplement

- When `databaseName` is not specified, the default is the currently used `DATABASE`; if `DATABASE` is not used, it will prompt `No database selected`.

Return Value Description

Column	Description
name	Data source name
type	Data source type
host	Data source host
port	Data source port
db	Database name
attribute	Data source attribute

Example

- Query resources for the specified database

```sql
SHOW UNUSED DATABASE RESOURCES FROM sharding_db;
```

```
+------+-------+-----------+------+------+---------------------------------+-------
--------------------+---------------------------+---------------+---------------+--
---------+-------------------------------------------------------------------------
-----------------------------------------------------------------------------------
-----------------------------------------------------------------------------------
-----------------------------------------------------------------------------------
-----------------------------------------------------------------------------------
-----------------------------------------------------------------------------------
-----------------------------------------------------------------------------------
---------------------------------------------------------+
| name | type  | host      | port | db   | connection_timeout_milliseconds | idle_
timeout_milliseconds | max_lifetime_milliseconds | max_pool_size | min_pool_size |
read_only | other_attributes






                                                    |
+------+-------+-----------+------+------+---------------------------------+-------
--------------------+---------------------------+---------------+---------------+--
---------+-------------------------------------------------------------------------
-----------------------------------------------------------------------------------
-----------------------------------------------------------------------------------
-----------------------------------------------------------------------------------
-----------------------------------------------------------------------------------
-----------------------------------------------------------------------------------
-----------------------------------------------------------------------------------
---------------------------------------------------------+
| ds_0 | MySQL | 127.0.0.1 | 3306 | db_0 | 30000                           | 60000
                   | 1800000                    | 50            | 1             |
false     | {"dataSourceProperties":{"cacheServerConfiguration":"true",
"elideSetAutoCommits":"true","useServerPrepStmts":"true","cachePrepStmts":"true",
"rewriteBatchedStatements":"true","cacheResultSetMetadata":"false",
"useLocalSessionState":"true","maintainTimeStats":"false","prepStmtCacheSize":
"200000","tinyInt1isBit":"false","prepStmtCacheSqlLimit":"2048",
"zeroDateTimeBehavior":"round","netTimeoutForStreamingResults":"0"},
"healthCheckProperties":{},"initializationFailTimeout":1,"validationTimeout":5000,
"leakDetectionThreshold":0,"registerMbeans":false,"allowPoolSuspension":false,
"autoCommit":true,"isolateInternalQueries":false} |
+------+-------+-----------+------+------+---------------------------------+-------
--------------------+---------------------------+---------------+---------------+--
---------+-------------------------------------------------------------------------
-----------------------------------------------------------------------------------
-----------------------------------------------------------------------------------
```

```
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
-------------------------------------------------------+
1 rows in set (0.26 sec)
```

- Query resources for the current database

```sql
SHOW UNUSED DATABASE RESOURCES;
```

```
+------+-------+-----------+------+------+---------------------------------+-------
--------------------+---------------------------+---------------+---------------+--
---------+----------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
-------------------------------------------------------+
| name | type  | host      | port | db   | connection_timeout_milliseconds | idle_
timeout_milliseconds | max_lifetime_milliseconds | max_pool_size | min_pool_size |
read_only | other_attributes

                                                   |
+------+-------+-----------+------+------+---------------------------------+-------
--------------------+---------------------------+---------------+---------------+--
---------+----------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
-------------------------------------------------------+
| ds_0 | MySQL | 127.0.0.1 | 3306 | db_0 | 30000                           | 60000
                   | 1800000                   | 50            | 1             |
false     | {"dataSourceProperties":{"cacheServerConfiguration":"true",
"elideSetAutoCommits":"true","useServerPrepStmts":"true","cachePrepStmts":"true",
"rewriteBatchedStatements":"true","cacheResultSetMetadata":"false",
"useLocalSessionState":"true","maintainTimeStats":"false","prepStmtCacheSize":
"200000","tinyInt1isBit":"false","prepStmtCacheSqlLimit":"2048",
```

```
"zeroDateTimeBehavior":"round","netTimeoutForStreamingResults":"0"},
"healthCheckProperties":{},"initializationFailTimeout":1,"validationTimeout":5000,
"leakDetectionThreshold":0,"registerMbeans":false,"allowPoolSuspension":false,
"autoCommit":true,"isolateInternalQueries":false} |
+-------+-------+-----------+------+------+---------------------------------+-------
--------------------+---------------------------+---------------+---------------+--
---------+-------------------------------------------------------------------------
-----------------------------------------------------------------------------------
-----------------------------------------------------------------------------------
-----------------------------------------------------------------------------------
-----------------------------------------------------------------------------------
-----------------------------------------------------------------------------------
-----------------------------------------------------------------------------------
---------------------------------------------------------+
1 rows in set (0.26 sec)
```

SHOW RULES USED RESOURCE

Description

The `SHOW RULES USED RESOURCE` syntax is used to query the rules that use the specified resource in the specified database.

Syntax

```
showRulesUsedResource ::=
  'SHOW' 'RULES' 'USED' 'RESOURCES' resourceName ('FROM' databaseName)?

resourceName ::=
  IDENTIFIER | STRING

databaseName ::=
  IDENTIFIER
```

Supplement

- When `databaseName` is not specified, the default is the currently used `DATABASE`; if `DATABASE` is not used, it will prompt `No database selected`.

Return Value Description

Column	Description
type	features
name	Data source name

Example

- Query the rules that use the specified resource in the specified database

```
SHOW RULES USED RESOURCE ds_0 FROM sharding_db;
```

```
+----------+--------------+
| type     | name         |
+----------+--------------+
| sharding | t_order      |
| sharding | t_order_item |
+----------+--------------+
2 rows in set (0.00 sec)
```

- Query the rules that use the specified resource in the current database

```
SHOW RULES USED RESOURCE ds_0;
```

```
+----------+--------------+
| type     | name         |
+----------+--------------+
| sharding | t_order      |
| sharding | t_order_item |
+----------+--------------+
2 rows in set (0.00 sec)
```

Rule Query

This chapter describes the syntax of rule query.

Sharding

This chapter describes the syntax of sharding.

SHOW SHARDING TABLE RULE

Description

The `SHOW SHARDING TABLE RULE` syntax is used to query the sharding table rule in the specified database.

Syntax

```
ShowShardingTableRule ::=
  'SHOW' 'SHARDING' 'TABLE' ('RULE' tableName | 'RULES') ('FROM' databaseName)?

tableName ::=
  identifier

databaseName ::=
  identifier
```

Supplement

- When databaseName is not specified, the default is the currently used DATABASE. If DATABASE is not used, No database selected will be prompted.

Return value description

Column	Description
table	Logical table name
actual_data_nodes	Actual data node
actual_data_sources	Actual data source (Displayed when creating rules by RDL)
database_strategy_type	Database sharding strategy type
d atabase_sharding_column	Database sharding column
database_ sharding_algorithm_type	Database sharding algorithm type
database_s harding_algorithm_props	Database sharding algorithm properties
table_strategy_type	Table sharding strategy type
table_sharding_column	Table sharding column
table_ sharding_algorithm_type	Table sharding algorithm type
table_s harding_algorithm_props	Table sharding algorithm properties
key_generate_column	Sharding key generator column
key_generator_type	Sharding key generator type
key_generator_props	Sharding key generator properties

Example - Query the sharding table rules of the specified logical database

```
SHOW SHARDING TABLE RULES FROM sharding_db;
```

```
+--------------+-------------------+---------------------+-----------------------
+-------------------------+-----------------------------------+-------------------
--------------+--------------------+-----------------------+---------------------
----------+---------------------------------+---------------------+---------------
---+---------------------+
| table        | actual_data_nodes | actual_data_sources | database_strategy_type |
database_sharding_column | database_sharding_algorithm_type | database_sharding_
algorithm_props | table_strategy_type | table_sharding_column | table_sharding_
algorithm_type | table_sharding_algorithm_props | key_generate_column | key_
generator_type | key_generator_props |
+--------------+-------------------+---------------------+-----------------------
+-------------------------+-----------------------------------+-------------------
--------------+--------------------+-----------------------+---------------------
----------+---------------------------------+---------------------+---------------
---+---------------------+
| t_order      |                   | ds_0,ds_1           |                        |
                         |                                  |
            | mod                | order_id              | mod
      | sharding-count=4                 |                      |                     |
```

```
                    |
| t_order_item |                   | ds_0,ds_1          |                        |
                         |                                  |
            | mod                 | order_id               | mod
      | sharding-count=4                |                     |                    |
                    |
+--------------+-------------------+---------------------+------------------------
+--------------------------+-----------------------------------+--------------------
---------------+--------------------+-----------------------+----------------------
----------+---------------------------------+---------------------+-----------------
---+---------------------+
2 rows in set (0.12 sec)
```

- Query the sharding table rules of the current logic database

```
SHOW SHARDING TABLE RULES;
```

```
+--------------+-------------------+---------------------+------------------------
+--------------------------+-----------------------------------+--------------------
---------------+--------------------+-----------------------+----------------------
----------+---------------------------------+---------------------+-----------------
---+---------------------+
| table        | actual_data_nodes | actual_data_sources | database_strategy_type |
database_sharding_column | database_sharding_algorithm_type | database_sharding_
algorithm_props | table_strategy_type | table_sharding_column | table_sharding_
algorithm_type | table_sharding_algorithm_props | key_generate_column | key_
generator_type | key_generator_props |
+--------------+-------------------+---------------------+------------------------
+--------------------------+-----------------------------------+--------------------
---------------+--------------------+-----------------------+----------------------
----------+---------------------------------+---------------------+-----------------
---+---------------------+
| t_order      |                   | ds_0,ds_1          |                        |
                         |                                  |
            | mod                 | order_id               | mod
      | sharding-count=4                |                     |                    |
                    |
| t_order_item |                   | ds_0,ds_1          |                        |
                         |                                  |
            | mod                 | order_id               | mod
      | sharding-count=4                |                     |                    |
                    |
+--------------+-------------------+---------------------+------------------------
+--------------------------+-----------------------------------+--------------------
---------------+--------------------+-----------------------+----------------------
----------+---------------------------------+---------------------+-----------------
---+---------------------+
2 rows in set (0.12 sec)
```

- Query the specified sharding table rule

```
SHOW SHARDING TABLE RULE t_order;
```

```
+--------------+-------------------+---------------------+------------------------
+--------------------------+-----------------------------------+--------------------
---------------+---------------------+----------------------+---------------------
----------+--------------------------------+---------------------+-----------------
---+---------------------+
| table        | actual_data_nodes | actual_data_sources | database_strategy_type |
database_sharding_column | database_sharding_algorithm_type | database_sharding_
algorithm_props | table_strategy_type | table_sharding_column | table_sharding_
algorithm_type | table_sharding_algorithm_props | key_generate_column | key_
generator_type | key_generator_props |
+--------------+-------------------+---------------------+------------------------
+--------------------------+-----------------------------------+--------------------
---------------+---------------------+----------------------+---------------------
----------+--------------------------------+---------------------+-----------------
---+---------------------+
| t_order      |                   | ds_0,ds_1           |                        |
                          |                                   |
            | mod                 | order_id             | mod
      | sharding-count=4                |                     |                    |
                    |
+--------------+-------------------+---------------------+------------------------
+--------------------------+-----------------------------------+--------------------
---------------+---------------------+----------------------+---------------------
----------+--------------------------------+---------------------+-----------------
---+---------------------+
1 rows in set (0.12 sec)
```

RAL Syntax

RAL (Resource & Rule Administration Language) responsible for the added-on feature of hint, transaction type switch, scaling, sharding execute planning and so on.

Reserved word

Resource Definition

ADD, ALTER, DROP, RESOURCE, IF, EXISTS, HOST, PORT, DB, USER, PASSWORD, PROPERTIES, URL , IGNORE, SINGLE, TABLES

Rule Definition

SHARDING

CREATE, DEFAULT, SHARDING, BROADCAST, BINDING, DATABASE, TABLE, STRATEGY, RULE, RULES, ALGORITHM , DATANODES, DATABASE_STRATEGY, TABLE_STRATEGY, KEY_GENERATE_STRATEGY, RESOURCES, SHARDING_COLUMN, KEY , GENERATOR, TYPE, SHARDING_COLUMNS, KEY_GENERATOR, SHARDING_ALGORITHM, COLUMN, NAME, PROPERTIES

Single Table

CREATE, SHARDING, SINGLE, TABLE, RULE, RESOURCE

Readwrite Splitting

CREATE, READWRITE_SPLITTING, RULE, WRITE_RESOURCE, READ_RESOURCES, AUTO_AWARE_RESOURCE , WRITE_DATA_SOURCE_QUERY_ENABLED, TYPE, NAME, PROPERTIES, TRUE, FALSE

Encrypt

CREATE, ENCRYPT, RULE, COLUMNS, NAME, CIPHER, PLAIN, QUERY_WITH_CIPHER_COLUMN, TYPE, TRUE, FALSE

Database Discovery

CREATE, DB_DISCOVERY, RULE, RESOURCES, TYPE, NAME, PROPERTIES, HEARTBEAT

Shadow

CREATE, SHARDING, SINGLE, TABLE, RULE, RESOURCE

Supplement

- The above reserved words are not case-sensitive

9.11 Architecture

Apache ShardingSphere' s pluggable architecture is designed to enable developers to customize their own unique systems by adding the desired features, just like adding building blocks.

A plugin-oriented architecture has very high requirements for program architecture design. It requires making each module independent, and using a pluggable kernel to combine various functions in an overlapping manner. Designing an architecture system that completely isolates the feature development not only fosters an active open source community, but also ensures the quality of the project.

Apache ShardingSphere began to focus on the pluggable architecture since version 5.X, and the functional components of the project can be flexibly extended in a pluggable manner. Currently, features such as data sharding, read/write splitting, database high availability, data encryption, shadow DB stress testing, and support for SQL and protocols such as MySQL, PostgreSQL, SQLServer, Oracle, etc. are woven into the project through plugins. Apache ShardingSphere has provided dozens of SPIs (service provider interfaces) as extension points of the system, with the total number still increasing.

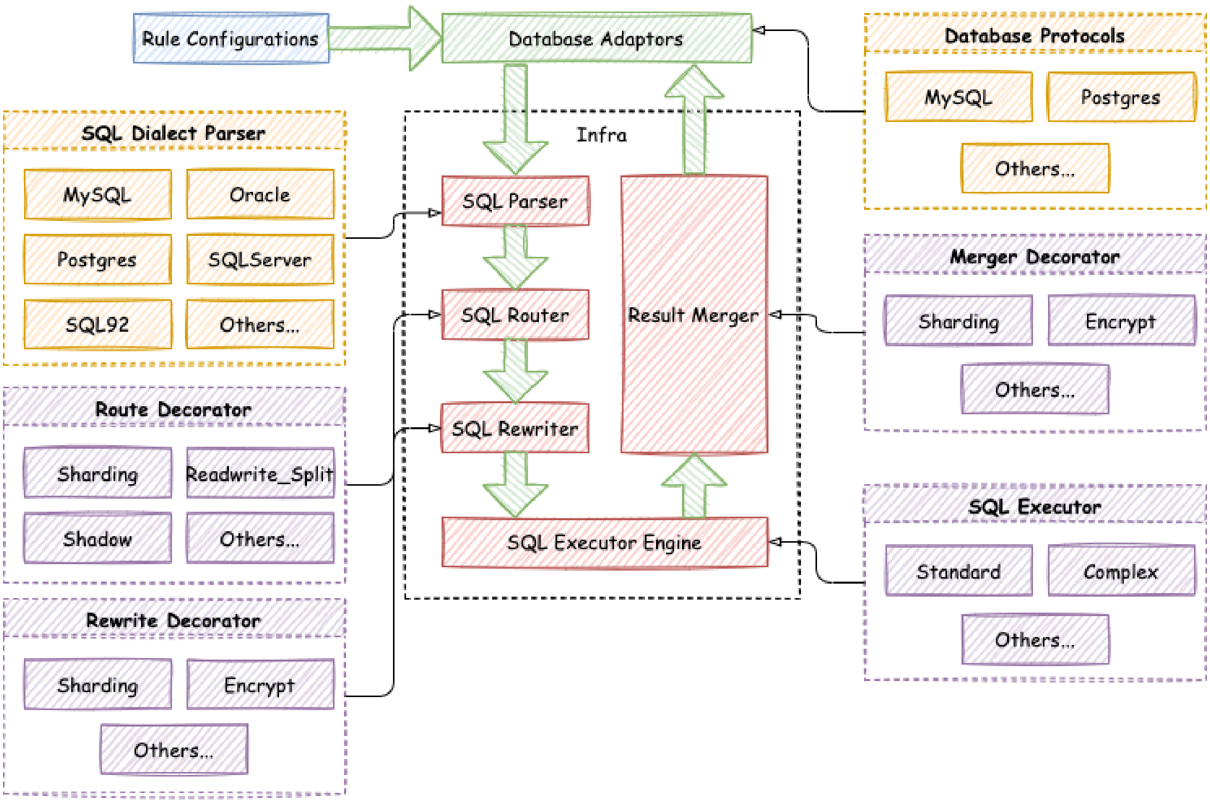


Figure30: Pluggable Platform

10

FAQ

##DBC

###JDBC] Why there may be an error when configure both shardingsphere-jdbc-spring-boot-starter and a spring-boot-starter of certain datasource pool (such as druid)?

Answer:

1. Because the spring-boot-starter of certain datasource pool (such as druid) will be configured before shardingsphere-jdbc-spring-boot-starter and create a default datasource, causing conflict to occur when ShardingSphere-JDBC create datasources.
2. A simple way to solve this issue is removing the spring-boot-starter of certain datasource pool, allowing shardingsphere-jdbc to create datasources with suitable pools.

###JDBC] Why is xsd unable to be found when Spring Namespace is used?

Answer:

The norm of Spring Namespace does not require deploying xsd files to the official website. But considering some users' needs, we will deploy them to ShardingSphere's official website. Actually, META-INF:raw-latex:*spring*.schemas in the jar package of shardingsphere-jdbc-spring-namespace has been configured with the position of xsd files: META-INF:raw-latex:*namespace***:raw-latex:`\sharding`**.xsd and META-INF:raw-latex:*namespace***:raw-latex:`\replica`**-query.xsd, so you only need to make sure that the file is in the jar package.

###JDBC] Found a JtaTransactionManager in spring boot project when integrating with XAtransaction.

Answer:

1. `shardingsphere-transaction-xa-core` include atomikos, it will trigger auto-configuration mechanism in spring-boot, add `@SpringBootApplication(exclude = JtaAutoConfiguration.class)` will solve it.

###JDBC] The tableName and columnName configured in yaml or properties leading incorrect result when loading Oracle metadata?

Answer:

Note that, in Oracle's metadata, the tableName and columnName is default UPPERCASE, while double-quoted such as `CREATE TABLE "TableName"("Id" number)` the tableName and columnName is the actual content double-quoted, refer to the following SQL for the reality in metadata:

```
SELECT OWNER, TABLE_NAME, COLUMN_NAME, DATA_TYPE FROM ALL_TAB_COLUMNS WHERE TABLE_
NAME IN ('TableName')
```

ShardingSphere uses the `OracleTableMetaDataLoader` to load the metadata, keep the tableName and columnName in the yaml or properties consistent with the metadata. ShardingSphere assembled the SQL using the following code:

```
private String getTableMetaDataSQL(final Collection<String> tables, final
DatabaseMetaData metaData) throws SQLException {
    StringBuilder stringBuilder = new StringBuilder(28);
    if (versionContainsIdentityColumn(metaData)) {
        stringBuilder.append(", IDENTITY_COLUMN");
    }
    if (versionContainsCollation(metaData)) {
        stringBuilder.append(", COLLATION");
    }
    String collation = stringBuilder.toString();
    return tables.isEmpty() ? String.format(TABLE_META_DATA_SQL, collation)
            : String.format(TABLE_META_DATA_SQL_IN_TABLES, collation, tables.
stream().map(each -> String.format("'%s'", each)).collect(Collectors.joining(",
")));
}
```

##roxy

###Proxy] In Windows environment, could not find or load main class org.apache.shardingsphere.proxy.Bootstrap, how to solve it?

Answer:

Some decompression tools may truncate the file name when decompressing the ShardingSphere-Proxy binary package, resulting in some classes not being found. The solutions: Open cmd.exe and execute the following command:

```
tar zxvf apache-shardingsphere-${RELEASE.VERSION}-shardingsphere-proxy-bin.tar.gz
```

###Proxy] How to add a new logic database dynamically when use ShardingSphere-Proxy?

Answer:

When using ShardingSphere-Proxy, users can dynamically create or drop logic database through `DistSQL`, the syntax is as follows:

```
CREATE DATABASE [IF NOT EXISTS] databaseName;
DROP DATABASE [IF EXISTS] databaseName;
```

Example:

```sql
CREATE DATABASE sharding_db;
DROP DATABASE sharding_db;
```

###Proxy] How to use suitable database tools connecting ShardingSphere-Proxy?

Answer:

1. ShardingSphere-Proxy could be considered as a MySQL server, so we recommend using MySQL command line tool to connect to and operate it.
2. If users would like to use a third-party database tool, there may be some errors cause of the certain implementation/options.
3. The currently tested third-party database tools are as follows:
   - Navicat: 11.1.13, 15.0.20.
   - DataGrip: 2020.1, 2021.1 (turn on "introspect using jdbc metadata" in idea or datagrip).
   - WorkBench: 8.0.25.

###Proxy] When using a client such as Navicat to connect to ShardingSphere-Proxy, if ShardingSphere-Proxy does not create a database or does not add a resource, the client connection will fail?

Answer:

1. Third-party database tools will send some SQL query metadata when connecting to ShardingSphere-Proxy. When ShardingSphere-Proxy does not create a `database` or does not add a `resource`, ShardingSphere-Proxy cannot execute SQL.
2. It is recommended to create `database` and `resource` first, and then use third-party database tools to connect.
3. Please refer to Related introduction the details about `resource`.

##harding

###Sharding] How to solve `Cloud not resolve placeholder …in string value …` error?

Answer:

`${...}` or `$->{...}` can be used in inline expression identifiers, but the former one clashes with place holders in Spring property files, so `$->{...}` is recommended to be used in Spring as inline expression identifiers.

###Sharding] Why does float number appear in the return result of inline expression?

Answer:

The division result of Java integers is also integer, but in Groovy syntax of inline expression, the division result of integers is float number. To obtain integer division result, A/B needs to be modified as A.intdiv(B).

###Sharding] If sharding database is partial, should tables without sharding database & table configured in sharding rules?

Answer:

No, ShardingSphere will recognize it automatically.

###Sharding] When generic Long type `SingleKeyTableShardingAlgorithm` is used, why does the `ClassCastException: Integer can not cast to Long` exception appear?

Answer:

You must make sure the field in the database table is consistent with that in the sharding algorithms. For example, the field type in database is int(11) and the sharding type corresponds to genetic type is Integer. If you want to configure Long type, please make sure the field type in the database is bigint.

###Sharding:raw-latex:*PROXY*] When implementing the `StandardShardingAlgorithm` custom algorithm, the specific type of `Comparable` is specified as Long, and the field type in the database table is bigint, a `ClassCastException: Integer can not cast to Long` exception occurs.

Answer：

When implementing the `doSharding` method, it is not recommended to specify the specific type of `Comparable` in the method declaration, but to convert the type in the implementation of the `doSharding` method. You can refer to the `ModShardingAlgorithm#doSharding` method.

###Sharding] Why is the default distributed auto-augment key strategy provided by ShardingSphere not continuous and most of them end with even numbers?

Answer:

ShardingSphere uses snowflake algorithms as the default distributed auto-augment key strategy to make sure unrepeated and decentralized auto-augment sequence is generated under the distributed situations. Therefore, auto-augment keys can be incremental but not continuous. But the last four numbers of snowflake algorithm are incremental value within one millisecond. Thus, if concurrency degree in one millisecond is not high, the last four numbers are likely to be zero, which explains why the rate of even end number is higher. In 3.1.0 version, the problem of ending with even numbers has been totally solved, please refer to: https://github.com/apache/shardingsphere/issues/1617

###Sharding] How to allow range query with using inline sharding strategy (BETWEEN AND, >, <, >=, <=)?

Answer:

1. Update to 4.1.0 above.
2. Configure(A tip here: then each range query will be broadcast to every sharding table):

- Version 4.x: `allow.range.query.with.inline.sharding` to `true` (Default value is `false`).
- Version 5.x: `allow-range-query-with-inline-sharding` to `true` in InlineShardingStrategy (Default value is `false`).

###Sharding] Why does my custom distributed primary key do not work after implementing `KeyGenerateAlgorithm` interface and configuring `type` property?

Answer:

Service Provider Interface (SPI) is a kind of API for the third party to implement or expand. Except implementing interface, you also need to create a corresponding file in `META-INF/services` to make the JVM load these SPI implementations. More detail for SPI usage, please search by yourself. Other ShardingSphere functionality implementation will take effect in the same way.

###Sharding] In addition to internal distributed primary key, does ShardingSphere support other native auto-increment keys?

Answer:

Yes. But there is restriction to the use of native auto-increment keys, which means they cannot be used as sharding keys at the same time. Since ShardingSphere does not have the database table structure and native auto-increment key is not included in original SQL, it cannot parse that field to the sharding field. If the auto-increment key is not sharding key, it can be returned normally and is needless to be cared. But if the auto-increment key is also used as sharding key, ShardingSphere cannot parse its sharding value, which will make SQL routed to multiple tables and influence the rightness of the application. The premise for returning native auto-increment key is that INSERT SQL is eventually routed to one table. Therefore, auto-increment key will return zero when INSERT SQL returns multiple tables.

##ncryption

###Encryption] How to solve that `data encryption` can't work with JPA?

Answer:

Because DDL for data encryption has not yet finished, JPA Entity cannot meet the DDL and DML at the same time, when JPA that automatically generates DDL is used with data encryption. The solutions are as follows: 1. Create JPA Entity with logicColumn which needs to encrypt. 2. Disable JPA auto-ddl, For example setting auto-ddl=none. 3. Create table manually. Table structure should use `cipherColumn`,`plainColumn` and `assistedQueryColumn` to replace the logicColumn.

##istSQL

###DistSQL] How to set custom JDBC connection properties or connection pool properties when adding a data source using DistSQL?

Answer:

1. If you need to customize JDBC connection properties, please take the `urlSource` way to define `dataSource`.
2. ShardingSphere presets necessary connection pool properties, such as `maxPoolSize`, `idleTimeout`, etc. If you need to add or overwrite the properties, please specify it with `PROPERTIES` in the `dataSource`.
3. Please refer to Related introduction for above rules.

###DistSQL] How to solve `Resource [xxx] is still used by [SingleTableRule].` exception when dropping a data source using DistSQL?

Answer:

1. Resources referenced by rules cannot be deleted
2. If the resource is only referenced by single table rule, and the user confirms that the restriction can be ignored, the optional parameter ignore single tables can be added to perform forced deletion

```
DROP RESOURCE dataSourceName [, dataSourceName] ... [ignore single tables]
```

###DistSQL] How to solve `Failed to get driver instance for jdbcURL=xxx.` exception when adding a data source using DistSQL?

Answer：

ShardingSphere Proxy do not have jdbc driver during deployment. Some example of this include `mysql-connector`. To use it otherwise following syntax can be used:

```
ADD RESOURCE dataSourceName [..., dataSourceName]
```

##ther

###Other] How to debug when SQL can not be executed rightly in ShardingSphere?

Answer:

`sql.show` configuration is provided in ShardingSphere-Proxy and post-1.5.0 version of ShardingSphere-JDBC, enabling the context parsing, rewritten SQL and the routed data source printed to info log. `sql.show` configuration is off in default, and users can turn it on in configurations. A Tip: Property `sql.show` has changed to `sql-show` in version 5.x.

###Other] Why do some compiling errors appear? Why did not the IDEA index the generated codes?

Answer:

ShardingSphere uses lombok to enable minimal coding. For more details about using and installment, please refer to the official website of lombok. The codes under the package `org.apache.shardingsphere.sql.parser.autogen` are generated by ANTLR. You may execute the following command to generate codes:

```
./mvnw -Dcheckstyle.skip=true -Drat.skip=true -Dmaven.javadoc.skip=true -Djacoco.
skip=true -DskipITs -DskipTests install -T1C
```

The generated codes such as `org.apache.shardingsphere.sql.parser.autogen.PostgreSQLStatementParser` may be too large to be indexed by the IDEA. You may configure the IDEA' s property `idea.max.intellisense.filesize=10000`.

###Other] In SQLSever and PostgreSQL, why does the aggregation column without alias throw exception?

Answer:

SQLServer and PostgreSQL will rename aggregation columns acquired without alias, such as the following SQL:

```
SELECT SUM(num), SUM(num2) FROM tablexxx;
```

Columns acquired by SQLServer are empty string and (2); columns acquired by PostgreSQL are empty sum and sum(2). It will cause error because ShardingSphere is unable to find the corresponding column. The right SQL should be written as:

```
SELECT SUM(num) AS sum_num, SUM(num2) AS sum_num2 FROM tablexxx;
```

###Other] Why does Oracle database throw "Order by value must implements Comparable" exception when using Timestamp Order By?

Answer:

There are two solutions for the above problem: 1. Configure JVM parameter "-oracle.jdbc.J2EE13Compliant=true" 2. Set System.getProperties().setProperty("oracle.jdbc.J2EE13Compliant", "true") codes in the initialization of the project. Reasons: `org.apache.shardingsphere.sharding.merge.dql.orderby.OrderByValue#getOrderValues()`:

```
private List<Comparable<?>> getOrderValues() throws SQLException {
    List<Comparable<?>> result = new ArrayList<>(orderByItems.size());
    for (OrderByItem each : orderByItems) {
        Object value = queryResult.getValue(each.getIndex(), Object.class);
        Preconditions.checkState(null == value || value instanceof Comparable,
"Order by value must implements Comparable");
        result.add((Comparable<?>) value);
    }
    return result;
}
```

After using resultSet.getObject(int index), for TimeStamp oracle, the system will decide whether to return java.sql.TimeStamp or define oralce.sql.TIMESTAMP according to the property of oracle.jdbc.J2EE13Compliant. See oracle.jdbc.driver.TimestampAccessor#getObject(int var1) method in ojdbc codes for more detail:

```
Object getObject(int var1) throws SQLException {
    Object var2 = null;
    if(this.rowSpaceIndicator == null) {
        DatabaseError.throwSqlException(21);
    }
    if(this.rowSpaceIndicator[this.indicatorIndex + var1] != -1) {
        if(this.externalType != 0) {
            switch(this.externalType) {
            case 93:
                return this.getTimestamp(var1);
            default:
                DatabaseError.throwSqlException(4);
                return null;
            }
        }
        if(this.statement.connection.j2ee13Compliant) {
            var2 = this.getTimestamp(var1);
        } else {
            var2 = this.getTIMESTAMP(var1);
        }
    }
```

```
    return var2;
}
```

###Other] In Windows environment,when cloning ShardingSphere source code through Git, why prompt filename too long and how to solve it?

Answer:

To ensure the readability of source code,the ShardingSphere Coding Specification requires that the naming of classes,methods and variables be literal and avoid abbreviations,which may result in some source files have long names. Since the Git version of Windows is compiled using msys,it uses the old version of Windows Api,limiting the file name to no more than 260 characters. The solutions are as follows: Open cmd.exe (you need to add git to environment variables) and execute the following command to allow git supporting log paths:

```
git config --global core.longpaths true
```

If we use windows 10, also need enable win32 log paths in registry editor or group strategy(need reboot): > Create the registry key `HKLM\SYSTEM\CurrentControlSet\Control\FileSystem LongPathsEnabled` (Type: REG_DWORD) in registry editor, and be set to 1. > Or click "setting" button in system menu, print "Group Policy" to open a new window "Edit Group Policy", and then click 'Computer Configuration' > 'Administrative Templates' > 'System' > 'Filesystem', and then turn on 'Enable Win32 long paths' option. Reference material: https://docs.microsoft.com/zh-cn/windows/desktop/FileIO/naming-a-file https://ourcodeworld.com/articles/read/109/how-to-solve-filename-too-long-error-in-git-powershell-and-github-application-for-windows

###Other] How to solve `Type is required` error?

Answer:

In Apache ShardingSphere, many functionality implementation are uploaded through SPI, such as Distributed Primary Key. These functions load SPI implementation by configuring the `type`, so the `type` must be specified in the configuration file.

###Other] How to speed up the metadata loading when service starts up?

Answer:

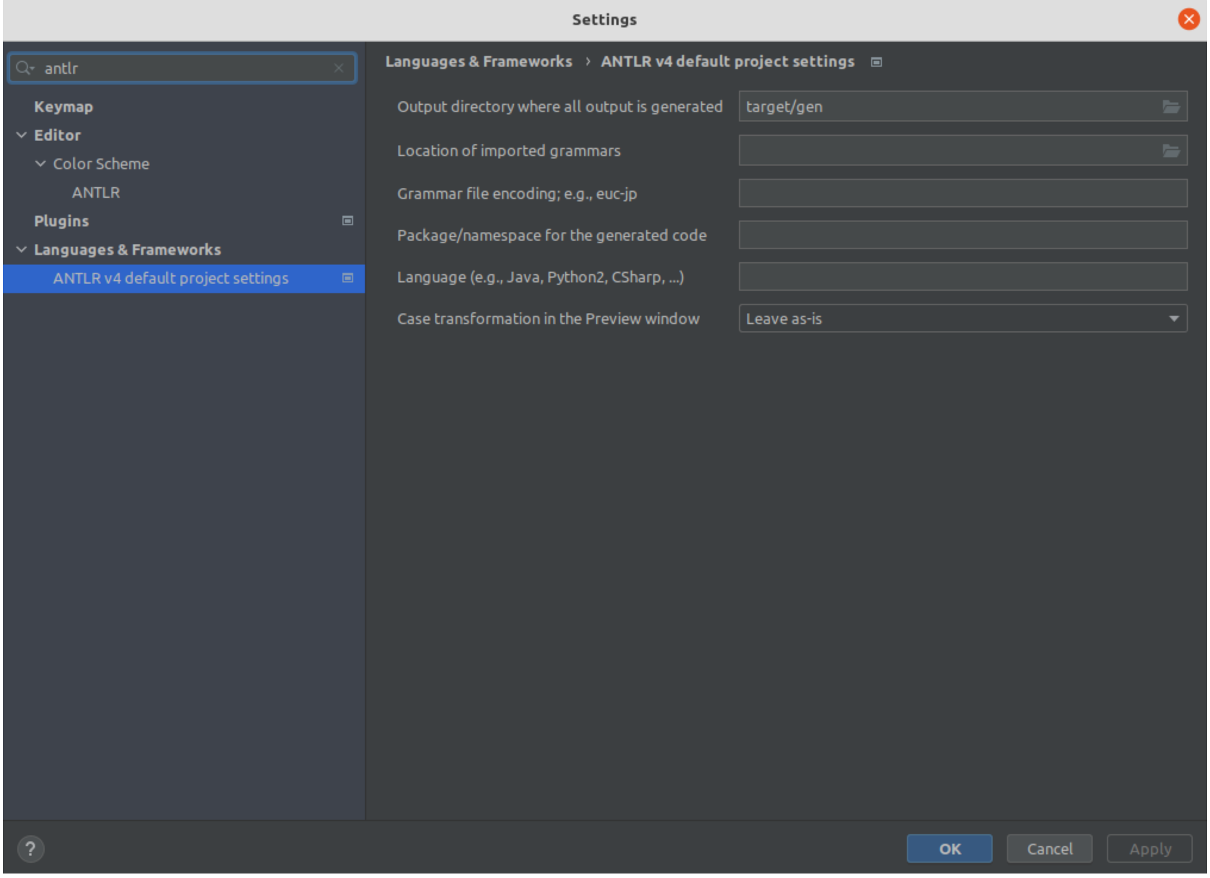
1. Update to 4.0.1 above, which helps speed up the process of loading table metadata.
2. Configure:

- `max.connections.size.per.query`(Default value is 1) higher referring to connection pool you adopt(Version >= 3.0.0.M3 & Version < 5.0.0).
- `max-connections-size-per-query`(Default value is 1) higher referring to connection pool you adopt(Version >= 5.0.0).

###Other] The ANTLR plugin generates codes in the same level directory as src, which is easy to commit by mistake. How to avoid it?

Answer:

Goto Settings -> Languages & Frameworks -> ANTLR v4 default project settings and configure the output directory of the generated code as `target/gen` as shown:



###Other] Why is the database sharding result not correct when using `Proxool`?

Answer:

When using `Proxool` to configure multiple data sources, each one of them should be configured with alias. It is because `Proxool` would check whether existing alias is included in the connection pool or not when acquiring connections, so without alias, each connection will be acquired from the same data source. The followings are core codes from ProxoolDataSource getConnection method in `Proxool`:

```
if(!ConnectionPoolManager.getInstance().isPoolExists(this.alias)) {
    this.registerPool();
}
```

For more alias usages, please refer to Proxool official website.

###Other] The property settings in the configuration file do not take effect when integrating ShardingSphere with Spring Boot 2.x ?

Answer:

Note that the property name in the Spring Boot 2.x environment is constrained to allow only lowercase letters, numbers and short transverse lines, `[a-z][0-9]` and `-`. Reasons: In the Spring Boot 2.x environment, ShardingSphere binds the properties through Binder, and the unsatisfied property name (such as camel case or underscore.) can throw a `NullPointerException` exception when the property setting does not work to check the property value. Refer to the following error examples:

Underscore case: database_inline

```
spring.shardingsphere.rules.sharding.sharding-algorithms.database_inline.
type=INLINE
spring.shardingsphere.rules.sharding.sharding-algorithms.database_inline.props.
algorithm-expression=ds-$->{user_id % 2}
```

```
Caused by: org.springframework.beans.factory.BeanCreationException: Error creating
bean with name 'database_inline': Initialization of bean failed; nested exception
is java.lang.NullPointerException: Inline sharding algorithm expression cannot be
null.
    ...
Caused by: java.lang.NullPointerException: Inline sharding algorithm expression
cannot be null.
    at com.google.common.base.Preconditions.checkNotNull(Preconditions.java:897)
    at org.apache.shardingsphere.sharding.algorithm.sharding.inline.
InlineShardingAlgorithm.getAlgorithmExpression(InlineShardingAlgorithm.java:58)
    at org.apache.shardingsphere.sharding.algorithm.sharding.inline.
InlineShardingAlgorithm.init(InlineShardingAlgorithm.java:52)
    at org.apache.shardingsphere.spring.boot.registry.
AbstractAlgorithmProvidedBeanRegistry.
postProcessAfterInitialization(AbstractAlgorithmProvidedBeanRegistry.java:98)
    at org.springframework.beans.factory.support.
AbstractAutowireCapableBeanFactory.
applyBeanPostProcessorsAfterInitialization(AbstractAutowireCapableBeanFactory.
java:431)
    ...
```

Camel case：databaseInline

```
spring.shardingsphere.rules.sharding.sharding-algorithms.databaseInline.type=INLINE
spring.shardingsphere.rules.sharding.sharding-algorithms.databaseInline.props.
algorithm-expression=ds-$->{user_id % 2}
```

```
Caused by: org.springframework.beans.factory.BeanCreationException: Error creating
bean with name 'databaseInline': Initialization of bean failed; nested exception is
java.lang.NullPointerException: Inline sharding algorithm expression cannot be
null.
    ...
Caused by: java.lang.NullPointerException: Inline sharding algorithm expression
cannot be null.
    at com.google.common.base.Preconditions.checkNotNull(Preconditions.java:897)
    at org.apache.shardingsphere.sharding.algorithm.sharding.inline.
InlineShardingAlgorithm.getAlgorithmExpression(InlineShardingAlgorithm.java:58)
    at org.apache.shardingsphere.sharding.algorithm.sharding.inline.
InlineShardingAlgorithm.init(InlineShardingAlgorithm.java:52)
    at org.apache.shardingsphere.spring.boot.registry.
AbstractAlgorithmProvidedBeanRegistry.
```

```
postProcessAfterInitialization(AbstractAlgorithmProvidedBeanRegistry.java:98)
    at org.springframework.beans.factory.support.
AbstractAutowireCapableBeanFactory.
applyBeanPostProcessorsAfterInitialization(AbstractAutowireCapableBeanFactory.
java:431)
    ...
```

From the exception stack, the `AbstractAlgorithmProvidedBeanRegistry.registerBean` method calls `PropertyUtil.containPropertyPrefix (environment, prefix)`, and `PropertyUtil.containPropertyPrefix (environment, prefix)` determines that the configuration of the specified prefix does not exist, while the method uses Binder in an unsatisfied property name (such as camelcase or underscore) causing property settings does not to take effect.

11 Downloads

11.1 Latest Releases

Apache ShardingSphere is released as source code tarballs with corresponding binary tarballs for convenience. The downloads are distributed via mirror sites and should be checked for tampering using GPG or SHA-512.

11.1.1 Apache ShardingSphere - Version: 5.1.2 (Release Date: June 17th, 2022)

- Source Codes: SRC (ASC, SHA512)
- ShardingSphere-JDBC Binary Distribution: TAR (ASC, SHA512)
- ShardingSphere-Proxy Binary Distribution: TAR (ASC, SHA512)
- ShardingSphere-Agent Binary Distribution: TAR (ASC, SHA512)

11.2 All Releases

Find all releases in the Archive repository. Find all incubator releases in the Archive incubator repository.

11.3 Verify the Releases

PGP signatures KEYS

It is essential that you verify the integrity of the downloaded files using the PGP or SHA signatures. The PGP signatures can be verified using GPG or PGP. Please download the KEYS as well as the asc signature files for relevant distribution. It is recommended to get these files from the main distribution directory and not from the mirrors.

```
gpg -i KEYS
```

or

```
pgpk -a KEYS
```

or

```
pgp -ka KEYS
```

To verify the binaries/sources you can download the relevant asc files for it from main distribution directory and follow the below guide.

```
gpg --verify apache-shardingsphere-********.asc apache-shardingsphere-*********
```

or

```
pgpv apache-shardingsphere-********.asc
```

or

```
pgp apache-shardingsphere-********.asc
```