Apache ShardingSphere document

Apache ShardingSphere

Nov 17, 2021

Contents

Stargazers Over Time

Contributors Over Time

Apache ShardingSphere is positioned as a Database Plus, and aims at building a new criterion and ecosystem above multi-model databases. It focuses on how to reuse existing databases and their respective upper layer, rather than creating a new database.

The concepts at the core of the project are Link, Enhance and Pluggable.

- `Link:` Flexible adaptation of database protocol, SQL dialect and database storage. It can quickly link applications and multi-mode heterogeneous databases quickly.
- `Enhance:` Capture database access entry to provide additional features transparently, such as: redirect (sharding, readwrite-splitting and shadow), transform (data encrypt and mask), authentication (security, audit and authority), governance (circuit breaker and access limitation and analyze, QoS and observability).
- `Pluggable:` Leveraging the micro kernel and 3 layers pluggable mode, features and database ecosystem can be embedded flexibily. Developers can customize their ShardingSphere just like building with LEGO blocks.

ShardingSphere became an Apache Top-Level Project on April 16, 2020.

Welcome to interact with community via the official mail list and the ShardingSphere Slack.

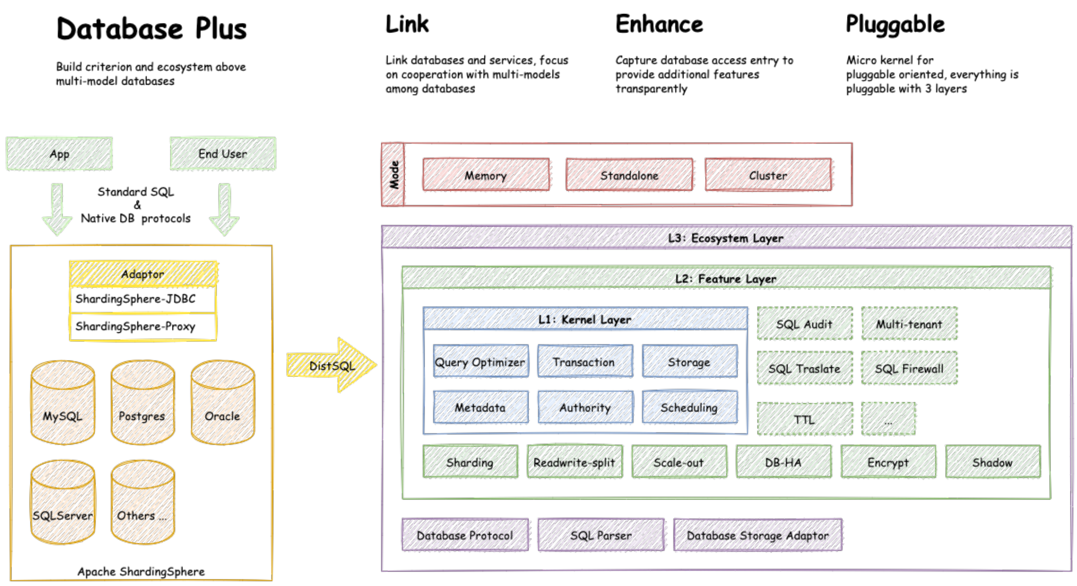

Figure1: Overview

1 Introduction

Apache ShardingSphere including 3 independent products: JDBC, Proxy & Sidecar (Planning). They all provide functions of data scale-out, distributed transaction and distributed governance, applicable in a variety of situations such as Java isomorphism, heterogeneous language and Cloud-Native.

As the cornerstone of enterprises, the relational database has a huge market share. Therefore, we prefer to focus on its incrementation instead of a total overturn.

1.1 ShardingSphere-JDBC

ShardingSphere-JDBC defines itself as a lightweight Java framework that provides extra services at the Java JDBC layer. With the client end connecting directly to the database, it provides services in the form of a jar and requires no extra deployment and dependence. It can be considered as an enhanced 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;
- Supports 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.

1.2 ShardingSphere-Proxy

ShardingSphere-Proxy defines itself as a transparent database proxy, providing a database server that encapsulates database binary protocol to support heterogeneous languages. Currently, MySQL and PostgreSQL (compatible with PostgreSQL-based databases, such as openGauss) versions are provided. It can use any kind of terminal (such as MySQL Command Client, MySQL Workbench, etc.) that is compatible of MySQL or PostgreSQL protocol to operate data, which is friendlier to DBAs.

- Transparent towards applications, it can be used directly as MySQL and PostgreSQL servers;
- Applicable to any kind of terminal that is compatible with MySQL and PostgreSQL protocol.

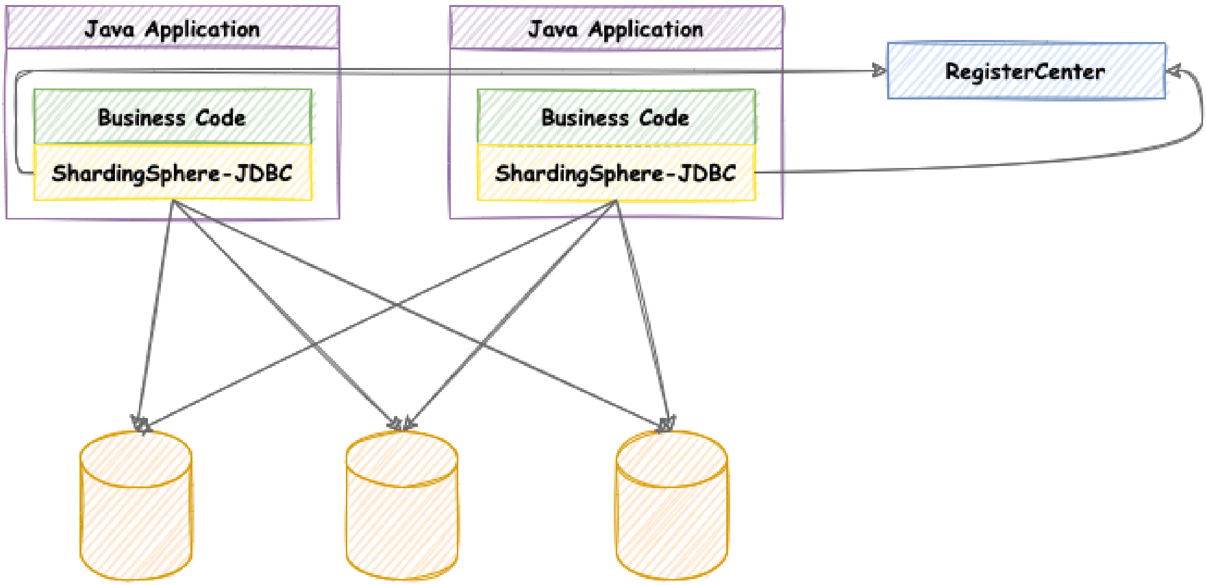

Figure1: ShardingSphere-JDBC Architecture

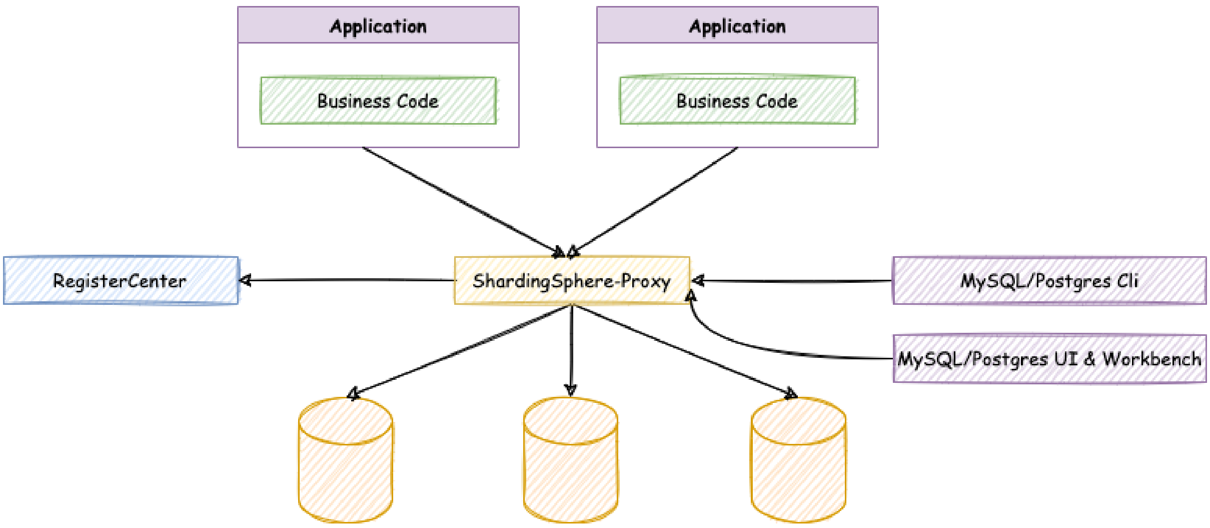

Figure2: ShardingSphere-Proxy Architecture

1.3 ShardingSphere-Sidecar(TODO)

ShardingSphere-Sidecar (TODO) defines itself as a cloud-native database agent of the Kubernetes environment, in charge of all database access in the form of a sidecar. It provides a mesh layer interacting with the database, we call this `Database Mesh`.

Database Mesh emphasizes how to connect distributed data-access applications with the databases. Focusing on interaction, it effectively organizes the interaction between messy applications and databases. The applications and databases that use Database Mesh to visit databases will form a large grid system, where they just need to be put into the right positions accordingly. They are all governed by the mesh layer.

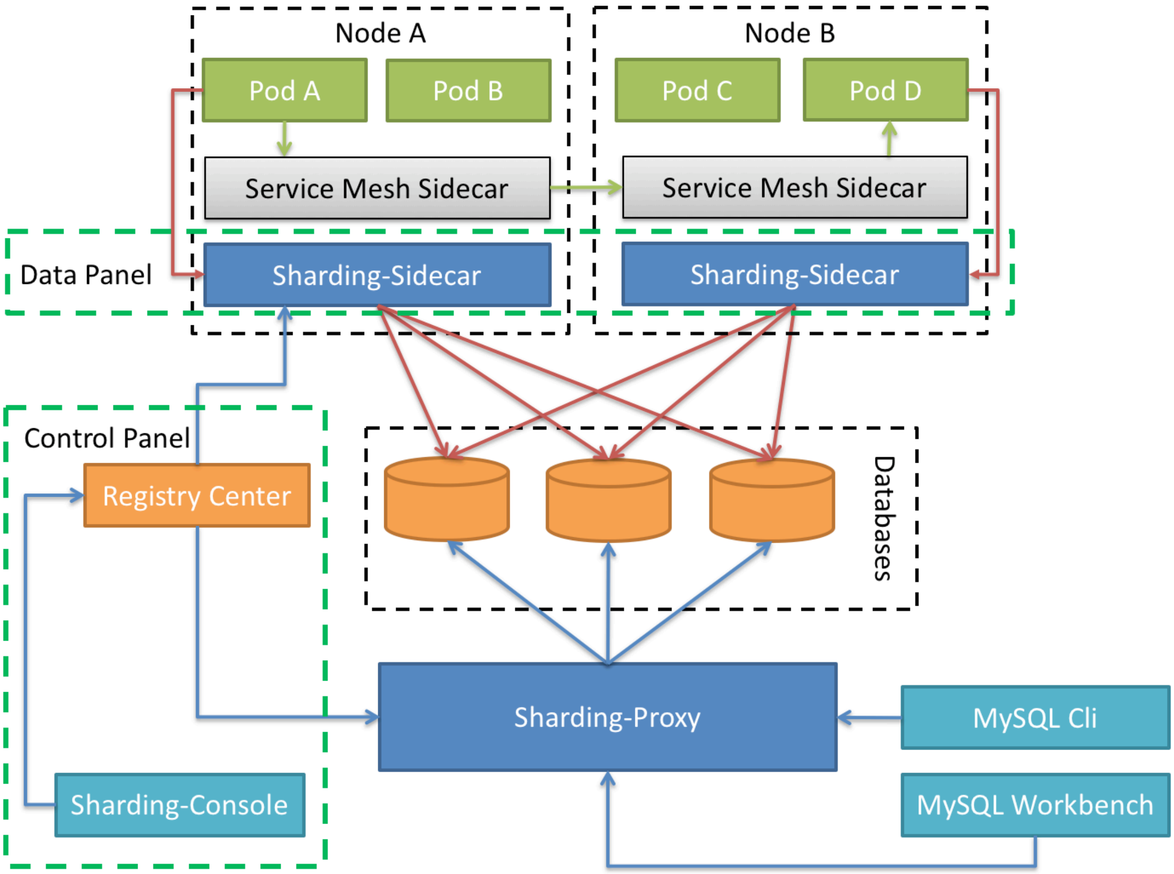

Figure3: ShardingSphere-Sidecar Architecture

	Shardi ngSphere-JDBC	*Shardin gSphere-Proxy*	*ShardingS phere-Sidecar*
Database	Any	My SQL/PostgreSQL	My SQL/PostgreSQL
Connections Count Cost	High	Low	High
Supported Languages	Java Only	Any	Any
Performance	Low loss	Relatively High loss	Low loss
De centralization	Yes	No	No
Static Entry	No	Yes	No

1.4 Hybrid Architecture

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

ShardingSphere is an ecosystem consisting of multiple endpoints together. Through a mixed use of ShardingSphere-JDBC and ShardingSphere-Proxy and a unified sharding strategy by the same registry center, ShardingSphere can build an application system that is applicable to all kinds of scenarios. Architects can adjust the system architecture to the most applicable one to their needs to conduct business more freely.

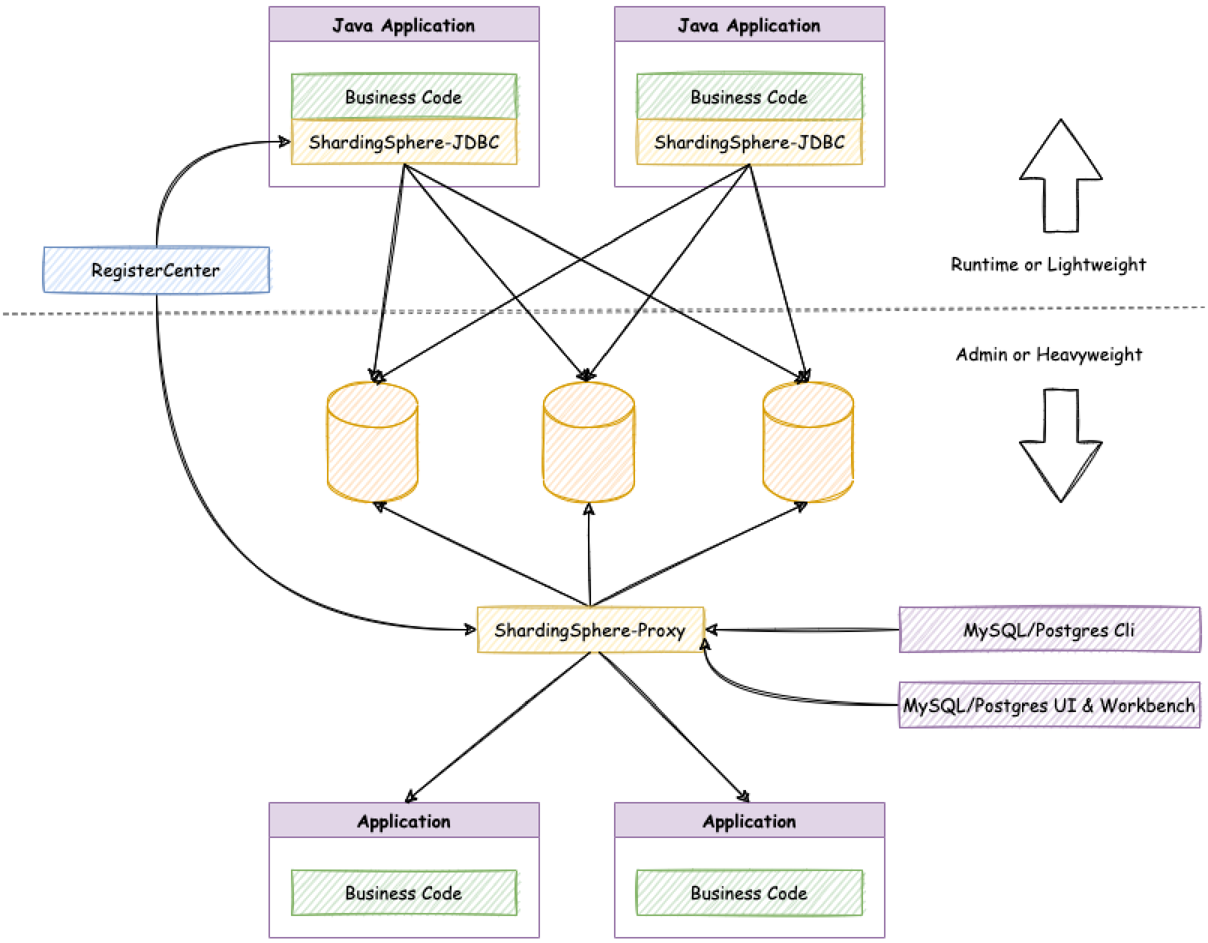

Figure4: ShardingSphere Hybrid Architecture

2 Solution

S olutions/ Features	• Distributed Database*	*Data Security*	*Database Gateway*	*Stress T esting*
	Data Sharding	Data Encrypt	Multi-model Databases Supported	Shadow D atabase
	Readwrit e-splitting	Row Authority (TODO)	SQL Dialect Translate (TODO)	Observ ability
	Distributed Transaction	SQL Audit (TODO)		
	Elastic Scale-out	SQL Firewall (TODO)		
	Highly Available			

3 Roadmap

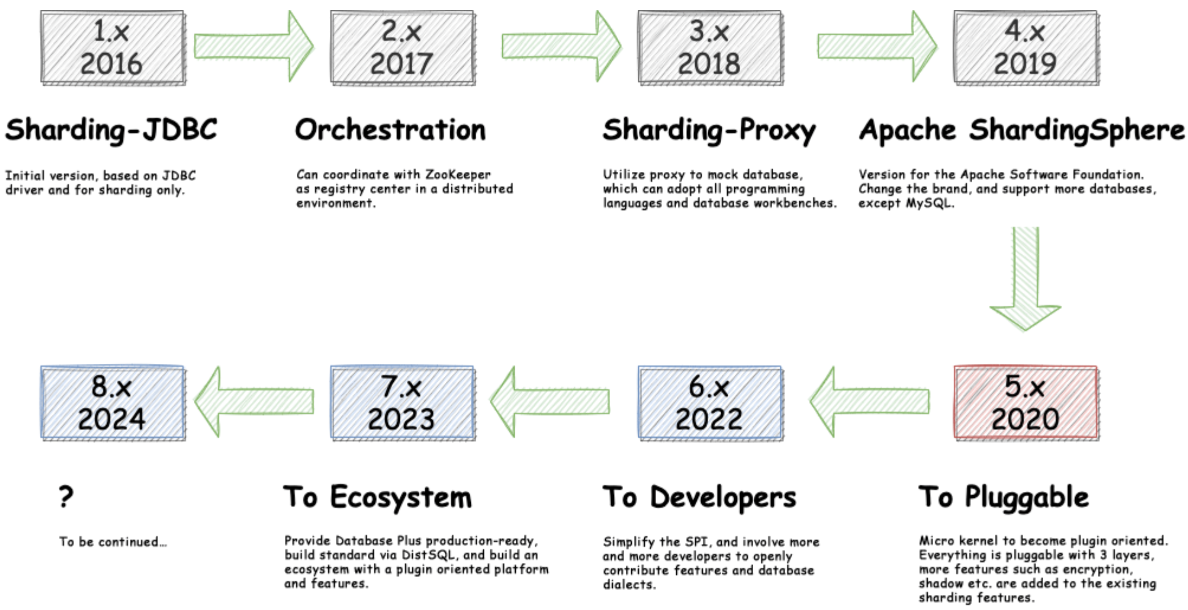

Figure1: Roadmap

4 Quick Start

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

4.1 ShardingSphere-JDBC

4.1.1 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.

4.1.2 Rules Configuration

ShardingSphere-JDBC can be configured by four methods, `Java`, `YAML`, `Spring namespace` and `Spring boot starter`. Developers can choose the suitable method according to different situations. Please refer to User Manual for more details.

4.1.3 3. Create Data Source

Use `ShardingSphereDataSourceFactory` and rule configurations to create `ShardingSphereDataSource`, which implements DataSource interface of JDBC. It can be used for native JDBC or JPA, Hibernate, MyBatis and other ORM frameworks.

```
DataSource dataSource = ShardingSphereDataSourceFactory.
createDataSource(dataSourceMap, configurations, properties);
```

4.2 ShardingSphere-Proxy

4.2.1 Rule Configuration

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

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

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

Please refer to Configuration Manual for more details.

4.2.2 Import Dependencies

If the backend database is PostgreSQL, there' s no need for additional dependencies.

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 `%SHARDINGSPHERE_PROXY_HOME%/lib` directory.

4.2.3 Start Server

- Use default configuration to start

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

Default port is `3307`, default profile directory is `%SHARDINGSPHERE_PROXY_HOME%/conf/` .

- Customize port and profile directory

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

4.2.4 Use ShardingSphere-Proxy

Use MySQL or PostgreSQL client to connect ShardingSphere-Proxy. For example with MySQL:

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

4.3 ShardingSphere-Scaling (Experimental)

4.3.1 Rule Configuration

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

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

Please refer to Build Manual for more details.

4.3.2 Import Dependencies

If the backend database is PostgreSQL, there' s no need for additional dependencies.

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

4.3.3 Start Server

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

4.3.4 Create Migration Job

Use DistSQL interface to manage the migration jobs.

Please refer to Usage Manual for more details.

4.3.5 Related documents

- Features#Scaling : Core Concept, User Norms
- User Manual#Scaling : Build, Manual
- RAL#Scaling : DistSQL for Scaling
- Dev Manual#Scaling : SPI interfaces and implementations

5 Concepts

The functions of Apache ShardingSphere are pretty complex with hundreds of modules, but the concepts are very simple and clear. Most modules are horizontal extensions faced to these concepts.

The concepts include: adaptor faced to independent products, runtime mode faced to startup, DistSQL faced to users and pluggable architecture faced to developers.

This chapter describes concepts about Apache ShardingSphere.

5.1 Adaptor

Apache ShardingSphere including 2 independent products: ShardingSphere-JDBC & ShardingSphere-Proxy. They all provide functions of data scale-out, distributed transaction and distributed governance, applicable in a variety of situations such as Java isomorphism, heterogeneous language and Cloud-Native.

5.1.1 ShardingSphere-JDBC

As the first product and the predecessor of Apache ShardingSphere, ShardingSphere-JDBC defines itself as a lightweight Java framework that provides extra service at Java JDBC layer. With the client end connecting directly to the database, it provides service in the form of jar and requires no extra deployment and dependence. It can be considered as an enhanced 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.

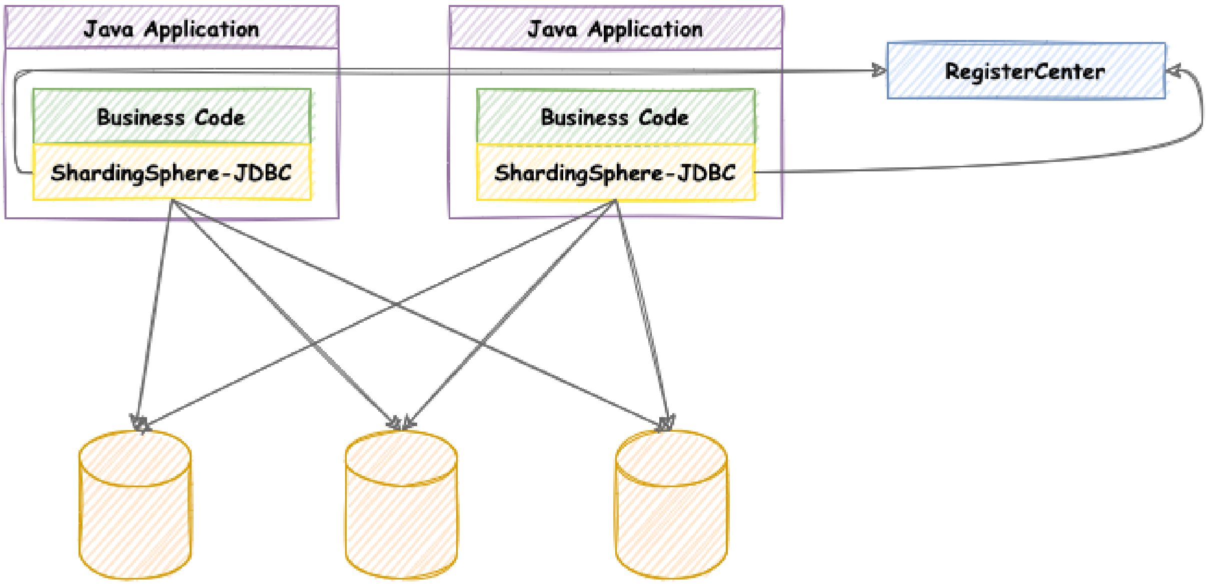

Figure1: ShardingSphere-JDBC Architecture

	ShardingSphere-JDBC	*ShardingSphere-Proxy*
Database	Any	MySQL/PostgreSQL
Connections Count Cost	More	Less
Supported Languages	Java Only	Any
Performance	Low loss	Relatively High loss
Decentralization	Yes	No
Static Entry	No	Yes

ShardingSphere-JDBC is suitable for java application.

5.1.2 ShardingSphere-Proxy

ShardingSphere-Proxy is the second product of Apache ShardingSphere. It defines itself as a transparent database proxy, providing a database server that encapsulates database binary protocol to support heterogeneous languages. Currently, MySQL and PostgreSQL (compatible with PostgreSQL-based databases, such as openGauss) versions are provided. It can use any kind of terminal (such as MySQL Command Client, MySQL Workbench, etc.) that is compatible of MySQL or PostgreSQL protocol to operate data, which is friendlier to DBAs

- Totally transparent to applications, it can be used directly as MySQL/PostgreSQL;
- Applicable to any kind of client end that is compatible with MySQL/PostgreSQL protocol.

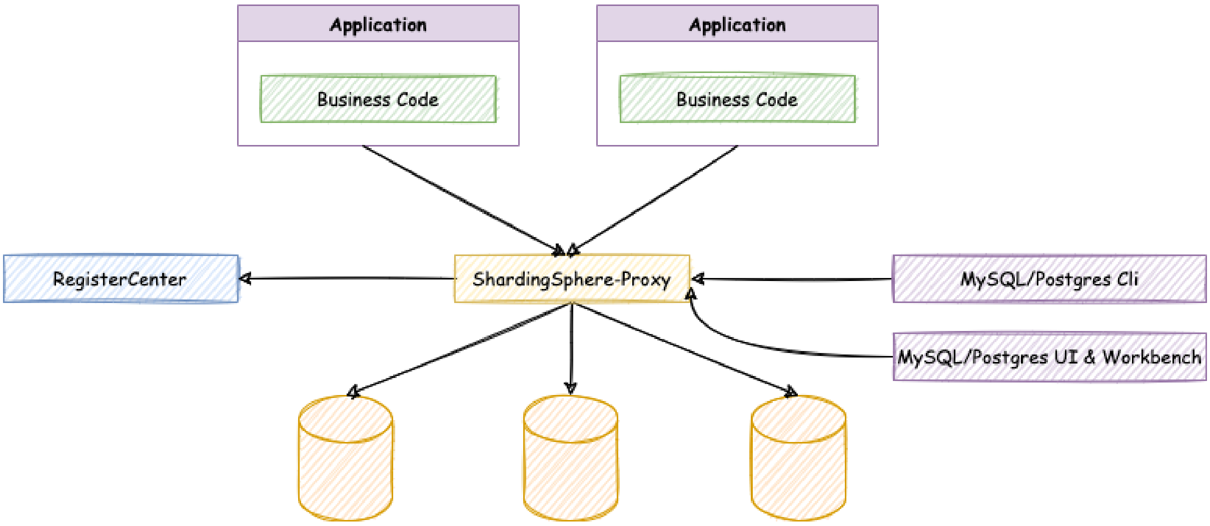

Figure2: ShardingSphere-Proxy Architecture

	ShardingSphere-JDBC	*ShardingSphere-Proxy*
Database	Any	MySQL/PostgreSQL
Connections Count Cost	High	Low
Supported Languages	Java Only	Any
Performance	Low loss	Relatively high loss
Decentralization	Yes	No
Static Entry	No	Yes

The advantages of ShardingSphere-Proxy lie in supporting heterogeneous languages and providing operational entries for DBA.

5.1.3 Hybrid Adaptors

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

ShardingSphere is an ecosystem consisting of multiple endpoints together. Through a mixed use of ShardingSphere-JDBC and ShardingSphere-Proxy and a unified sharding strategy by the same registry center, ShardingSphere can build an application system that is applicable to all kinds of scenarios. Architects can adjust the system architecture to the most applicable one to their needs to conduct business more freely.

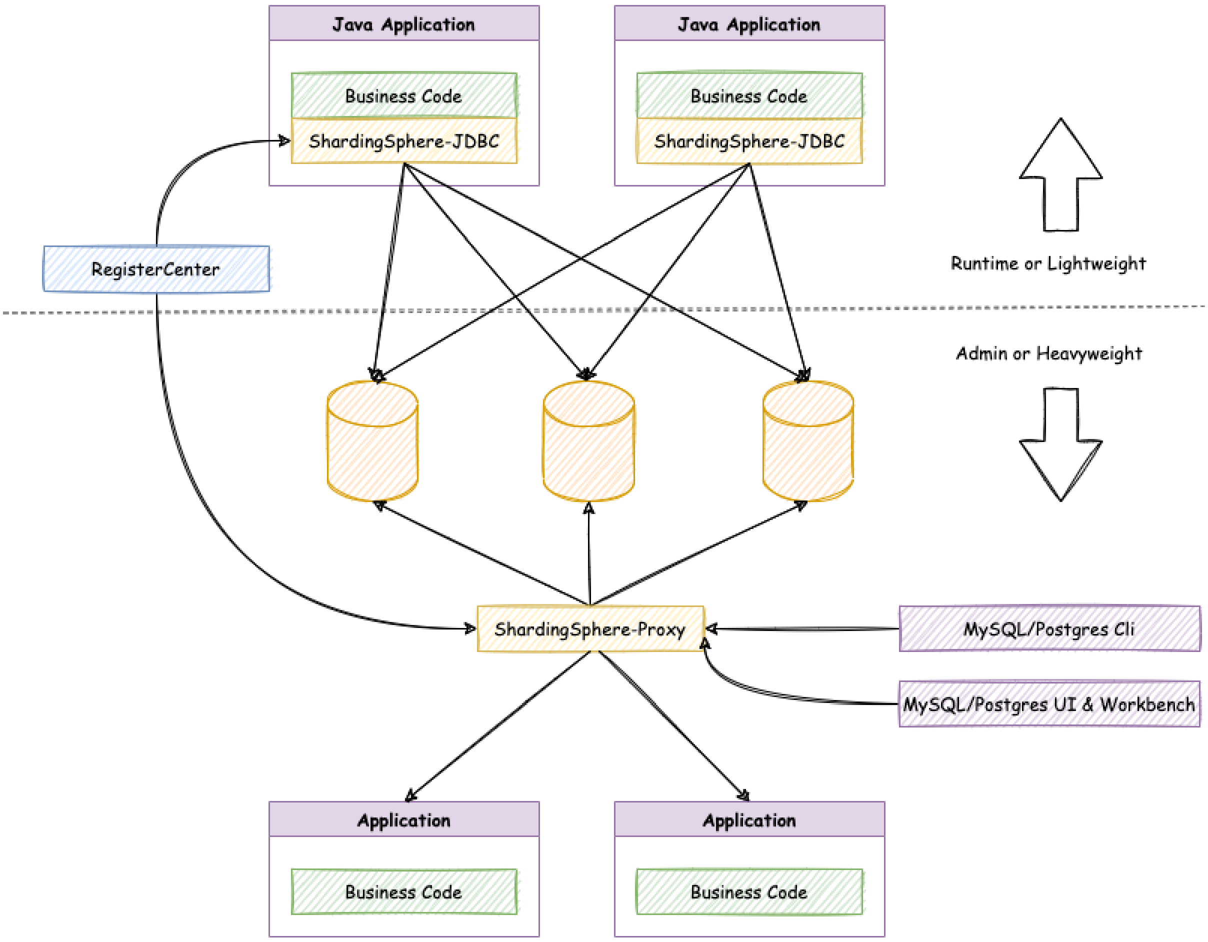

Figure3: Hybrid Architecture

5.2 Mode

5.2.1 Background

In order to meet the different needs of users for quick test startup, stand-alone running and cluster running, Apache shardingsphere provides various mode such as memory, stand-alone and cluster.

5.2.2 Memory mode

Suitable for fast integration testing, which is convenient for testing, such as for developers looking to perform fast integration function testing. This is the default mode of Apache ShardingSphere.

5.2.3 Standalone mode

Suitable in a standalone environment, through which data sources, rules, and metadata can be persisted. Will create a `.shardingsphere` file in the root directory to store configuration data by default.

5.2.4 Cluster mode

Suitable for use in distributed scenarios which provides metadata sharing and state coordination among multiple computing nodes. It is necessary to provide registry center for distributed coordination, such as ZooKeeper or Etcd.

5.3 DistSQL

5.3.1 Background

DistSQL（Distributed SQL）is Apache ShardingSphere specific SQL, which provide added-on operation capability beside standard SQL.

5.3.2 Challenges

When using ShardingSphere-Proxy, developers can operate data just like using database, but they need to configure resources and rules through YAML file (or registry center). However, the format of YAML and habits changed by using registry center are not friendly to DBA.

DistSQL enables users to operate Apache ShardingSphere like a database, transforming it from a framework and middleware for developers to a database product for DBAs.

DistSQL is divided into RDL, RQL and RAL.

- RDL (Resource & Rule Definition Language) responsible for the definition of resources and rules;
- RQL (Resource & Rule Query Language) responsible for the query of resources and rules;

- RAL (Resource & Rule Administration Language) responsible for the added-on administrator feature of hint, transaction type switch, sharding execute planning and so on.

5.3.3 Goal

It is the design goal of DistSQL to break the boundary between middleware and database and let developers use Apache ShardingSphere just like database.

5.3.4 Notice

DistSQL can use for ShardingSphere-Proxy only, not for ShardingSphere-JDBC now.

5.4 Pluggable Architecture

5.4.1 Background

In Apache ShardingSphere, many functionality implementations are uploaded through SPI (Service Provider Interface), which is a kind of API for the third party to implement or expand, and can be applied in framework expansion or component replacement.

5.4.2 Challenges

Pluggable architecture is very difficult to design for the project architecture. It needs to make each module decouple to independent and imperceptible to each other totally, and enables appendable functions in a way of superposition through a pluggable kernel. Design an architecture to completely isolate each function, not only can stimulate the enthusiasm of the open source community, but also can guarantee the quality of the project.

Apache ShardingSphere begin to focus on pluggable architecture from version 5.x, features can be embedded into project flexibility. Currently, the features such as data sharding, readwrite-splitting, data encrypt, shadow database, and SQL dialects / database protocols such as MySQL, PostgreSQL, SQLServer, Oracle supported are all weaved by plugins. Developers can customize their own ShardingSphere just like building lego blocks. There are lots of SPI extensions for Apache ShardingSphere now and increase continuously.

5.4.3 Goal

It is the design goal of Apache shardingsphere pluggable architecture to enable developers to customize their own unique systems just like building blocks.

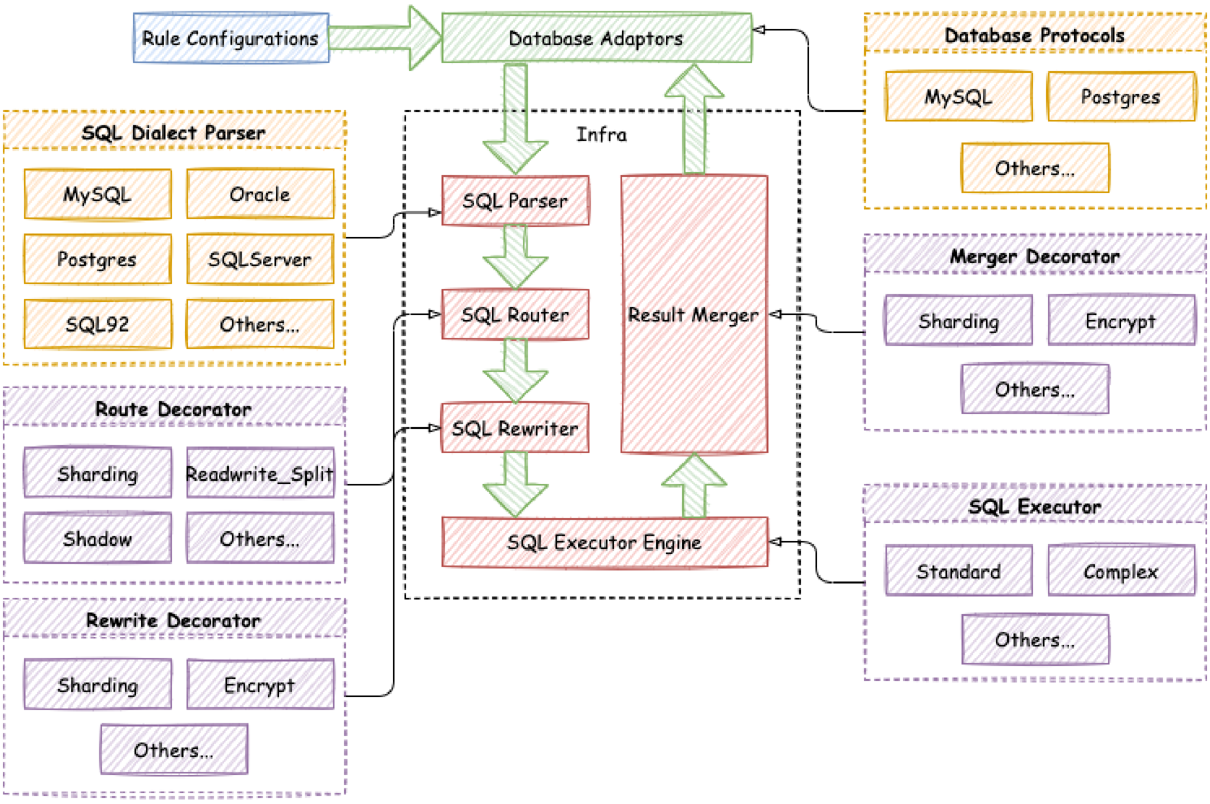

Figure4: Pluggable Platform

5.4.4 Implementation

The pluggable architecture of Apache ShardingSphere are composed by L1 Kernel Layer, L2 Feature Layer and L3 Ecosystem Layer.

L1 Kernel Layer

An abstraction of basic capabilities of database. All components are required and the specific implementation can be replaced by pluggable way. It includes query optimizer, distributed transaction engine, distributed execution engine, authority engine and scheduling engine.

L2 Feature Layer

Used to provide enhanced capability. All components are optional and can contain zero or multiple components. Components isolate each other and multiple components can be used together superimposed. It includes data sharding, readwrite-splitting, database highly availability, data encryption, 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.

L3 Ecosystem Layer

Used to integrate into the current database ecosystem. It includes database protocol, SQL parser and storage adapter.

6 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.

6.1 DB Compatibility

6.1.1 Background

With information technology innovating, more and more applications established in the new fields, prompt and push evolution of human society' s cooperation mode. Data is increasing explosively, the data storage and computing method are facing innovation all the time.

Transaction, big data, association analysis, Internet of things and other scenarios subdivided quickly, a single database can not apply to all application scenarios anymore. At the same time, the internal of scenario is becoming more and more detailed, and it has become normal for similar scenarios to use different databases.

The trend of database fragmentation is coming.

6.1.2 Challenges

There is no unified database access protocol and SQL dialect, as well as the maintenance and monitoring methods differences by various databases, learning and maintenance cost of developers and DBAs are increasing rapidly. Improving the compatibility with the original database is the premise of providing incremental services on it.

The compatibility between SQL dialect and database protocol is the key point to improve database compatibility.

6.1.3 Goal

The goal of database compatibility for Apache ShardingSphere is make user feel nothing changed among various original databases.

6.1.4 SQL Parser

SQL is the standard operation language between users and databases. SQL Parse engine used to parse SQL into an abstract syntax tree to provide Apache ShardingSphere understand and implement the add-on features.

It supports SQL dialect for MySQL, PostgreSQL, SQLServer, Oracle, openGauss and SQL that conform to the SQL92 specification. However, due to the complexity of SQL syntax, there are still a little of SQL do not support yet.

This chapter has listed unsupported SQLs reference for users.

There are some unsupported SQLs maybe missing, welcome to finish them. We will try best to support the unavailable SQLs in future versions.

MySQL

The unsupported SQL list for MySQL are as follows:

SQL
FLUSH PRIVILEGES
CLONE LOCAL DATA DIRECTORY = 'clone_dir'
INSTALL COMPONENT 'file://component1', 'file://component2'
UNINSTALL COMPONENT 'file://component1', 'file://component2'
SHOW CREATE USER user
REPAIR TABLE t_order
OPTIMIZE TABLE t_order
CHECKSUM TABLE t_order
CHECK TABLE t_order
SET RESOURCE GROUP group_name
DROP RESOURCE GROUP group_name
CREATE RESOURCE GROUP group_name TYPE = SYSTEM
ALTER RESOURCE GROUP rg1 VCPU = 0-63

PostgreSQL

The unsupported SQL list for PostgreSQL are as follows:

TODO

SQLServer

The unsupported SQL list for SQLServer are as follows:

TODO

Oracle

The unsupported SQL list for Oracle are as follows:

TODO

SQL92

The unsupported SQL list for SQL92 are as follows:

TODO

6.1.5 DB Protocol

Apache ShardingSphere implements MySQL and PostgreSQL Protocol.

6.2 Management

6.2.1 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.

6.2.2 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.

6.2.3 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.

6.2.4 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.

6.3 Sharding

6.3.1 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.

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.

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.

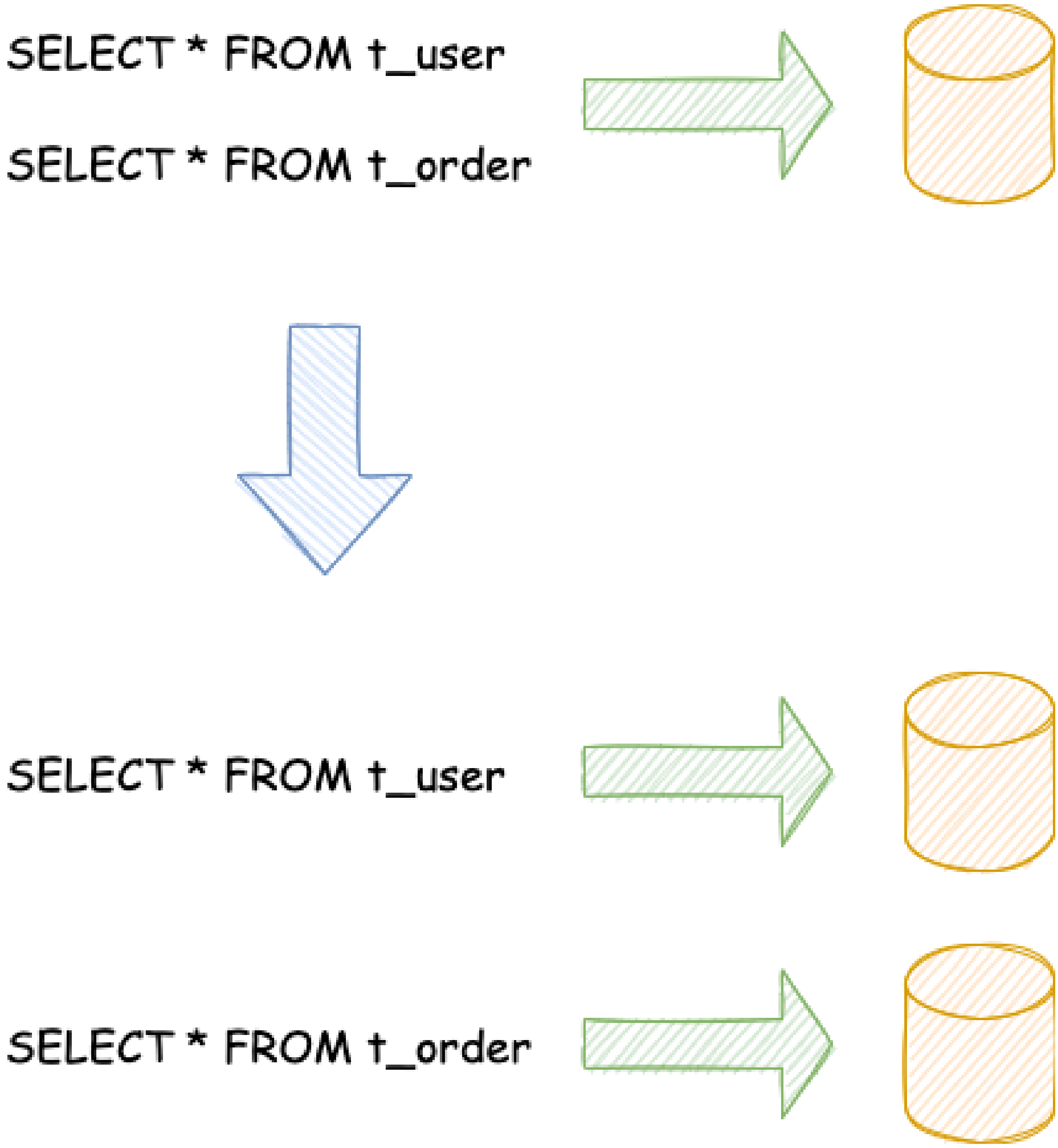

Figure1: Vertical Sharding

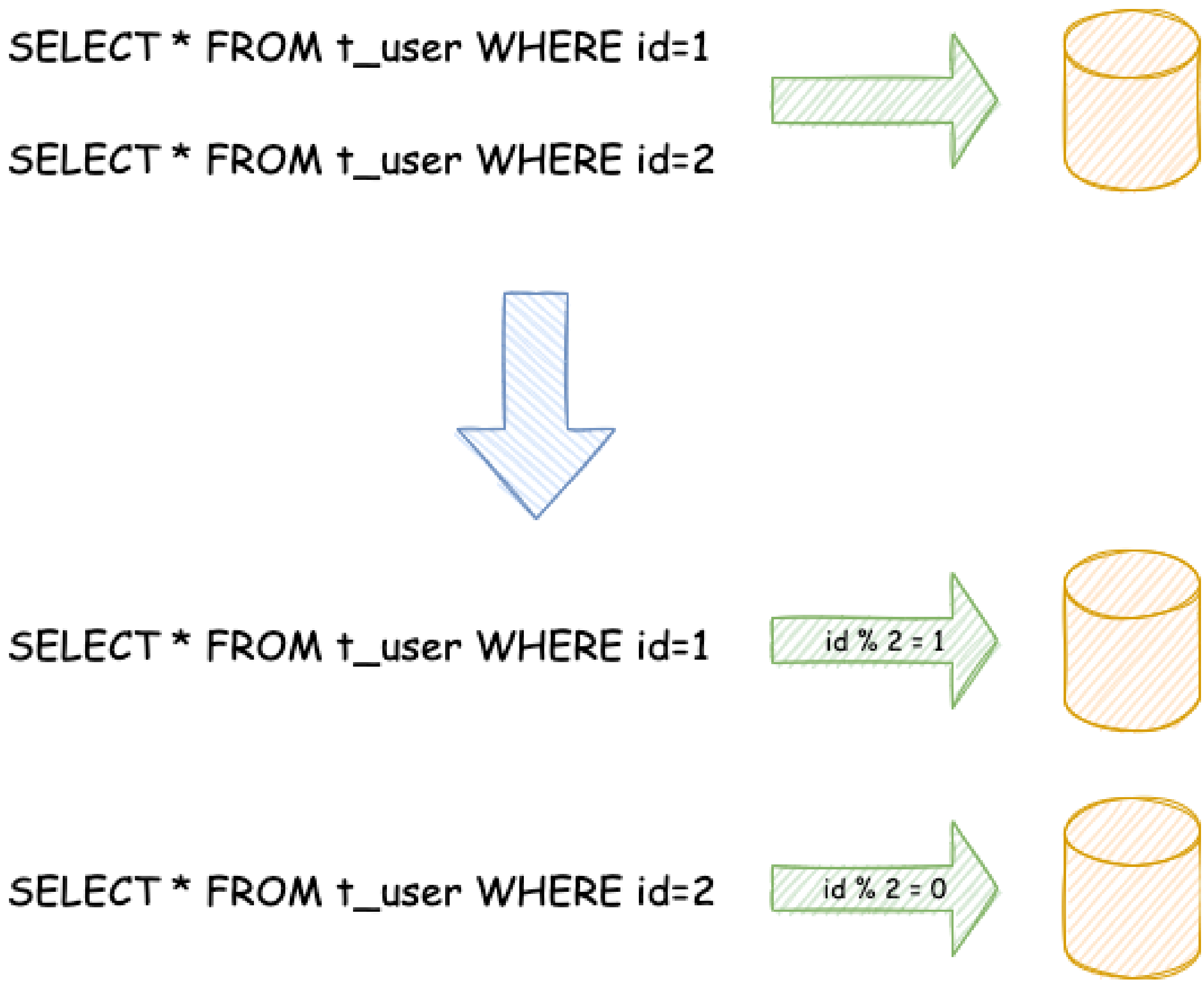

Figure2: Horizontal Sharding

6.3.2 Challenges

Though sharding has solved problems such as performance, availability and single-node backup and recovery, its distributed architecture has also introduced some new problems as acquiring profits.

One problem is that application development engineers and database administrators' operations become exceptionally laborious, when facing such scattered databases and tables. They should know exactly which database table is the one to acquire data from.

Another challenge is that, the SQL that runs rightly in single-node databases may not be right in the sharding database. The change of table name after sharding, or misconducts caused by operations such as pagination, order by or aggregated group by are just the case in point.

Cross-database transaction is also a tricky thing that distributed databases need to deal. Fair use of sharding tables can also lead to the full use of local transactions when single-table data amount decreases. Troubles brought by distributed transactions can be avoided by the wise use of different tables in the same database. When cross-database transactions cannot be avoided, some businesses still need to keep transactions consistent. Internet giants have not massively adopted XA based distributed transactions since they are not able to ensure its performance in high-concurrency situations. They usually replace strongly consistent transactions with eventually consistent soft state.

6.3.3 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.

6.3.4 Core Concept

Overview

This chapter is to introduce core concepts of data sharding.

Table

Table is the core concept of data sharding transparently. There are diversified tables provided for different data sharding requirements by Apache ShardingSphere.

Logic Table

The logical name of the horizontal sharding databases (tables) with the same schema, it is the logical table identification in SQL. For instance, the data of order is divided into 10 tables according to the last number of the primary key, and they are from `t_order_0` to `t_order_9`, whose logic name is `t_order`.

Actual Table

The physical table that really exists in the horizontal sharding database, i.e., t_order_0 to t_order_9 in the instance above.

Binding Table

It refers to the primary table and the joiner table with the same sharding rules. for example, t_order and t_order_item are both sharded by order_id, so they are binding tables with each other. Cartesian product correlation will not appear in the multi-tables correlating query, so the query efficiency will increase greatly. Take this one for example, if SQL is:

```
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);
```

When binding table relations are not configured, suppose the sharding key order_id routes value 10 to sharding 0 and value 11 to sharding 1, there will be 4 SQLs in Cartesian product after routing:

```
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);
```

With binding table configuration, there should be 2 SQLs after routing:

```
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);
```

In them, table t_order in the left end of FROM will be taken by ShardingSphere as the primary table of query. In a similar way, ShardingSphere will also take table t_order in the left end of FROM as the primary table of the whole binding table. All the route computations will only use the sharding strategy of the primary table, so sharding computation of t_order_item table will use the conditions of t_order. Due to this, sharding keys in binding tables should be totally identical.

Broadcast Table

It refers to tables that exist in all sharding database sources. The schema and data must consist in each database. It can be applied to the small data volume that needs to correlate with big data tables to query, dictionary table for example.

Single Table

It refers to only one table that exists in all sharding database sources. It is suitable for little data in table without sharding.

Data Node

As the atomic unit of sharding, it consists of data source name and actual table name, e.g. `ds_0.t_order_0`.

Mapping relationship between logic tables and actual tables and can be divided into two kinds: uniform topology and user-defined topology.

Uniform topology

It means that tables are evenly distributed in each data source, for example:

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

The data node configurations will be as follows:

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

User-defined topology

It means that tables are distributed with certain rules, for example:

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

The data node configurations will be as follows:

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

Sharding

Sharding Key

Column used to determine database (table) sharding. For example, in last number modulo of order ID sharding, order ID is taken as the sharding key. The full route executed when there is no sharding column in SQL has a poor performance. Besides single sharding column, Apache ShardingSphere also supports multiple sharding columns.

Sharding Algorithm

Data sharding can be achieved by sharding algorithms through `=`, `>=`, `<=`, `>`, `<`, `BETWEEN` and `IN`. It can be implemented by developers themselves, or using built-in syntactic sugar of Apache ShardingSphere, with high flexibility.

Auto Sharding Algorithm

It provides syntactic sugar for sharding algorithm. It used to manage all data nodes automatically, user do not care about the topology of physical data nodes. It includes lots of implementation for Mod, Hash, Range and Time Interval etc.

User-Defined Sharding Algorithm

It provides interfaces for developers to implement the sharding algorithm related to business implementation, and allows users to manage the physical topology physical data nodes by themselves. It includes:

- Standard Sharding Algorithm

It is to process the sharding case in which single sharding keys `=`, `IN`, `BETWEEN AND`, `>`, `<`, `>=`, `<=` are used.

- Complex Keys Sharding Algorithm

It is to process the sharding case in which multiple sharding keys are used. It has a relatively complex logic that requires developers to deal by themselves.

- Hint Sharding Algorithm

It is to process the sharding case in which Hint is used.

Sharding Strategy

It includes the sharding key and the sharding algorithm, and the latter one is extracted out for its independence. Only sharding key + sharding algorithm can be used in sharding operation.

SQL Hint

In the case that the sharding column is not decide by SQL but other external conditions, SQL hint can be used to inject sharding value. For example, databases are shard according to the staff' s ID, but column does not exist in the database. SQL Hint can be used by two ways, Java API and SQL comment (TODO). Please refer to Hint for more details.

Inline Expression

Motivation

Configuration simplicity and unity are two main problems that inline expression intends to solve.

In complex sharding rules, with more data nodes, a large number of configuration repetitions make configurations difficult to maintain. Inline expressions can simplify data node configuration work.

Java codes are not helpful in the unified management of common configurations. Writing sharding algorithms with inline expressions, users can store rules together, making them easier to be browsed and stored.

Syntax Explanation

The use of inline expressions is really direct. Users only need to configure `${ expression }` or `$->{ expression }` to identify them. ShardingSphere currently supports the configurations of data nodes and sharding algorithms. Inline expressions use Groovy syntax, which can support all kinds of operations, including inline expressions. For example:

`${begin..end}` means range

`${[unit1, unit2, unit_x]}` means enumeration

If there are many continuous `${ expression }` or `$->{ expression }` expressions, according to each sub-expression result, the ultimate result of the whole expression will be in cartesian combination.

For example, the following inline expression:

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

Will be parsed as:

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

Configuration

Data Node

For evenly distributed data nodes, if the data structure is as follow:

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

It can be simplified by inline expression as:

```
db${0..1}.t_order${0..1}
```

Or

```
db$->{0..1}.t_order$->{0..1}
```

For self-defined data nodes, if the data structure is:

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

It can be simplified by inline expression as:

```
db0.t_order${0..1},db1.t_order${2..4}
```

Or

```
db0.t_order$->{0..1},db1.t_order$->{2..4}
```

For data nodes with prefixes, inline expression can also be used to configure them flexibly, if the data structure is:

```
db0
  ├── t_order_00
  ├── t_order_01
  ├── t_order_02
  ├── t_order_03
  ├── t_order_04
  ├── t_order_05
  ├── t_order_06
```

```
  ├── t_order_07
  ├── t_order_08
  ├── t_order_09
  ├── t_order_10
  ├── t_order_11
  ├── t_order_12
  ├── t_order_13
  ├── t_order_14
  ├── t_order_15
  ├── t_order_16
  ├── t_order_17
  ├── t_order_18
  ├── t_order_19
  └── t_order_20
db1
  ├── t_order_00
  ├── t_order_01
  ├── t_order_02
  ├── t_order_03
  ├── t_order_04
  ├── t_order_05
  ├── t_order_06
  ├── t_order_07
  ├── t_order_08
  ├── t_order_09
  ├── t_order_10
  ├── t_order_11
  ├── t_order_12
  ├── t_order_13
  ├── t_order_14
  ├── t_order_15
  ├── t_order_16
  ├── t_order_17
  ├── t_order_18
  ├── t_order_19
  └── t_order_20
```

Users can configure separately, data nodes with prefixes first, those without prefixes later, and automatically combine them with the cartesian product feature of inline expressions. The example above can be simplified by inline expression as:

```
db${0..1}.t_order_0${0..9}, db${0..1}.t_order_${10..20}
```

Or

```
db$->{0..1}.t_order_0$->{0..9}, db$->{0..1}.t_order_$->{10..20}
```

Sharding Algorithm

For single sharding SQL that uses = and `IN`, inline expression can replace codes in configuration.

Inline expression is a piece of Groovy code in essence, which can return the corresponding real data source or table name according to the computation method of sharding keys.

For example, sharding keys with the last number 0 are routed to the data source with the suffix of 0, those with the last number 1 are routed to the data source with the suffix of 1, the rest goes on in the same way. The inline expression used to indicate sharding algorithm is:

```
ds${id % 10}
```

Or

```
ds$->{id % 10}
```

Distributed Primary Key

Motivation

In the development of traditional database software, the automatic sequence generation technology is a basic requirement. All kinds of databases have provided corresponding support for this requirement, such as MySQL auto-increment key, Oracle auto-increment sequence and so on. It is a tricky problem that there is only one sequence generated by different data nodes after sharding. Auto-increment keys in different physical tables in the same logic table can not perceive each other and thereby generate repeated sequences. It is possible to avoid clashes by restricting the initiative value and increasing the step of auto-increment key. But introducing extra operation rules can make the solution lack integrity and scalability.

Currently, there are many third-party solutions that can solve this problem perfectly, (such as UUID and others) relying on some particular algorithms to generate unrepeated keys or introducing sequence generation services. We have provided several built-in key generators, such as UUID, SNOWFLAKE. Besides, we have also extracted a key generator interface to make users implement self-defined key generator.

Built-In Key Generator

UUID

Use `UUID.randomUUID()` to generate the distributed key.

SNOWFLAKE

Users can configure the strategy of each table in sharding rule configuration module, with default snowflake algorithm generating 64bit long integral data.

As the distributed sequence generation algorithm published by Twitter, snowflake algorithm can ensure sequences of different processes do not repeat and those of the same process are ordered.

Principle

In the same process, it makes sure that IDs do not repeat through time, or through order if the time is identical. In the same time,with monotonously increasing time, if servers are generally synchronized, generated sequences are generally assumed to be ordered in a distributed environment. This can guarantee the effectiveness in index field insertion, like the sequence of MySQL Innodb storage engine.

In the sequence generated with snowflake algorithm, binary form has 4 parts, 1 bit sign, 41bit timestamp, 10bit work ID and 12bit sequence number from high to low.

- sign bit (1bit)

Reserved sign bit, constantly to be zero.

- timestamp bit (41bit)

41bit timestamp can contain 2 to the power of 41 milliseconds. One year can use `365 * 24 * 60 * 60 * 1000` milliseconds. We can see from the calculation:

```
Math.pow(2, 41) / (365 * 24 * 60 * 60 * 1000L);
```

The result is approximately equal to 69.73 years. Apache ShardingSphere snowflake algorithm starts from November 1st, 2016, and can be used until 2086, which we believe can satisfy the requirement of most systems.

- work ID bit (10bit)

The sign is the only one in Java process. If applied in distributed deployment, each work ID should be different. The default value is 0 and can be set through properties.

- sequence number bit (12bit)

The sequence number is used to generate different IDs in a millisecond. If the number generated in that millisecond exceeds 4,096 (2 to the power of 12), the generator will wait till the next millisecond to continue.

Please refer to the following picture for the detailed structure of snowflake algorithm sequence.

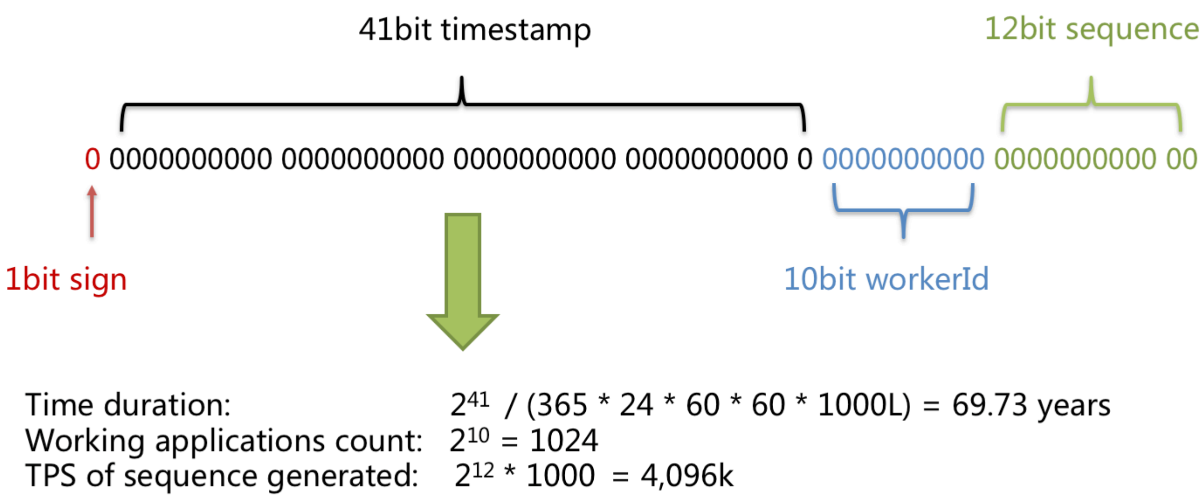

Figure3: Snowflake

Clock-Back

The clock-back of server can generate repeated sequence, so the default distributed sequence generator has provided a maximum clock-back millisecond. If the clock-back time has exceeded it, the program will report error. If it is within the tolerance range, the generator will wait till after the last generation time and then continue to work. The default maximum clock-back millisecond is 0 and can be set through properties.

Hint Sharding Route

Motivation

Apache ShardingSphere can be compatible with SQL in way of parsing SQL statements and extracting columns and values to shard. If SQL does not have sharding conditions, it is impossible to shard without full data node route.

In some applications, sharding conditions are not in SQL but in external business logic. So it requires to designate sharding result externally, which is referred to as `Hint` in ShardingSphere.

Mechanism

Apache ShardingSphere uses `ThreadLocal` to manage sharding key values. Users can program to add sharding conditions to `HintManager`, but the condition is only effective within the current thread.

In addition to the programming method, Apache ShardingSphere also plans to cite Hint through special notation in SQL, so that users can use that function in a more transparent way.

The SQL designated with sharding hint will ignore the former sharding logic but directly route to the designated node.

6.3.5 Use Norms

Background

Though Apache ShardingSphere intends to be compatible with all the SQLs and stand-alone databases, the distributed scenario has brought more complex situations to the database. Apache ShardingSphere wants to solve massive data OLTP problem first and complete relevant OLAP support problem little by little.

SQL

SQL Supporting Status

Compatible with all regular SQL when **routing to single data node**; **The SQL routing to multiple data nodes** is pretty complex, it divides the scenarios as totally supported, experimental supported and unsupported.

Totally Supported

Fully support DML, DDL, DCL, TCL and most regular DAL. Support complex query with pagination, DISTINCT, ORDER BY, GROUP BY, aggregation and table JOIN.

Regular Query

- SELECT Clause

```
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)]
```

Subquery

Stable supported when sharding keys are using in both subquery and outer query, and values of sharding keys are the same.

For example:

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

Stable supported for subquery with pagination.

For example:

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

Sharding value in expression

Sharding value in calculated expressions will lead to full routing.

For example, if `create_time` is sharding value:

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

Experimental Supported

Experimental support specifically refers to use of `Federation execution engine`. The engine still in rapid development, basically available to users, but it still needs lots of optimization. It is an experimental product.

Subquery

Experimental supported when sharding keys are not using for both subquery and outer query, or values of sharding keys are not the same.

For example:

```
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;
```

Join with cross databases

When tables in a join query are distributed on different database instances, sql statement will be supported by `Federation execution engine`. Assuming that `t_order` and `t_order_item` are sharding tables with multiple data nodes, and no binding table rules are configured, `t_user` and `t_user_role` are single tables that distributed on different database instances. `Federation execution engine` can support the following commonly used join query:

```
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;
```

Unsupported

CASE WHEN can not support as following:

- `CASE WHEN` containing sub-query
- `CASE WHEN` containing logical-table (instead of table alias)

UNION and UNION ALL can not support as following:

- containing sharding or broadcast table

SQL Example

Stable supported SQL	Necessary conditions
SELECT * FROM tbl_name	
SELECT * FROM tbl_name WHERE (col1 = ? or col2 = ?) and col3 = ?	
SELECT * FROM tbl_name WHERE col1 = ? ORDER BY col2 DESC LIMIT ?	
SELECT COUNT(*), SUM(col1), MIN(col1), MAX(col1), AVG(col1) FROM tbl_name WHERE col1 = ?	
SELECT COUNT(col1) FROM tbl_name WHERE col2 = ? GROUP BY col1 ORDER BY col3 DESC LIMIT ?, ?	
SELECT DISTINCT * FROM tbl_name WHERE col1 = ?	
SELECT COUNT(DISTINCT col1), SUM(DISTINCT col1) FROM tbl_name	
(SELECT * FROM tbl_name)	
SELECT * FROM (SELECT * FROM tbl_name WHERE col1 = ?) o WHERE o.col1 = ?	Subquery and outer query in same sharded data node after route
INSERT INTO tbl_name (col1, col2,…) VALUES (?, ?, ….)	
INSERT INTO tbl_name VALUES (?, ?,….)	
INSERT INTO tbl_name (col1, col2, …) VALUES(1 + 2, ?, …)	
INSERT INTO tbl_name (col1, col2, …) VALUES (?, ?, ….), (?, ?, ….)	
INSERT INTO tbl_name (col1, col2, …) SELECT col1, col2, …FROM tbl_name WHERE col3 = ?	Inserted and selected table must be the same or binding tables
REPLACE INTO tbl_name (col1, col2, …) SELECT col1, col2, …FROM tbl_name WHERE col3 = ?	Replaced and selected table must be the same or binding tables
UPDATE tbl_name SET col1 = ? WHERE col2 = ?	
DELETE FROM tbl_name WHERE col1 = ?	
CREATE TABLE tbl_name (col1 int, …)	
ALTER TABLE tbl_name ADD col1 varchar(10)	
DROP TABLE tbl_name	
TRUNCATE TABLE tbl_name	
CREATE INDEX idx_name ON tbl_name	
DROP INDEX idx_name ON tbl_name	
DROP INDEX idx_name	

Experimental supported SQL	Necessary conditions
SELECT * FROM (SELECT * FROM tbl_name) o	
SELECT * FROM (SELECT * FROM tbl_name) o WHERE o.col1 = ?	
SELECT * FROM (SELECT * FROM tbl_name WHERE col1 = ?) o	
SELECT * FROM (SELECT * FROM tbl_name WHERE col1 = ?) o WHERE o.col1 = ?	Subquery and outer query in different sharded data node after route
SELECT (SELECT MAX(col1) FROM tbl_name) a, col2 from tbl_name	
SELECT SUM(DISTINCT col1), SUM(col1) FROM tbl_name	
SELECT col1, SUM(col2) FROM tbl_name GROUP BY col1 HAVING SUM(col2) > ?	

Slow SQL	Reason
SELECT * FROM tbl_name WHERE to_date(create_time, 'yyyy-mm-dd') = ?	Full route because of sharding value in calculate expression

---

Unsupported SQL	Reason	So lution
INSERT INTO tbl_name (col1, col2, …) SELECT * FROM tbl_name WHERE col3 = ?	SELECT clause does not support *-shorthand and built-in key generator	•
REPLACE INTO tbl_name (col1, col2, …) SELECT * FROM tbl_name WHERE col3 = ?	SELECT clause does not support *-shorthand and built-in key generator	•
SELECT MAX(tbl_name.col1) FROM tbl_name	Use table name as column owner in function	I nstead of table alias

Pagination

Totally support pagination queries of MySQL, PostgreSQL and Oracle; partly support SQLServer pagination query due to its complexity.

Pagination Performance

Performance Bottleneck

Pagination with query offset too high can lead to a low data accessibility, take MySQL as an example:

```
SELECT * FROM t_order ORDER BY id LIMIT 1000000, 10
```

This SQL will make MySQL acquire another 10 records after skipping 1,000,000 records when it is not able to use indexes. Its performance can thus be deduced. In sharding databases and sharding tables (suppose there are two databases), to ensure the data correctness, the SQL will be rewritten as this:

```
SELECT * FROM t_order ORDER BY id LIMIT 0, 1000010
```

It also means taking out all the records prior to the offset and only acquire the last 10 records after ordering. It will further aggravate the performance bottleneck effect when the database is already slow in execution. The reason for that is the former SQL only needs to transmit 10 records to the user end, but now it will transmit `1,000,010 * 2` records after the rewrite.

Optimization of ShardingSphere

ShardingSphere has optimized in two ways.

Firstly, it adopts stream process + merger ordering to avoid excessive memory occupation. SQL rewrite unavoidably occupies extra bandwidth, but it will not lead to sharp increase of memory occupation. Most people may assume that ShardingSphere would upload all the `1,000,010 * 2` records to the memory and occupy a large amount of it, which can lead to memory overflow. But each ShardingSphere comparison only acquires current result set record of each shard, since result set records have their own order. The record stored in the memory is only the current position pointed by the cursor in the result set of the shard routed to. For the item to be sorted which has its own order, merger ordering only has the time complexity of `O(nlogn)`, with a very low performance consumption.

Secondly, ShardingSphere further optimizes the query that only falls into single shards. Requests of this kind can guarantee the correctness of records without rewriting SQLs. Under this kind of situation, ShardingSphere will not do that in order to save the bandwidth.

Pagination Solution Optimization

For LIMIT cannot search for data through indexes, if the ID continuity can be guaranteed, pagination by ID is a better solution:

```
SELECT * FROM t_order WHERE id > 100000 AND id <= 100010 ORDER BY id
```

Or use the ID of last record of the former query result to query the next page:

```
SELECT * FROM t_order WHERE id > 100000 LIMIT 10
```

Pagination Sub-query

Both Oracle and SQLServer pagination need to be processed by sub-query, ShardingSphere supports pagination related sub-query.

- Oracle

Support rownum pagination:

```sql
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 > ?
```

Do not support rownum + BETWEEN pagination for now.

- SQLServer

Support TOP + ROW_NUMBER() OVER pagination:

```sql
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 OFFSET FETCH pagination after SQLServer 2012:

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

Do not support `WITH xxx AS (SELECT ...)` pagination. Because SQLServer automatically generated by Hibernate uses WITH statements, Hibernate SQLServer pagination or two TOP + sub-query pagination is not available now.

- MySQL, PostgreSQL

Both MySQL and PostgreSQL support LIMIT pagination, no need for sub-query:

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

6.4 Distributed Transaction

6.4.1 Background

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

- Atomicity guarantees that each transaction is treated as a single unit, which either succeeds completely, or fails completely;
- Consistency ensures that a transaction can only bring the database from one valid state to another, maintaining database invariants;
- Isolation ensures that concurrent execution of transactions leaves the database in the same state that would have been obtained if the transactions were executed sequentially;

- Durability guarantees that once a transaction has been committed, it will remain committed even in the case of a system failure (e.g., power outage or crash).

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.

Local Transaction

It means let each data node to manage their own transactions on the premise that any distributed transaction manager is not on. They do not have any coordination and communication ability, or know other data nodes have succeeded or not. Though without any consumption in performance, local transactions are not capable enough in high consistency and eventual consistency.

2PC Transaction

The earliest distributed transaction model of XA standard is `X/Open Distributed Transaction Processing (DTP)` model brought up by `X/Open`, XA for short.

Distributed transaction based on XA standard has little intrusion to businesses. Its biggest advantage is the transparency to users, who can use distributed transactions based on XA standard just as local transactions. XA standard can strictly guarantee ACID features of transactions.

That guarantee can be a double-edged sword. It is more proper in the implementation of short transactions with fixed time, because it will lock all the resources needed during the implementation process. For long transactions, data monopolization during its implementation will lead to an obvious concurrency performance recession for business systems depend on hot spot data. Therefore, in high concurrency situations that take performance as the highest, distributed transaction based on XA standard is not the best choice.

BASE Transaction

If we call transactions that satisfy ACID features as hard transactions, then transactions based on BASE features are called soft transactions. BASE is the abbreviation of basically available, soft state and eventually consistent those there factors.

- Basically available feature means not all the participants of distributed transactions have to be online at the same time.
- Soft state feature permits some time delay in system renewal, which may not be noticed by users.
- Eventually consistent feature of systems is usually guaranteed by message availability.

There is a high requirement for isolation in ACID transactions: all the resources must be locked during the transaction implementation process. The concept of BASE transactions is uplifting mutex operation from resource level to business level through business logic. Broaden the requirement for high consistency to exchange the rise in system throughput.

Highly consistent transactions based on ACID and eventually consistent transactions based on BASE are not silver bullets, and they can only take the most effect in the most appropriate situations. The detailed distinctions between them are illustrated in the following table to help developers to choose technically:

	Local transaction	*2PC (3PC) transaction*	*BASE transaction*
Business trans formation	None	None	Relevant interface
Co nsistency	Not support	Support	Eventual consistency
Isolation	Not support	Support	Business-side guarantee
Co ncurrency pe rformance	No influence	Serious recession	Minor recession
Situation	Inconsistent operation at business side	Short transaction & low concurrency	Long transaction & high concurrency

6.4.2 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.

6.4.3 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.

6.4.4 Core Concept

Navigation

This chapter mainly introduces the core concepts of distributed transactions, including:

- XA transaction
- BASE transaction

XA

2PC transaction submit uses the DTP Model defined by X/OPEN, in which created AP (Application Program), TM (Transaction Manager) and RM (Resource Manager) can guarantee a high transaction consistency. TM and RM use XA protocol for bidirectional streaming. Compared with traditional local transactions, XA transactions have a prepared phase, where the database cannot only passively receive commands, but also notify the submitter whether the transaction can be accepted. TM can collect all the prepared results of branch transactions before submitting all of them together, which has guaranteed the distributed consistency.

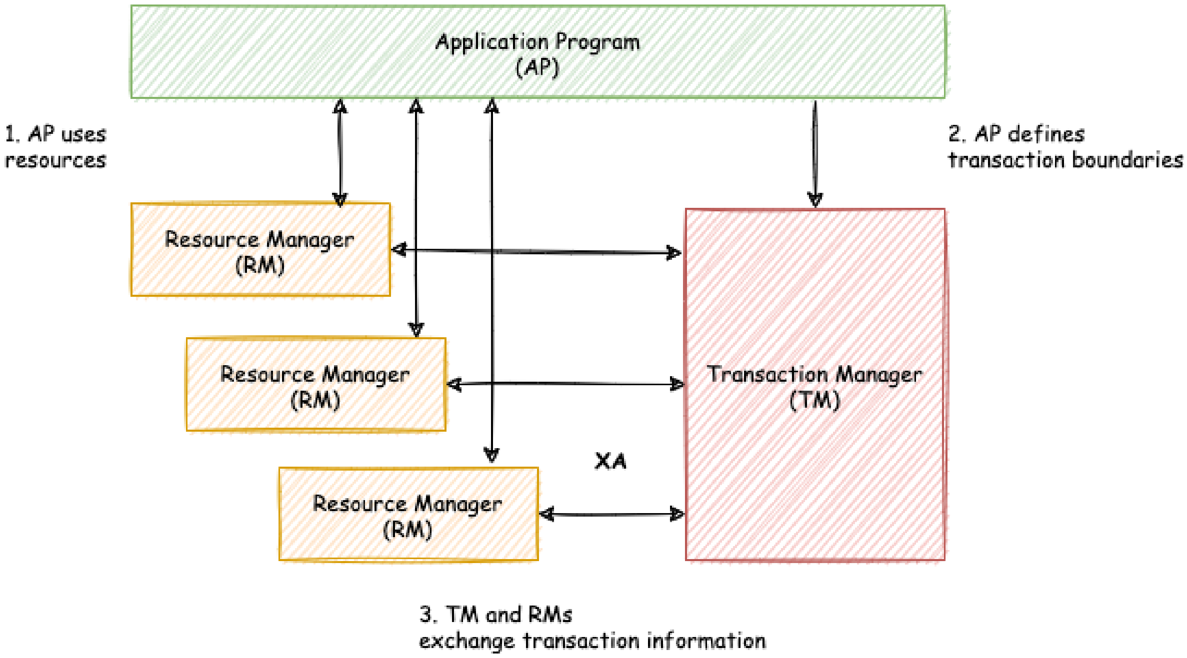

Figure4: 2PC XA model

Java implements the XA model through defining a JTA interface, in which `ResourceManager` requires an XA driver provided by database manufacturers and `TransactionManager` is provided by transaction manager manufacturers. Traditional transaction managers need to be bound with application server, which poises a high use cost. Built-in transaction managers have already been able to provide services through jar packages. Integrated with Apache ShardingSphere, it can guarantee the high consistency in cross-database transactions after sharding.

Usually, to use XA transaction, users must use its connection pool provided by transaction manager manufacturers. However, when Apache ShardingSphere integrates XA transactions, it has separated the management of XA transaction and its connection pool, so XA will not invade the applications.

BASE

A paper published in 2008 first mentioned on BASE transaction, it advocates the use of eventual consistency to instead of consistency when improve concurrency of transaction processing.

TCC and Sage are two regular implementations. They use reverse operation implemented by developers themselves to ensure the eventual consistency when data rollback. SEATA implements SQL reverse operation automatically, so that BASE transaction can be used without the intervention of developers.

Apache ShardingSphere integrates SEATA as solution of BASE transaction.

6.4.5 Use Norms

Background

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

Supported

- 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.

BASE

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.

6.5 Readwrite-splitting

6.5.1 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.

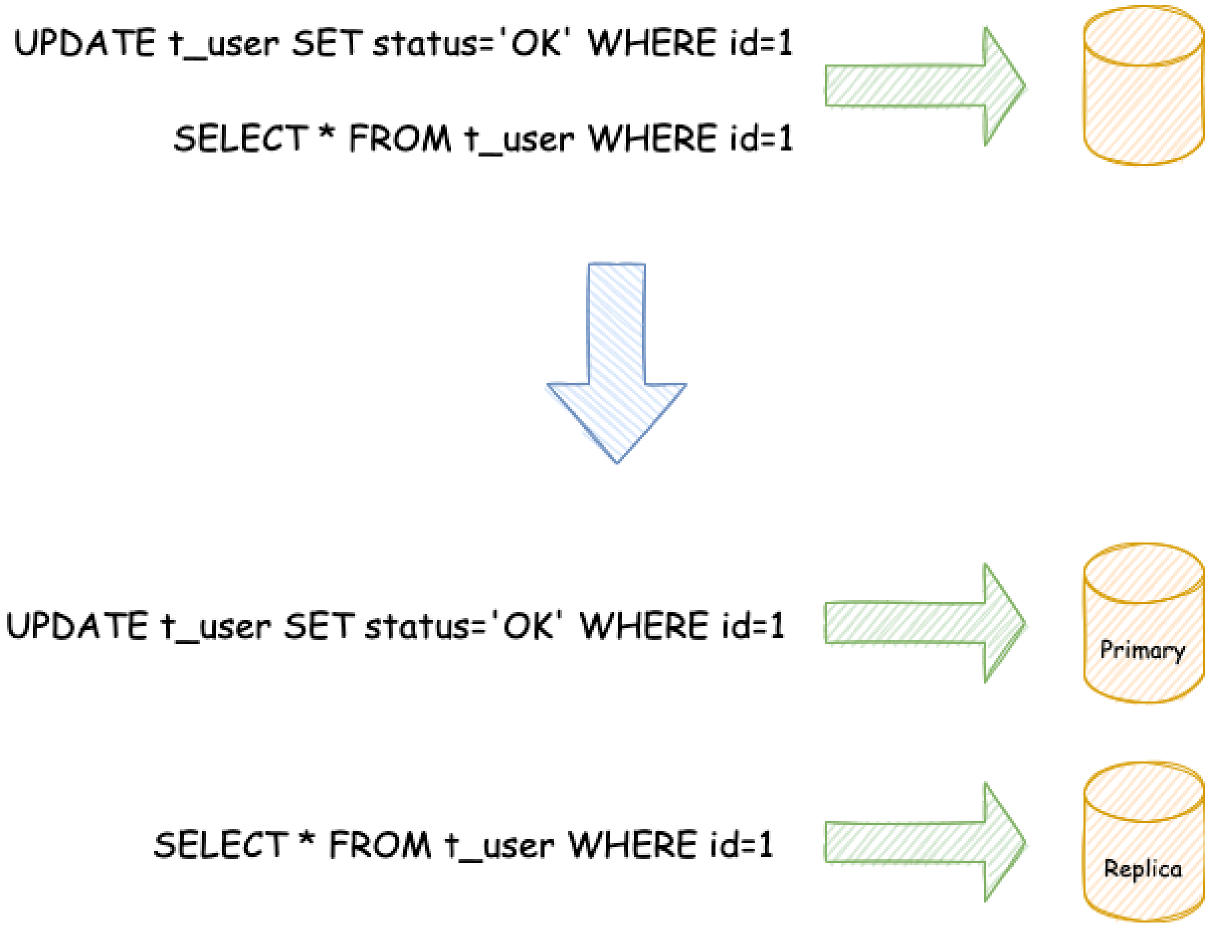

Figure5: Background

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.

6.5.2 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.

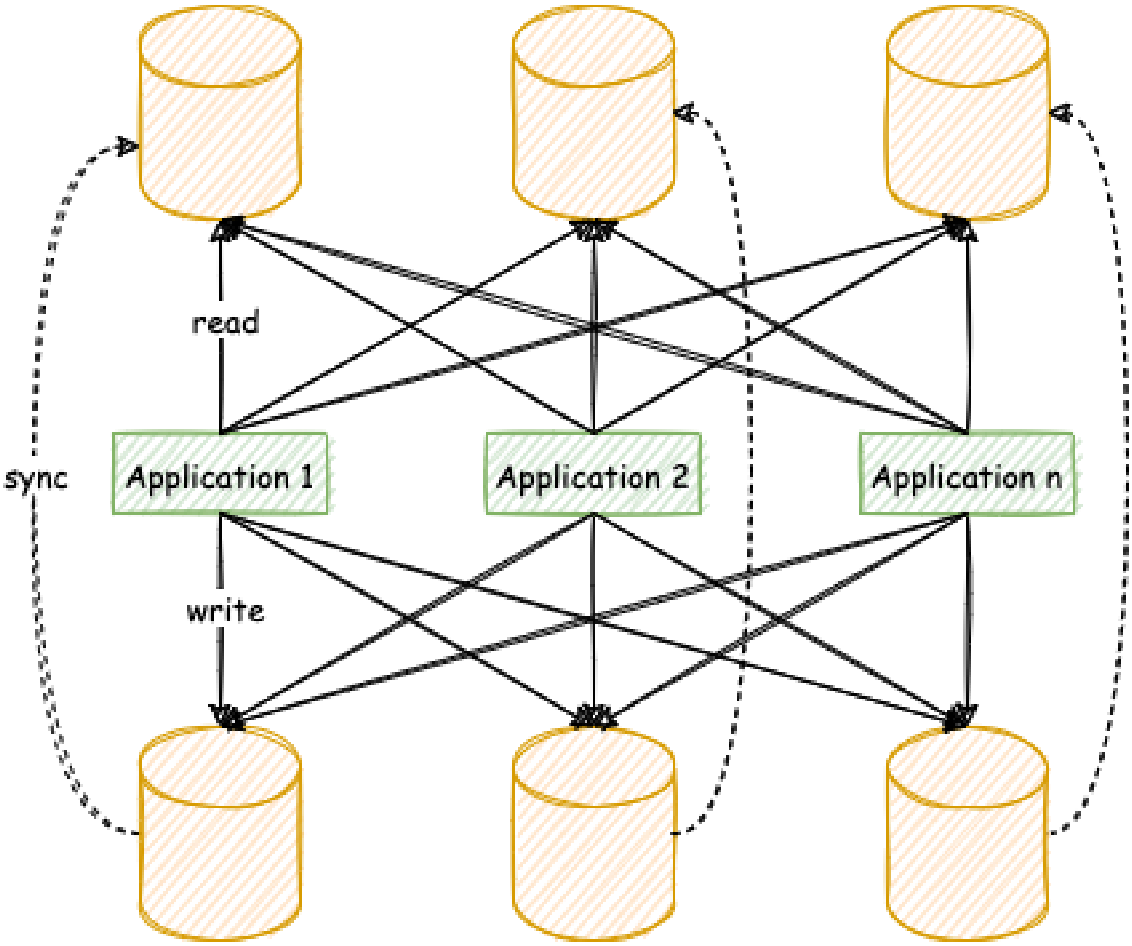

Figure6: Challenges

6.5.3 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.

6.5.4 Core Concept

Primary Database

It refers to the database used in data insertion, update and deletion. It only supports single primary database for now.

Replica Database

It refers to the database used in data query. It supports multiple replica databases.

Primary Replica Replication

It refers to the operation to asynchronously replicate data from the primary database to the replica database. Because of the asynchrony of primary-replica synchronization, there may be short-time data inconsistency between them.

Load Balance Strategy

Through this strategy, queries separated to different replica databases.

6.5.5 Use Norms

Supported

- Provide the readwrite-splitting configuration of one primary database with multiple replica databases, which can be used alone or with sharding table and database;
- Primary nodes need to be used for both reading and writing in the transaction;
- Forcible primary database route based on SQL Hint;

Unsupported

- Data replication between the primary and the replica databases;
- Data inconsistency caused by replication delay between databases;
- Double or multiple primary databases to provide write operation;
- The data for transaction across primary and replica nodes are inconsistent; In the readwrite-splitting model, primary nodes need to be used for both reading and writing in the transaction.

6.6 HA

6.6.1 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.

6.6.2 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.

6.6.3 Goal

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

6.6.4 Core Concept

high Availability Type

Apache ShardingSphere does not provide high availability solution of database, it reuses 3rd party high availability solution and auto-detect switch of primary and replica databases. Specifically, the ability of Apache ShardingSphere provided is database discovery, detect the primary and replica databases automatically, and updates the connection of compute nodes to the databases.

Dynamic Readwrite-Splitting

When high availability and readwrite-splitting are used together, there is unnecessary to configure specific primary and replica databases for readwrite-splitting. Highly available data sources will update the primary and replica databases of readwrite-splitting dynamically, and route the query and update SQL correctly.

6.7 Scaling

6.7.1 Background

There is a problem which how to migrate data from stand-alone database to sharding data nodes safely and simply; For applications which have used Apache ShardingSphere, scale out elastically is a mandatory requirement.

6.7.2 Challenges

Apache ShardingSphere provides great flexibility in sharding algorithms, but it gives a great challenge to scaling out. So it' s the first challenge that how to find a way can support kinds of sharding algorithms and scale data nodes efficiently.

What' s more, During the scaling process, it should not affect the running applications. So It is another big challenge for scaling to reduce the time window of data unavailability during the scaling as much as possible, or even completely unaware.

Finally, scaling should not affect the existing data. How to ensure the availability and correctness of data is the third challenge of scaling.

ShardingSphere-Scaling is a common solution for migrating or scaling data.

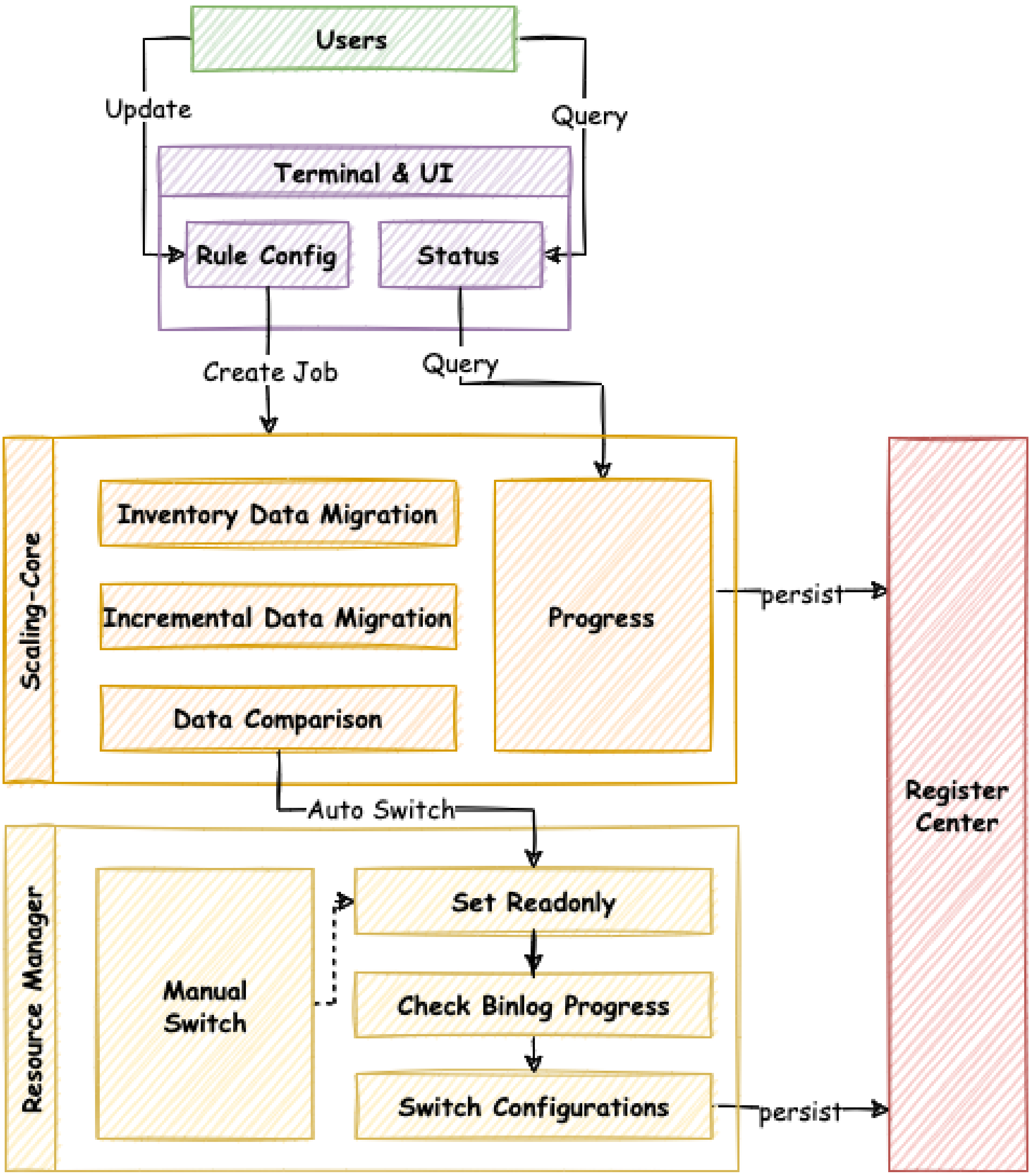

Figure7: Overview

6.7.3 Goal

The main design goal of ShardingSphere-Scaling is providing common solution which can support kinds of sharding algorithm and reduce the impact as much as possible during scaling.

6.7.4 Status

ShardingSphere-Scaling since version **4.1.0**. Current status is in **alpha** development.

6.7.5 Core Concept

Scaling Job

It refers one complete process of scaling data from old rule to new rule.

Inventory Data

It refers all existing data stored in data nodes before the scaling job started.

Incremental Data

It refers the new data generated by application during scaling job.

6.7.6 User Norms

Supported

- Migrate data outside into databases which managed by Apache ShardingSphere;
- Scale out data between data nodes of Apache ShardingSphere.

Unsupported

- Scale table without primary key, primary key can not be composite;
- Scale table with composite primary key;
- Do not support scale on in used databases, need to prepare a new database cluster for target.

6.8 Encryption

6.8.1 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.

6.8.2 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.

6.8.3 Goal

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

6.8.4 Core Concept

Logic Column

Column name used to encryption, it is the logical column identification in SQL. It includes cipher column(required), query assistant column(optional) and plain column(optional).

Cipher Column

Encrypted data column.

Query Assistant Column

Column used to assistant for query. For non-idempotent encryption algorithms with higher security level, irreversible idempotent columns provided for query.

Plain Column

Column used to persist plain column, for service provided during data encrypting. Should remove them after data clean.

6.8.5 Use Norms

Supported

- Encrypt/decrypt one or more columns in the database table;
- Compatible with all regular SQL.

Unsupported

- Need to process original inventory data before encryption;
- The value of encryption columns cannot support comparison, such as: >, <, ORDER BY, BETWEEN, LIKE, etc;
- The value of encryption columns cannot support calculation, such as AVG, SUM, and calculation expressions.

6.9 Shadow DB

6.9.1 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.

6.9.2 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.

6.9.3 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.

6.9.4 Core Concept

Pressure Testing Switcher

Pressure testing is a requirement for a specific period, turned on when needed.

Production Database

The database used for production data.

Shadow Database

The database for pressure testing data isolation.

Shadow Algorithm

The shadow algorithms are closely related to business, there are 2 types of shadow algorithms provided.

- Column based shadow algorithm

Recognize data from SQL and route to shadow databases. Suitable for test data driven scenario.

- Hint based shadow algorithm

Recognize comment from SQL and route to shadow databases. Suitable for identify passed by upstream system scenario.

6.9.5 Use Norms

Supported

- Note based shadow algorithm support all SQL;
- Column based shadow algorithm support part of SQL.

Unsupported

Note based shadow algorithm

- None

Column based shadow algorithm

- Does not support DDL;
- Does not support range, group and subquery, for example: BETWEEN, GROUP BY …HAVING…;

SQL support list:

- INSERT

SQL	*Supp orted*
INSERT INTO table (column,⋯) VALUES (value,⋯)	Y
INSERT INTO table (column,⋯) VALUES (value,⋯),(value,⋯),⋯	Y
INSERT INTO table (column,⋯) SELECT column1 from table1 where column1 = value1	N

- SELECT/UPDATE/DELETE

• Condition*	*SQL*	*S upp ort ed*
=	SELECT/UPDATE/DELETE ⋯ WHERE column = value	Y
LIKE/NOT LIKE	SELECT/UPDATE/DELETE ⋯ WHERE column LIKE/NOT LIKE value	Y
IN/NOT IN	SELECT/UPDATE/DELETE ⋯ WHERE column IN/NOT IN (value1,value2,⋯)	Y
BETWEEN	SELECT/UPDATE/DELETE ⋯ WHERE column BETWEEN value1 AND value2	N
GROUP BY ⋯HAVING⋯	SELECT/UPDATE/DELETE ⋯ WHERE ⋯GROUP BY column HAVING column > value	N
Subquery	SELECT/UPDATE/DELETE ⋯ WHERE column = (SELECT column FROM table WHERE column = value)	N

6.10 Observability

6.10.1 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. Observability and telemetry are the recommended operation way for them. APM (application performance monitoring) and metrics (statistical indicator monitoring) are important system health indicators.

APM is the abbreviation for application performance monitoring. It works for performance diagnosis of distributed systems, including chain demonstration, application topology analysis and so on.

Apache ShardingSphere is not responsible for gathering, storing and demonstrating APM data, but sends the core information of SQL parsing and enforcement to APM to process. In other words, Apache

ShardingSphere is only responsible for generating valuable data and submitting it to relevant systems through standard protocol. It can connect to APM systems in 3 ways.

- Use OpenTracing API

APM products facing OpenTracing protocol can all automatically connect to Apache ShardingSphere, like SkyWalking, Zipkin and Jaeger. In this way, users only need to configure the implementation of OpenTracing protocol at the start. Its advantage is the compatibility of all the products compatible of OpenTracing protocol, such as the APM demonstration system. If companies intend to implement their own APM systems, they only need to implement the OpenTracing protocol, and they can automatically show the chain tracing information of Apache ShardingSphere. Its disadvantage is that OpenTracing protocol is not stable in its development, has only a few new versions, and is too neutral to support customized products as native ones do.

- Use OpenTelemetry API

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' s automatic monitor agent

Cooperating with Apache SkyWalking team, Apache ShardingSphere team has realized `ShardingSphere` automatic monitor agent to automatically send application performance data to `SkyWalking`.

Metrics used to collect and display statistical indicator of cluster.

6.10.2 Challenges

APM 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.

6.10.3 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.

6.10.4 Core Concept

Agent

Based on bytecode enhance and plugin design to provide APM, metrics event tracing and log output. Enable agent to collection data and then can display data into integrated 3rd APM projects.

APM

APM is the abbreviation for application performance monitoring. It works for performance diagnosis of distributed systems, including chain demonstration, application topology analysis and so on. Use agent to send tracing data into 3rd party APM protocols.

Metrics

System statistical indicator which collected from agent. Write to time series databases periodically. 3rd party UI can display the metrics data simply.

7 User Manual

This chapter describes how to use projects of Apache ShardingSphere.

7.1 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.

7.1.1 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 schema name, mode configuration, data source map, rule configurations and properties.

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

```
String schemaName = "foo_schema"; // Indicate logic schema 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(schemaName, modeConfig, dataSourceMap, ruleConfigs, props);
```

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 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(schemaName, 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 Configuration

Root Configuration

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

Attributes:

• Name*	*Data Type*	*Description*	• DefaultValue*
type	S tring	Type of mode configurationValues could be: Memory, Standalone, Cluster	Memory
repository	Persi stRep osito ryCon figur ation	Persist repository configurationMemory type does not need persist, could be nullStandalone type uses StandalonePersistRepositoryConfigurationCluster type uses ClusterPersistRepositoryConfiguration	
overwrite	bo olean	Whether overwrite persistent configuration with local configuration	false

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	*DataType*	*Description*
type	String	Type of persist repository
namespace	String	Namespace of registry center
serverLists	String	Server lists of registry center
props	Properties	Properties of persist repository

Please refer to Builtin Persist Repository List for more details about type of repository.

Data Source

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

Example

In this example, the database driver is MySQL, and connection pool is HikariCP, which can be replaced with other database drivers and connection pools.

```java
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);

// Configure other data sources
...
```

Rules

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

Sharding

Root Configuration

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

Attributes:

Name	*DataType*	*Description*	*Def ault Va lue*
tables (+)	Collection<ShardingTableRuleConfiguration>	Sharding table rules	•
autoTables (+)	Collection<ShardingAutoTableRuleConfiguration>	Sharding automatic table rules	•
bind ingTableGroups (*)	Collection<String>	Binding table rules	E mpty
b roadcastTables (*)	Collection<String>	Broadcast table rules	E mpty
def aultDatabaseShardingStrategy (?)	Sharding StrategyConfiguration	Default database sharding strategy	Not shar ding
defaultTableSh ardingStrategy (?)	Sharding StrategyConfiguration	Default table sharding strategy	Not shar ding
defaultKeyGe nerateStrategy (?)	KeyG eneratorConfiguration	Default key generator	S nowf lake
default ShardingColumn (?)	String	Default sharding column name	None
shar dingAlgorithms (+)	Map<String, ShardingSphereAl gorithmConfiguration>	Sharding algorithm name and configurations	None
keyGenerators (?)	Map<String, ShardingSphereAl gorithmConfiguration>	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 Automatic 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.

Key Generate 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.

Readwrite-splitting

Root Configuration

Class name: ReadwriteSplittingRuleConfiguration

Attributes:

• Name*	*DataType*	*Description*
d ataSo urces (+)	Collectio n<ReadwriteSplittingData SourceRuleConfiguration>	Data sources of write and reads
loa dBala ncers (*)	Map<String, ShardingSpher eAlgorithmConfiguration>	Load balance algorithm name and configurations of replica data sources

Readwrite-splitting Data Source Configuration

Class name: ReadwriteSplittingDataSourceRuleConfiguration

Attributes:

Name	*D ataType*	*Description*	*Default Value*
name	String	Readwrite-splitting data source name	•
writeDat aSource-Name	String	Write sources source name	•
readData Source-Names (+)	Co llection <String>	Read sources source name list	•
loadB alancerName (?)	String	Load balance algo-rithm name of replica sources	Round robin load bal-ance algorithm

Please refer to Built-in Load Balance Algorithm List for more details about type of algorithm. Please refer to Use Norms for more details about query consistent routing.

HA

TODO

Encryption

Root Configuration

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

Attributes:

Name	*DataType*	*Description*	*D ef au lt Va lu e*
tables (+)	Collectio n<EncryptTableRu leConfiguration>	Encrypt table rule configurations	
encr yptors (+)	Map<String, Shard ingSphereAlgorit hmCon-figuration>	Encrypt algorithm name and configurations	
que ryWith Cipher Col-umn (?)	boolean	Whether query with cipher column for data encrypt. User you can use plaintext to query if have	tr ue

Encrypt Table Rule Configuration

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

Attributes:

• Name*	*DataType*	*Description*
name	String	Table name
co lumns (+)	Collection <EncryptColumnRuleConfiguration>	Encrypt column rule configurations

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

Encrypt Algorithm Configuration

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

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.

Shadow DB

Root Configuration

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

Attributes:

Name	*DataType*	*Description*	*Def ault Va lue*
enable	boolean	Shadow database switch	f alse
d ataSources	Map<String, ShadowD ata-SourceConfiguration>	Shadow data source mapping name and configuration	
tables	Map<String, Sh adowTableCon-figuration>	Shadow table name and config-uration	
defaul tShadowAlg orithmName	String	default shadow algorithm name	
shadow Algo-rithms	Map<String, ShardingSphere Al-gorithmConfiguration>	Shadow algorithm name and configuration	

Shadow Data Source Configuration

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

Attributes:

Name	*DataType*	*Description*
sourceDataSourceName	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	*DataType*	*Description*
da taSourceNames	Colle ction<String>	Shadow table location shadow data source names
shadowA lgorithmNames	Colle ction<String>	Shadow table location shadow algorithm names

Shadow Algorithm Configuration

Please refer to Built-in Shadow Algorithm List.

Mixed Rules

Configuration Item Explanation

```
/* Data source configuration */
HikariDataSource writeDataSource0 = new HikariDataSource();
writeDataSource0.setDriverClassName("com.mysql.jdbc.Driver");
writeDataSource0.setJdbcUrl("jdbc:mysql://localhost:3306/db0?serverTimezone=UTC&
useSSL=false&useUnicode=true&characterEncoding=UTF-8");
writeDataSource0.setUsername("root");
writeDataSource0.setPassword("");

HikariDataSource writeDataSource1 = new HikariDataSource();
// ...Omit specific configuration.

HikariDataSource read0OfwriteDataSource0 = new HikariDataSource();
// ...Omit specific configuration.

HikariDataSource read1OfwriteDataSource0 = new HikariDataSource();
// ...Omit specific configuration.

HikariDataSource read0OfwriteDataSource1 = new HikariDataSource();
// ...Omit specific configuration.

HikariDataSource read1OfwriteDataSource1 = new HikariDataSource();
// ...Omit specific configuration.

Map<String, DataSource> datasourceMaps = new HashMap<>(6);

datasourceMaps.put("write_ds0", writeDataSource0);
datasourceMaps.put("write_ds0_read0", read0OfwriteDataSource0);
datasourceMaps.put("write_ds0_read1", read1OfwriteDataSource0);

datasourceMaps.put("write_ds1", writeDataSource1);
datasourceMaps.put("write_ds1_read0", read0OfwriteDataSource1);
datasourceMaps.put("write_ds1_read1", read1OfwriteDataSource1);

/* Sharding rule configuration */
// The enumeration value of `ds_$->{0..1}` is the name of the logical data source
configured with read-query
ShardingTableRuleConfiguration tOrderRuleConfiguration = new
ShardingTableRuleConfiguration("t_order", "ds_${0..1}.t_order_${[0, 1]}");
tOrderRuleConfiguration.setKeyGenerateStrategy(new
KeyGenerateStrategyConfiguration("order_id", "snowflake"));
```

```
tOrderRuleConfiguration.setTableShardingStrategy(new
StandardShardingStrategyConfiguration("order_id", "tOrderInlineShardingAlgorithm
"));
Properties tOrderShardingInlineProps = new Properties();
tOrderShardingInlineProps.setProperty("algorithm-expression", "t_order_${order_id %
2}");
ruleConfiguration.getShardingAlgorithms().putIfAbsent(
"tOrderInlineShardingAlgorithm", new ShardingSphereAlgorithmConfiguration("INLINE",
tOrderShardingInlineProps));

ShardingTableRuleConfiguration tOrderItemRuleConfiguration = new
ShardingTableRuleConfiguration("t_order_item", "ds_${0..1}.t_order_item_${[0, 1]}
");
tOrderItemRuleConfiguration.setKeyGenerateStrategy(new
KeyGenerateStrategyConfiguration("order_item_id", "snowflake"));
tOrderRuleConfiguration.setTableShardingStrategy(new
StandardShardingStrategyConfiguration("order_item_id",
"tOrderItemInlineShardingAlgorithm"));
Properties tOrderItemShardingInlineProps = new Properties();
tOrderItemShardingInlineProps.setProperty("algorithm-expression", "t_order_item_$
{order_item_id % 2}");
ruleConfiguration.getShardingAlgorithms().putIfAbsent(
"tOrderItemInlineShardingAlgorithm", new ShardingSphereAlgorithmConfiguration(
"INLINE",tOrderItemShardingInlineProps));

ShardingRuleConfiguration shardingRuleConfiguration = new
ShardingRuleConfiguration();
shardingRuleConfiguration.getTables().add(tOrderRuleConfiguration);
shardingRuleConfiguration.getTables().add(tOrderItemRuleConfiguration);
shardingRuleConfiguration.getBindingTableGroups().add("t_order, t_order_item");
shardingRuleConfiguration.getBroadcastTables().add("t_bank");
// Default database strategy configuration
shardingRuleConfiguration.setDefaultDatabaseShardingStrategy(new
StandardShardingStrategyConfiguration("user_id", "default_db_strategy_inline"));
Properties defaultDatabaseStrategyInlineProps = new Properties();
defaultDatabaseStrategyInlineProps.setProperty("algorithm-expression", "ds_${user_
id % 2}");
shardingRuleConfiguration.getShardingAlgorithms().put("default_db_strategy_inline",
new ShardingSphereAlgorithmConfiguration("INLINE",
defaultDatabaseStrategyInlineProps));

// Key generate algorithm configuration
Properties snowflakeProperties = new Properties();
snowflakeProperties.setProperty("worker-id", "123");
shardingRuleConfiguration.getKeyGenerators().put("snowflake", new
ShardingSphereAlgorithmConfiguration("SNOWFLAKE", snowflakeProperties));

/* Data encrypt rule configuration */
```

```
Properties encryptProperties = new Properties();
encryptProperties.setProperty("aes-key-value", "123456");
EncryptColumnRuleConfiguration columnConfigAes = new
EncryptColumnRuleConfiguration("user_name", "user_name", "", "user_name_plain",
"name_encryptor");
EncryptColumnRuleConfiguration columnConfigTest = new
EncryptColumnRuleConfiguration("pwd", "pwd", "assisted_query_pwd", "", "pwd_
encryptor");
EncryptTableRuleConfiguration encryptTableRuleConfig = new
EncryptTableRuleConfiguration("t_user", Arrays.asList(columnConfigAes,
columnConfigTest));
// Data encrypt algorithm configuration
Map<String, ShardingSphereAlgorithmConfiguration> encryptAlgorithmConfigs = new
LinkedHashMap<>(2, 1);
encryptAlgorithmConfigs.put("name_encryptor", new
ShardingSphereAlgorithmConfiguration("AES", encryptProperties));
encryptAlgorithmConfigs.put("pwd_encryptor", new
ShardingSphereAlgorithmConfiguration("assistedTest", encryptProperties));
EncryptRuleConfiguration encryptRuleConfiguration = new
EncryptRuleConfiguration(Collections.singleton(encryptTableRuleConfig),
encryptAlgorithmConfigs);

/* Readwrite-splitting rule configuration */
ReadwriteSplittingDataSourceRuleConfiguration dataSourceConfiguration1 = new
ReadwriteSplittingDataSourceRuleConfiguration("ds_0", "write_ds0", Arrays.asList(
"write_ds0_read0", "write_ds0_read1"), "roundRobin");
ReadwriteSplittingDataSourceRuleConfiguration dataSourceConfiguration2 = new
ReadwriteSplittingDataSourceRuleConfiguration("ds_1", "write_ds0", Arrays.asList(
"write_ds1_read0", "write_ds1_read0"), "roundRobin");

// Load balance algorithm configuration
Map<String, ShardingSphereAlgorithmConfiguration> loadBalanceMaps = new HashMap<>
(1);
loadBalanceMaps.put("roundRobin", new ShardingSphereAlgorithmConfiguration("ROUND_
ROBIN", new Properties()));

ReadwriteSplittingRuleConfiguration readWriteSplittingRuleConfiguration = new
ReadwriteSplittingRuleConfiguration(Arrays.asList(dataSourceConfiguration1,
dataSourceConfiguration2), loadBalanceMaps);

/* Other Properties configuration */
Properties otherProperties = new Properties();
otherProperties.setProperty("sql-show", "true");

/* The variable `shardingDataSource` is the logic data source referenced by other
frameworks(such as ORM, JPA, etc.) */
DataSource shardingDataSource = ShardingSphereDataSourceFactory.
createDataSource(datasourceMaps, Arrays.asList(shardingRuleConfiguration,
readWriteSplittingRuleConfiguration, encryptRuleConfiguration), otherProperties);
```

7.1.2 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 schema 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.

```
# Alias of the datasource in JDBC.
# Through this parameter to connect, ShardingSphere-JDBC and ShardingSphere-Proxy.
# Default value: logic_db
schemaName (?):

mode:

dataSources:

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

```yaml
props:
  key_1: value_1
  key_2: value_2
```

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.

Create Data Source

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

```java
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 Configuration

Configuration Item Explanation

```yaml
mode (?): # Default value is Memory
  type: # Type of mode configuration. Values could be: Memory, Standalone, Cluster
  repository (?): # Persist repository configuration. Memory type does not need
persist
  overwrite: # Whether overwrite persistent configuration with local configuration
```

Memory Mode

```yaml
mode:
  type: Memory
```

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

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

Please refer to Builtin Persist Repository List for more details about type of repository.

Data Source

It is divided into single data source configuration and multi data source configuration. ShardingSphere-JDBC Supports all JDBC drivers and database connection pools.

In this example, the database driver is MySQL, and connection pool is HikariCP, which can be replaced with other database drivers and connection pools.

Single Data Source Configuration

Used for data encryption rules.

Configuration Item Explanation

```
dataSource: # <!!Data source pool implementation class> `!!` means class
instantiation
  driverClassName: # Class name of database driver, ref property of connection pool
  url: # Database URL, ref property of connection pool
  username: # Database username, ref property of connection pool
  password: # Database password, ref property of connection pool
  # ... Other properties for data source pool
```

Example

```
dataSource: !!com.zaxxer.hikari.HikariDataSource
  driverClassName: com.mysql.jdbc.Driver
  jdbcUrl: jdbc:mysql://localhost:3306/ds
  username: root
  password:
```

Multiple Data Source Configuration

Used for fragmentation, readwrite-splitting and other rules. If features such as encryption and sharding are used in combination, a multi data source configuration should be used.

Configuration Item Explanation

```
dataSources: # Data sources configuration, multiple <data-source-name> available
  <data-source-name>: # <!!Data source pool implementation class> `!!` means class
instantiation
    driverClassName: # Class name of database driver, ref property of connection
pool
    url: # Database URL, ref property of connection pool
    username: # Database username, ref property of connection pool
    password: # Database password, ref property of connection pool
    # ... Other properties for data source pool
```

Example

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

Configuration Item Explanation

```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.
        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
```

```yaml
      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
      # ...
```

Readwrite-splitting

Configuration Item Explanation

```yaml
rules:
- !READWRITE_SPLITTING
  dataSources:
    <data-source-name> (+): # Logic data source name of readwrite-splitting
      writeDataSourceName: # Write data source name
      readDataSourceNames:
        - <read-data-source-name> (+) # Read data source name
      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 Use Norms for more details about query consistent routing.

HA

TODO

Encryption

Configuration Item Explanation

```
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

  # 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.

Shadow DB

Configuration Item Explanation

```
rules:
  - !SHADOW
    enable: # Shadow function switch. Optional values: true/false, the default is
false
    dataSources:
      shadowDataSource:
        sourceDataSourceName: # Production data source name
        shadowDataSourceName: # Shadow data source name
    tables:
      <table-name>:
        dataSourceNames: # Shadow table location shadow data source names
          - <shadow-data-source>
        shadowAlgorithmNames: # Shadow table location shadow algorithm names
          - <shadow-algorithm-name>
    defaultShadowAlgorithmName: # Default shadow algorithm name
    shadowAlgorithms:
      <shadow-algorithm-name> (+): # Shadow algorithm name
        type: # Shadow algorithm type
        props: # Shadow algorithm property configuration
        # ...
```

Mixed Rules

The overlay between rule items in a mixed configuration is associated by the data source name and the table name.

If the previous rule is aggregation-oriented, 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 aggregation-oriented, the next rule needs to use the aggregated logical table name configured by the previous rule when configuring the table.

Configuration Item Explanation

```
dataSources: # Configure the real data source name.
  write_ds:
    # ...Omit specific configuration.
  read_ds_0:
    # ...Omit specific configuration.
  read_ds_1:
    # ...Omit specific configuration.

rules:
```

```yaml
  - !SHARDING # Configure data sharding rules.
    tables:
      t_user:
        actualDataNodes: ds.t_user_${0..1} # Data source name 'ds' uses the logical
data source name of the readwrite-splitting configuration.
        tableStrategy:
          standard:
            shardingColumn: user_id
            shardingAlgorithmName: t_user_inline
    shardingAlgorithms:
      t_user_inline:
        type: INLINE
        props:
          algorithm-expression: t_user_${user_id % 2}

  - !ENCRYPT # Configure data encryption rules.
    tables:
      t_user: # Table `t_user` is the name of the logical table that uses the data
sharding configuration.
        columns:
          pwd:
            plainColumn: plain_pwd
            cipherColumn: cipher_pwd
            encryptorName: encryptor_aes
    encryptors:
      encryptor_aes:
        type: aes
        props:
          aes-key-value: 123456abc

  - !READWRITE_SPLITTING # Configure readwrite-splitting rules.
    dataSources:
      ds: # The logical data source name 'ds' for readwrite-splitting is used in
data sharding.
        writeDataSourceName: write_ds # Use the real data source name 'write_ds'.
        readDataSourceNames:
          - read_ds_0 # Use the real data source name 'read_ds_0'.
          - read_ds_1 # Use the real data source name 'read_ds_1'.
        loadBalancerName: roundRobin
    loadBalancers:
      roundRobin:
        type: ROUND_ROBIN

props:
  sql-show: true
```

7.1.3 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>
```

Use ShardingSphere Data Source in Spring

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 Configuration

Default is Memory mode.

Configuration Item Explanation

```
spring.shardingsphere.mode.type= # Type of mode configuration. Values could be:
Memory, Standalone, Cluster
spring.shardingsphere.mode.repository= # Persist repository configuration. Memory
type does not need persist
spring.shardingsphere.mode.overwrite= # Whether overwrite persistent configuration
with local configuration
```

Memory Mode

```
spring.shardingsphere.mode.type=Memory
```

Standalone Mode

```
spring.shardingsphere.mode.type=Standalone
spring.shardingsphere.mode.repository.type= # Type of persist repository
spring.shardingsphere.mode.repository.props.<key>= # Properties of persist
repository
spring.shardingsphere.mode.overwrite= # Whether overwrite persistent configuration
with local configuration
```

Cluster Mode

```
spring.shardingsphere.mode.type=Cluster
spring.shardingsphere.mode.repository.type= # Type of persist repository
spring.shardingsphere.mode.repository.namespace= # Namespace of registry center
spring.shardingsphere.mode.repository.serverLists= # Server lists of registry
center
spring.shardingsphere.mode.repository.props.<key>= # Properties of persist
repository
spring.shardingsphere.mode.overwrite= # Whether overwrite persistent configuration
with local configuration
```

Please refer to Builtin Persist Repository List for more details about type of repository.

Data Source

Use Native Data Source

Configuration Item Explanation

```
spring.shardingsphere.datasource.names= # Actual data source name, multiple split
by `,`

# <actual-data-source-name> indicate name of data source name
spring.shardingsphere.datasource.<actual-data-source-name>.type= # Full class name
of database connection pool
spring.shardingsphere.datasource.<actual-data-source-name>.driver-class-name= #
Class name of database driver, ref property of connection pool
spring.shardingsphere.datasource.<actual-data-source-name>.jdbc-url= # Database
URL, ref property of connection pool
spring.shardingsphere.datasource.<actual-data-source-name>.username= # Database
username, ref property of connection pool
```

```
spring.shardingsphere.datasource.<actual-data-source-name>.password= # Database
password, ref property of connection pool
spring.shardingsphere.datasource.<actual-data-source-name>.<xxx>= # ... Other
properties for data source pool
```

Example

In this example, the database driver is MySQL, and connection pool is HikariCP, which can be replaced with other database drivers and connection pools.

```
# Configure actual data sources
spring.shardingsphere.datasource.names=ds1,ds2

# Configure the 1st data source
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 2nd data source
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=
```

Use JNDI Data Source

If developer plan to use ShardingSphere-JDBC in Web Server (such as Tomcat) with JNDI data source, `spring.shardingsphere.datasource.${datasourceName}.jndiName` can be used as an alternative to series of configuration of data source.

Configuration Item Explanation

```
spring.shardingsphere.datasource.names= # Actual data source name, multiple split
by `,`

# <actual-data-source-name> indicate name of data source name
spring.shardingsphere.datasource.<actual-data-source-name>.jndi-name= # JNDI of
data source
```

Example

```
# Configure actual data sources
spring.shardingsphere.datasource.names=ds1,ds2

# Configure the 1st data source
spring.shardingsphere.datasource.ds1.jndi-name=java:comp/env/jdbc/ds1
# Configure the 2nd data source
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

Configuration Item Explanation

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

Readwrite splitting

Configuration Item Explanation

```
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>.primary-data-source-name= # Write data source name
spring.shardingsphere.rules.readwrite-splitting.data-sources.<readwrite-splitting-
data-source-name>.replica-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
```

Please refer to Built-in Load Balance Algorithm List for more details about type of algorithm. Please refer to Use Norms for more details about query consistent routing.

HA

TODO

Encryption

Configuration Item Explanation

```
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.

Shadow DB

Configuration Item Explanation

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

spring.shardingsphere.rules.shadow.enable= # Shadow DB switch. Optional values:
true/false, the default is false

spring.shardingsphere.rules.shadow.data-sources.shadow-data-source.source-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
```

Mixed Rules

Configuration Item Explanation

```
# data source configuration
spring.shardingsphere.datasource.names= write-ds0,write-ds1,write-ds0-read0,write-ds1-read0

spring.shardingsphere.datasource.write-ds0.url= # Database URL connection
spring.shardingsphere.datasource.write-ds0.type=  # Database connection pool type name
spring.shardingsphere.datasource.write-ds0.driver-class-name= # Database driver class name
spring.shardingsphere.datasource.write-ds0.username= # Database username
spring.shardingsphere.datasource.write-ds0.password= # Database password
spring.shardingsphere.datasource.write-ds0.xxx=  # Other properties of database connection pool

spring.shardingsphere.datasource.write-ds1.url= # Database URL connection
# ...Omit specific configuration.

spring.shardingsphere.datasource.write-ds0-read0.url= # Database URL connection
# ...Omit specific configuration.

spring.shardingsphere.datasource.write-ds1-read0.url= # Database URL connection
# ...Omit specific configuration.

# Sharding rules configuration
# Databases sharding strategy
spring.shardingsphere.rules.sharding.default-database-strategy.standard.sharding-column=user_id
spring.shardingsphere.rules.sharding.default-database-strategy.standard.sharding-algorithm-name=default-database-strategy-inline
# Binding table rules configuration ,and multiple groups of binding-tables configured with arrays
spring.shardingsphere.rules.sharding.binding-tables[0]=t_user,t_user_detail
spring.shardingsphere.rules.sharding.binding-tables[1]= # Binding table names, multiple table name are separated by commas
spring.shardingsphere.rules.sharding.binding-tables[x]= # Binding table names, multiple table name are separated by commas
# Broadcast table rules configuration
spring.shardingsphere.rules.sharding.broadcast-tables= # Broadcast table names, multiple table name are separated by commas

# Table sharding strategy
# The enumeration value of `ds_$->{0..1}` is the name of the logical data source configured with readwrite-splitting
spring.shardingsphere.rules.sharding.tables.t_user.actual-data-nodes=ds_$->{0..1}.t_user_$->{0..1}
```

```
spring.shardingsphere.rules.sharding.tables.t_user.table-strategy.standard.
sharding-column=user_id
spring.shardingsphere.rules.sharding.tables.t_user.table-strategy.standard.
sharding-algorithm-name=user-table-strategy-inline

# Data encrypt configuration
# Table `t_user` is the name of the logical table that uses for data sharding
configuration.
spring.shardingsphere.rules.encrypt.tables.t_user.columns.user_name.cipher-
column=user_name
spring.shardingsphere.rules.encrypt.tables.t_user.columns.user_name.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

# Data encrypt algorithm configuration
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

# Key generate strategy configuration
spring.shardingsphere.rules.sharding.tables.t_user.key-generate-strategy.
column=user_id
spring.shardingsphere.rules.sharding.tables.t_user.key-generate-strategy.key-
generator-name=snowflake

# Sharding algorithm configuration
spring.shardingsphere.rules.sharding.sharding-algorithms.default-database-strategy-
inline.type=INLINE
# The enumeration value of `ds_$->{user_id % 2}` is the name of the logical data
source configured with readwrite-splitting
spring.shardingsphere.rules.sharding.sharding-algorithms.default-database-strategy-
inline.algorithm-expression=ds$->{user_id % 2}
spring.shardingsphere.rules.sharding.sharding-algorithms.user-table-strategy-
inline.type=INLINE
spring.shardingsphere.rules.sharding.sharding-algorithms.user-table-strategy-
inline.algorithm-expression=t_user_$->{user_id % 2}

# Key generate algorithm configuration
spring.shardingsphere.rules.sharding.key-generators.snowflake.type=SNOWFLAKE
spring.shardingsphere.rules.sharding.key-generators.snowflake.props.worker-id=123

# read query configuration
# ds_0,ds_1 is the logical data source name of the readwrite-splitting
```

```
spring.shardingsphere.rules.readwrite-splitting.data-sources.ds_0.write-data-
source-name=write-ds0
spring.shardingsphere.rules.readwrite-splitting.data-sources.ds_0.read-data-source-
names=write-ds0-read0
spring.shardingsphere.rules.readwrite-splitting.data-sources.ds_0.load-balancer-
name=read-random
spring.shardingsphere.rules.readwrite-splitting.data-sources.ds_1.write-data-
source-name=write-ds1
spring.shardingsphere.rules.readwrite-splitting.data-sources.ds_1.read-data-source-
names=write-ds1-read0
spring.shardingsphere.rules.readwrite-splitting.data-sources.ds_1.load-balancer-
name=read-random

# Load balance algorithm configuration
spring.shardingsphere.rules.readwrite-splitting.load-balancers.read-random.
type=RANDOM
```

7.1.4 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-5.0.0.xsd

<shardingsphere:data-source />

N ame	• Type*	*Description*
id	Attribute	Spring Bean Id
sch ema- 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 Configurat ion 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" schema-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 ShardingSphere Data Source in Spring

Same with Spring Boot Starter.

Mode Configuration

Configuration Item Explanation

Namespace: http://shardingsphere.apache.org/schema/shardingsphere/datasource/datasource-5.0.0.xsd

<shardingsphere:mode />

Name	*Ty pe*	*Description*	*D efault Value*
type	Att rib ute	Type of mode configuration. Values could be: Memory, Standalone, Cluster	
reposi tory-ref (?)	Att rib ute	Persist repository configuration. Memory type does not need persist	
o verwrite (?)	Att rib ute	Whether overwrite persistent configuration with local configuration	false

Memory Mode

It is the default value.

Example

```xml
<?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"
       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" schema-name="foo_schema" data-source-names=
"..." rule-refs="..." />
</beans>
```

Standalone Mode

Configuration Item Explanation

Namespace: http://shardingsphere.apache.org/schema/shardingsphere/mode-repository/standalone-5.0.0.xsd

Name	*Type*	*Description*
id	Attribute	Name of persist repository bean
type	Attribute	Type of persist repository
props (?)	Tag	Properties of persist repository

Example

```xml
<?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">target</prop>
        </props>
    </standalone:repository>

    <shardingsphere:data-source id="ds" schema-name="foo_schema" data-source-names=
"..." rule-refs="..." >
        <shardingsphere:mode type="Standalone" repository-ref="standaloneRepository
" overwrite="true" />
    </shardingsphere:data-source>
</beans>
```

Cluster Mode

Configuration Item Explanation

Namespace: http://shardingsphere.apache.org/schema/shardingsphere/mode-repository/cluster-5.0.0.xsd

Name	*Type*	*Description*
id	Attribute	Name of persist repository bean
type	Attribute	Type of persist repository
namespace	Attribute	Namespace of registry center
server-lists	Attribute	Server lists of registry center
props (?)	Tag	Properties of persist repository

Example

```
<?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" schema-name="foo_schema" data-source-names=
"..." rule-refs="...">
        <shardingsphere:mode type="Cluster" repository-ref="clusterRepository"
overwrite="true" />
```

```
    </shardingsphere:data-source>
</beans>
```

Please refer to Builtin Persist Repository List for more details about type of repository.

Data Source

Any data source configured as spring bean can be cooperated with spring namespace.

Example

In this example, the database driver is MySQL, and connection pool is HikariCP, which can be replaced with other database drivers and connection pools.

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

    <shardingsphere:data-source id="ds" schema-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

Configuration Item Explanation

Namespace: http://shardingsphere.apache.org/schema/shardingsphere/sharding/sharding-5.0.0.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 sharding 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.

Readwrite-splitting

Configuration Item Explanation

Namespace: http://shardingsphere.apache.org/schema/shardingsphere/readwrite-splitting/readwrite-splitting-5.0.0.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	*Ty pe*	*Description*
id	Att rib ute	Readwrite-splitting data source rule name
write-da ta-source-name	Att rib ute	Write data source name
read-dat a-source-names	Att rib ute	Read data source names, multiple data source names separated with comma
load-balance - algorithm-ref	Att rib ute	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 Use Norms for more details about query consistent routing.

HA

TODO

Encryption

Configuration Item Explanation

Namespace: http://shardingsphere.apache.org/schema/shardingsphere/encrypt/encrypt-5.0.0.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.

Shadow DB

Configuration Item Explanation

Namespace: http://shardingsphere.apache.org/schema/shardingsphere/shadow/shadow-5.0.0.xsd

<shadow:rule />

Name	*Type*	*Description*
id	Attribute	Spring Bean Id
enable	Attribute	Shadow DB switch. Optional values: true/false, the default is false
d ata-source(?)	Tag	Shadow data source configuration
defaul t-shadow-algo rithm-name(?)	Tag	Default shadow algorithm configuration
sh adow-table(?)	Tag	Shadow table configuration

<shadow:data-source />

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

<shadow:default-shadow-algorithm-name /> | *Name* | *Type* | *Description* | | —-| ——| ——| | name | Attribute | Default shadow algorithm name |

<shadow:shadow-table />

Name	*Type*	*Description*
name	Attribute	Shadow table name
data-sources	Attribute	Shadow table location shadow data source names (multiple values are separated by “,”)
algorithm (?)	Tag	Shadow table location shadow algorithm configuration

<shadow:algorithm />

Name	*Type*	*Description*
shadow-algorithm-ref	Attribute	Shadow table location shadow algorithm name

<shadow:shadow-algorithm />

Name	*Type*	*Description*
id	Attribute	Shadow algorithm name
type	Attribute	Shadow algorithm type
props (?)	Attribute	Shadow algorithm property configuration

Mixed Rules

Configuration Item Explanation

```
<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:readwrite-splitting="http://shardingsphere.apache.org/schema/
shardingsphere/readwrite-splitting"
       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://shardingsphere.apache.org/schema/shardingsphere/
datasource
                           http://shardingsphere.apache.org/schema/shardingsphere/
datasource/datasource.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/
encrypt
                           http://shardingsphere.apache.org/schema/shardingsphere/
encrypt/encrypt.xsd
                           ">
    <bean id="write_ds0" class="  com.zaxxer.hikari.HikariDataSource" init-method=
"init" destroy-method="close">
        <property name="driverClassName" value="com.mysql.jdbc.Driver" />
        <property name="jdbcUrl" value="jdbc:mysql://localhost:3306/write_ds?
useSSL=false&amp;useUnicode=true&amp;characterEncoding=UTF-8" />
        <property name="username" value="root" />
        <property name="password" value="" />
    </bean>

    <bean id="read_ds0_0" class="  com.zaxxer.hikari.HikariDataSource" init-method=
"init" destroy-method="close">
        <!-- ...Omit specific configuration. -->
    </bean>

    <bean id="read_ds0_1" class="  com.zaxxer.hikari.HikariDataSource" init-method=
"init" destroy-method="close">
        <!-- ...Omit specific configuration. -->
    </bean>

    <bean id="write_ds1" class="  com.zaxxer.hikari.HikariDataSource" init-method=
"init" destroy-method="close">
        <!-- ...Omit specific configuration. -->
    </bean>

    <bean id="read_ds1_0" class="  com.zaxxer.hikari.HikariDataSource" init-method=
"init" destroy-method="close">
        <!-- ...Omit specific configuration. -->
    </bean>

    <bean id="read_ds1_1" class="  com.zaxxer.hikari.HikariDataSource" init-method=
"init" destroy-method="close">
        <!-- ...Omit specific configuration. -->
    </bean>

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

    <!-- readwrite-splitting rule configuration -->
    <readwrite-splitting:rule id="readWriteSplittingRule">
        <readwrite-splitting:data-source-rule id="ds_0" write-data-source-name=
"write_ds0" read-data-source-names="read_ds0_0, read_ds0_1" load-balance-algorithm-
ref="randomStrategy" />
```

```
        <readwrite-splitting:data-source-rule id="ds_1" write-data-source-name=
"write_ds1" read-data-source-names="read_ds1_0, read_ds1_1" load-balance-algorithm-
ref="randomStrategy" />
    </readwrite-splitting:rule>

    <!-- sharding strategy configuration -->
    <sharding:standard-strategy id="databaseStrategy" sharding-column="user_id"
algorithm-ref="inlineDatabaseStrategyAlgorithm" />
    <sharding:standard-strategy id="orderTableStrategy" sharding-column="order_id"
algorithm-ref="inlineOrderTableStrategyAlgorithm" />
    <sharding:standard-strategy id="orderItemTableStrategy" sharding-column="order_
item_id" algorithm-ref="inlineOrderItemTableStrategyAlgorithm" />

    <sharding:sharding-algorithm id="inlineDatabaseStrategyAlgorithm" type="INLINE
">
        <props>
            <!-- the expression enumeration is the logical data source name of the
readwrite-splitting configuration -->
            <prop key="algorithm-expression">ds_${user_id % 2}</prop>
        </props>
    </sharding:sharding-algorithm>
    <sharding:sharding-algorithm id="inlineOrderTableStrategyAlgorithm" type=
"INLINE">
        <props>
            <prop key="algorithm-expression">t_order_${order_id % 2}</prop>
        </props>
    </sharding:sharding-algorithm>
    <sharding:sharding-algorithm id="inlineOrderItemTableStrategyAlgorithm" type=
"INLINE">
        <props>
            <prop key="algorithm-expression">t_order_item_${order_item_id % 2}</
prop>
        </props>
    </sharding:sharding-algorithm>

    <!-- sharding rule configuration -->
    <sharding:rule id="shardingRule">
        <sharding:table-rules>
            <!-- the expression 'ds_${0..1}' enumeration is the logical data source
name of the readwrite-splitting configuration  -->
            <sharding:table-rule logic-table="t_order" actual-data-nodes="ds_${0..
1}.t_order_${0..1}" database-strategy-ref="databaseStrategy" table-strategy-ref=
"orderTableStrategy" key-generate-strategy-ref="orderKeyGenerator"/>
            <sharding:table-rule logic-table="t_order_item" actual-data-nodes="ds_$
{0..1}.t_order_item_${0..1}" database-strategy-ref="databaseStrategy" table-
strategy-ref="orderItemTableStrategy" 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>

    <!-- data encrypt 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">
        <encrypt:table name="t_user">
            <encrypt:column logic-column="user_name" cipher-column="user_name"
plain-column="user_name_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>

    <!-- datasource configuration -->
    <!-- the element data-source-names's value is all of the datasource name -->
    <shardingsphere:data-source id="readQueryDataSource" data-source-names="write_
ds0, read_ds0_0, read_ds0_1, write_ds1, read_ds1_0, read_ds1_1"
        rule-refs="readWriteSplittingRule, shardingRule, encryptRule" >
        <props>
            <prop key="sql-show">true</prop>
        </props>
    </shardingsphere:data-source>
</beans>
```

7.1.5 Properties Configuration

Apache ShardingSphere provides the way of property configuration to configure system level configuration.

Configuration Item Explanation

N ame	• DataType*	*Description*	• DefaultValue*
sql- show (?)	boolean	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	false
sq l-si mple (?)	boolean	Whether show SQL details in simple style	false
kern el-e xecu tor- size (?)	int	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	infinite
ma x-co nnec tion s-si ze-p er-q uery (?)	int	Max opened connection size for each query	1
chec k-ta ble- meta data -ena bled (?)	boolean	Whether validate table meta data consistency when application startup or updated	false
c heck -dup lica te-t able -ena bled (?)	boolean	Whether validate duplicate table when application startup or updated	false
s ql-c omme nt-p arse -ena bled (?)	boolean	Whether parse the comment of SQL	false
sq l-fe dera tion -ena bled (?)	boolean	Whether enable SQL federation	false

7.1.6 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

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

Sharding Algorithm

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:

Name	*Data Type*	*Description*
date time-lower	S tring	Shard datetime begin boundary, pattern: yyyy-MM-dd HH:mm:ss
date time-upper	S tring	Shard datetime end boundary, pattern: yyyy-MM-dd HH:mm:ss
shardi ng-seconds	long	Max seconds for the data in one shard

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

Type: INTERVAL

Attributes:

Name	• DataType*	*Description*	• DefaultValue*
date time-p attern	String	Timestamp pattern of sharding value, must can be transformed to Java LocalDateTime. For example: yyyy-MM-dd HH:mm:ss	•
da tetime -lower	String	Datetime sharding lower boundary, pattern is defined `datetime-pattern`	•
da tetime -upper (?)	String	Datetime sharding upper boundary, pattern is defined `datetime-pattern`	Now
shard ing-su ffix-p attern	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	•
dateti me-int erval-amount (?)	int	Interval of sharding value	1
date time-i nterva l-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.

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

Key Generate Algorithm

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

UUID

Type: UUID

Attributes: None

Load Balance Algorithm

Round Robin Algorithm

Type: ROUND_ROBIN

Attributes: None

Random Algorithm

Type: RANDOM

Attributes: None

Encryption Algorithm

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

Shadow Algorithm

Column Shadow Algorithm

Column Value Match Shadow Algorithm

Type： VALUE_MATCH

Attributes:

Name	*DataType*	*Description*
column	String	Shadow column
operation	String	SQL operation type（INSERT, UPDATE, DELETE, SELECT）
value	String	Shadow column matching value

Column Regex Match Shadow Algorithm

Type: REGEX_MATCH

Attributes:

Name	*DataType*	*Description*
column	String	Shadow column
operation	String	SQL operation type (insert, update, delete, select)
regex	String	Shadow column matching regular expression

Note Shadow Algorithm

Simple SQL Note Shadow Algorithm

Type: SIMPLE_HINT

Attributes:

Configure at least a set of arbitrary key-value pairs. For example: foo:bar

Name	*DataType*	*Description*
foo	String	bar

7.1.7 Special API

This chapter will introduce the special API of ShardingSphere-JDBC.

Sharding

This chapter will introduce the Sharding API of ShardingSphere-JDBC.

Hint

Introduction

Apache ShardingSphere uses ThreadLocal to manage sharding key value or hint route. Users can add sharding values to HintManager, and those values only take effect within the current thread.

Usage of hint:

- Sharding columns are not in SQL and table definition, but in external business logic.
- Some operations forced to do in the primary database.

Usage

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:

```yaml
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();
```

```java
     Connection conn = dataSource.getConnection();
     PreparedStatement preparedStatement = conn.prepareStatement(sql)) {
    hintManager.setDatabaseShardingValue(3);
    try (ResultSet rs = preparedStatement.executeQuery()) {
        while (rs.next()) {
            // ...
        }
    }
}
```

Primary Route with Hint

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:

```java
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()) {
            // ...
        }
    }
}
```

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

Import Maven Dependency

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

<!-- import if using XA transaction -->
<dependency>
    <groupId>org.apache.shardingsphere</groupId>
    <artifactId>shardingsphere-transaction-xa-core</artifactId>
    <version>${shardingsphere.version}</version>
</dependency>

<!-- import if using BASE transaction -->
<dependency>
    <groupId>org.apache.shardingsphere</groupId>
    <artifactId>shardingsphere-transaction-base-seata-at</artifactId>
    <version>${shardingsphere.version}</version>
</dependency>
```

Use Distributed Transaction

```
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

Import Maven Dependency

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

<!-- import if using XA transaction -->
<dependency>
    <groupId>org.apache.shardingsphere</groupId>
    <artifactId>shardingsphere-transaction-xa-core</artifactId>
    <version>${shardingsphere.version}</version>
</dependency>

<!-- import if using BASE transaction -->
<dependency>
    <groupId>org.apache.shardingsphere</groupId>
    <artifactId>shardingsphere-transaction-base-seata-at</artifactId>
    <version>${shardingsphere.version}</version>
</dependency>
```

Configure Transaction Manager

```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 Transaction

```
@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();
    });
}
```

Use Spring Namespace

Import Maven Dependency

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

<!-- import if using XA transaction -->
<dependency>
    <groupId>org.apache.shardingsphere</groupId>
    <artifactId>shardingsphere-transaction-xa-core</artifactId>
    <version>${shardingsphere.version}</version>
</dependency>

<!-- import if using BASE transaction -->
<dependency>
    <groupId>org.apache.shardingsphere</groupId>
    <artifactId>shardingsphere-transaction-base-seata-at</artifactId>
    <version>${shardingsphere.version}</version>
</dependency>
```

Configure Transaction Manager

```
<!-- ShardingDataSource configuration -->
<!-- ...  -->

<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 auto scan @ShardingSphereTransactionType annotation to inject the
transaction type before connection created -->
<sharding:tx-type-annotation-driven />
```

Use Distributed Transaction

```
@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

The default XA transaction manager of Apache ShardingSphere is Atomikos.

Data Recovery

`xa_tx.log` generated in the project `logs` folder is necessary for the recovery when XA crashes. Please keep it.

Update Configuration

Developer can add `jta.properties` in classpath of the application to customize Atomikos configuration. For detailed configuration rules.

Please refer to Atomikos official documentation for more details.

Bitronix Transaction

Import 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>

<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>
```

Customize Configuration Items

Please refer to Bitronix official documentation for more details.

Configure XA Transaction Manager Type

Yaml:

```yaml
- !TRANSACTION
  defaultType: XA
  providerType: Bitronix
```

SpringBoot:

```yaml
spring:
  shardingsphere:
    props:
      xa-transaction-manager-type: Bitronix
```

Spring Namespace:

```xml
<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>
```

Narayana Transaction

Import Maven Dependency

```xml
<properties>
    <narayana.version>5.9.1.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>

<!-- Import if using XA transaction -->
<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>
```

Customize Configuration Items

Add `jbossts-properties.xml` in classpath of the application to customize Narayana configuration.

Please refer to Narayana official documentation for more details.

Configure XA Transaction Manager 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

Startup Seata Server

Download seata server according to seata-work-shop.

Create Undo Log Table

Create undo_log table in each physical database (sample for MySQL).

```sql
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';
```

Update Configuration

Configure `seata.conf` file in classpath.

```
client {
    application.id = example    ## application unique ID
    transaction.service.group = my_test_tx_group   ## transaction group
}
```

Modify `file.conf` and `registry.conf` if needed.

Observability

Introduce how to use agent and integrate 3rd party with observability.

Use Agent

Build

Local Build

```
> cd  shardingsphere/shardingsphere-agent
> mvn clean install
```

Download（Not Released Yet）

```
> weget http://xxxxx/shardingsphere-agent.tar.gz
> tar -zxvcf shardingsphere-agent.tar.gz
```

Configuration

Found `agent.yaml` file:

```
applicationName: shardingsphere-agent
ignoredPluginNames: # A collection of ignored plugins
  - Opentracing
  - Jaeger
  - Zipkin
  - Prometheus
  - OpenTelemetry
  - Logging

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"
      JAEGER_REPORTER_LOG_SPANS: "true"
      JAEGER_REPORTER_FLUSH_INTERVAL: "1"
  Zipkin:
    host: "localhost"
    port: 9411
    props:
      SERVICE_NAME: "shardingsphere-agent"
      URL_VERSION: "/api/v2/spans"
  Opentracing:
    props:
      OPENTRACING_TRACER_CLASS_NAME: "org.apache.skywalking.apm.toolkit.
opentracing.SkywalkingTracer"
  OpenTelemetry:
    props:
      otel.resource.attributes: "service.name=shardingsphere-agent" # Multiple
configurations can be split by ','
      otel.traces.exporter: "zipkin"
  Logging:
    props:
      LEVEL: "INFO"
```

Startup

Add arguments in startup script.

```
-javaagent:\absolute path\shardingsphere-agent.jar
```

APM Integration

Usage

Use OpenTracing

- Method 1: inject Tracer provided by APM system through reading system parameters

Add startup arguments

```
-Dorg.apache.shardingsphere.tracing.opentracing.tracer.class=org.apache.skywalking.
apm.toolkit.opentracing.SkywalkingTracer
```

Call initialization method.

```
ShardingTracer.init();
```

- Method 2: inject Tracer provided by APM through parameter.

```
ShardingTracer.init(new SkywalkingTracer());
```

Notice: when using SkyWalking OpenTracing agent, you should disable the former ShardingSphere agent plug-in to avoid the conflict between them.

Use SkyWalking's Automatic Agent

Please refer to SkyWalking Manual.

Use OpenTelemetry

Just fill in the configuration in `agent.yaml`. For example, export Traces data to Zipkin.

```
OpenTelemetry:
    props:
      otel.resource.attributes: "service.name=shardingsphere-agent"
      otel.traces.exporter: "zipkin"
      otel.exporter.zipkin.endpoint: "http://127.0.0.1:9411/api/v2/spans"
```

Result Demonstration

No matter in which way, it is convenient to demonstrate APM information in the connected system. Take SkyWalking for example:

Application Architecture

Use `ShardingSphere-Proxy` to visit two databases, `192.168.0.1:3306` and `192.168.0.2:3306`, and there are two tables in each one of them.

Topology

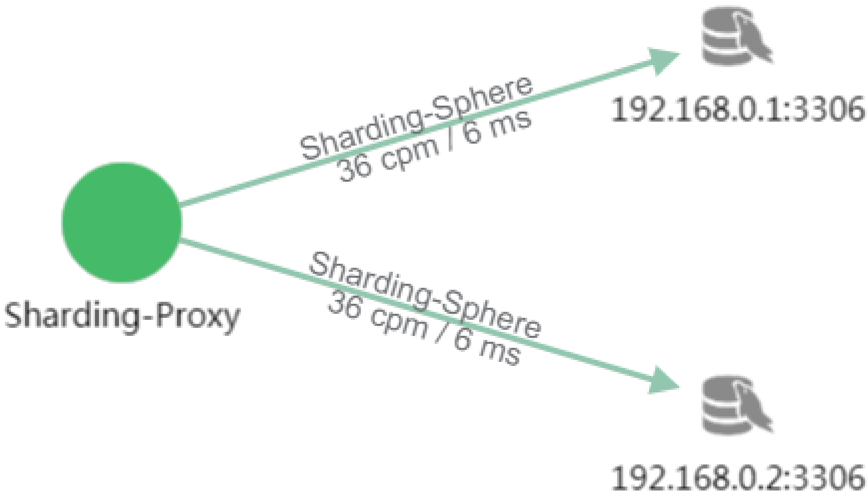

Figure1: The topology diagram

It can be seen from the picture that the user has accessed ShardingSphere-Proxy 18 times, with each database twice each time. It is because two tables in each database are accessed each time, so there are totally four tables accessed each time.

Tracking Data

Figure2: The tracking diagram

SQL parsing and implementation can be seen from the tracing diagram.

`/Sharding-Sphere/parseSQL/` indicates the SQL parsing performance this time.

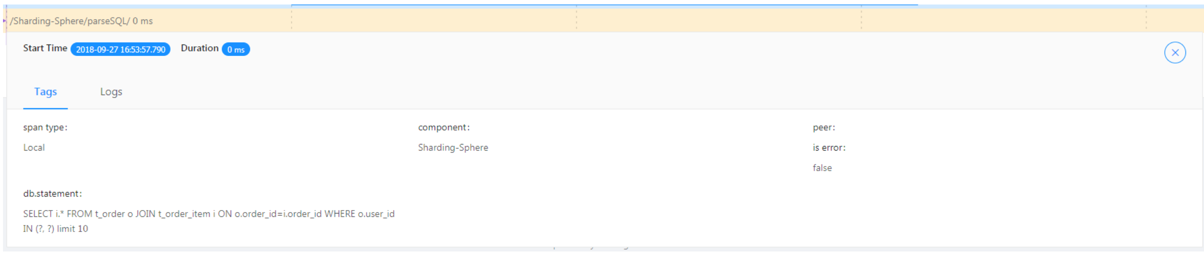

Figure3: The parsing node

`/Sharding-Sphere/executeSQL/` indicates the SQL parsing performance in actual execution.

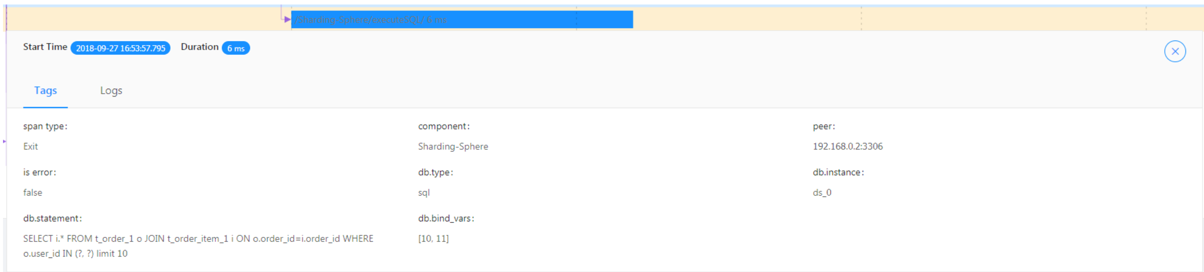

Figure4: The actual access node

Exception

Figure5: Exception tracking diagram

Exception nodes can be seen from the tracing diagram.

`/Sharding-Sphere/executeSQL/` indicates the exception results of SQL.

`/Sharding-Sphere/executeSQL/` indicates the exception log of SQL execution.

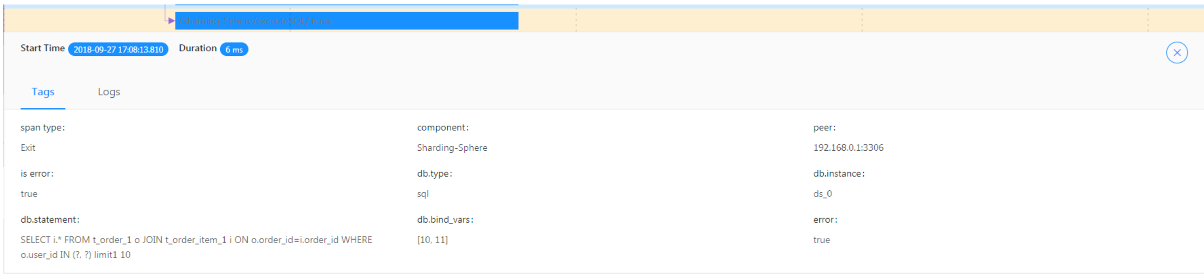

Figure6: Exception node

Figure7: Exception log

7.1.8 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.

7.2 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.

7.2.1 Startup

This chapter will introduce the deployment and startup of ShardingSphere-Proxy.

Use Binary Tar

Startup Steps

1. Download the latest version of ShardingSphere-Proxy.
2. After the decompression, revise `conf/server.yaml` and documents begin with `config-` prefix, `conf/config-xxx.yaml` for example, to configure sharding rules and readwrite-splitting rules. Please refer to Configuration Manual for the configuration method.
3. Please run `bin/start.sh` for Linux operating system; run `bin/start.bat` for Windows operating system to start ShardingSphere-Proxy. To configure start port and document location, please refer to Quick Start.

Using database protocol

Using PostgreSQL

1. Use any PostgreSQL terminal to connect, such as `psql -U root -h 127.0.0.1 -p 3307`.

Using MySQL

1. Copy MySQL' s JDBC driver to folder `ext-lib/`.
2. Use any MySQL terminal to connect, such as `mysql -u root -h 127.0.0.1 -P 3307`.

Using openGauss

1. Copy openGauss' s JDBC driver to folder `ext-lib/`.
2. Use any openGauss terminal to connect, such as `gsql -U root -h 127.0.0.1 -p 3307`.

Using metadata persist repository

Using ZooKeeper

Default integration.

Using Etcd

1. Copy Etcd' s client driver to folder `ext-lib/`.

Using Distributed Transaction

Same with ShardingSphere-JDBC. please refer to Distributed Transaction for more details.

Using user-defined algorithm

When developer need to use user-defined algorithm, should use the way below to configure algorithm, use sharding algorithm as example.

1. Implement `ShardingAlgorithm` interface.
2. Package Java file to jar.
3. Copy jar to ShardingSphere-Proxy' s `ext-lib/` folder.
4. Configure user-defined Java class into YAML file. Please refer to Configuration Manual for more details.

Notices

1. ShardingSphere-Proxy uses `3307` port in default. Users can start the script parameter as the start port number, like `bin/start.sh 3308`.
2. ShardingSphere-Proxy uses `conf/server.yaml` to configure the registry center, authentication information and public properties.
3. ShardingSphere-Proxy supports multi-logic data sources, with each yaml configuration document named by `config-` prefix as a logic data source.

Use Docker

Pull Official Docker Image

```
docker pull apache/shardingsphere-proxy
```

Build Docker Image Manually (Optional)

```
git clone https://github.com/apache/shardingsphere
mvn clean install
cd shardingsphere-distribution/shardingsphere-proxy-distribution
mvn clean package -Prelease,docker
```

Configure ShardingSphere-Proxy

Create `server.yaml` and `config-xxx.yaml` to configure sharding rules and server rule in `/${your_work_dir}/conf/`. Please refer to Configuration Manual. Please refer to Example.

Run Docker

```
docker run -d -v /${your_work_dir}/conf:/opt/shardingsphere-proxy/conf -e PORT=3308
-p13308:3308 apache/shardingsphere-proxy:latest
```

Notice

- You can define port `3308` and `13308` by yourself. `3308` refers to docker port; `13308` refers to the host port.
- You have to volume conf dir to `/opt/shardingsphere-proxy/conf`.

```
docker run -d -v /${your_work_dir}/conf:/opt/shardingsphere-proxy/conf -e JVM_OPTS=
"-Djava.awt.headless=true" -e PORT=3308 -p13308:3308 apache/shardingsphere-
proxy:latest
```

Notice

- You can define JVM related parameters to environment variable `JVM_OPTS`.

```
docker run -d -v /${your_work_dir}/conf:/opt/shardingsphere-proxy/conf -v /${your_
work_dir}/ext-lib:/opt/shardingsphere-proxy/ext-lib -p13308:3308 apache/
shardingsphere-proxy:latest
```

Notice

- If you want to import external jar packages, whose directory is supposed to volume to /opt/shardingsphere-proxy/ext-lib.

Access ShardingSphere-Proxy

It is in the same way as connecting to PostgreSQL.

```
psql -U ${your_user_name} -h ${your_host} -p 13308
```

FAQ

Question 1: there is I/O exception (`java.io.IOException`) when process request to `{}->unix://localhost:80: Connection` is refused.

Answer: before building image, please make sure docker daemon thread is running.

Question 2: there is error report of being unable to connect to the database.

Answer: please make sure the designated PostgreSQL’ s IP in `/${your_work_dir}/conf/config-xxx.yaml` configuration is accessible to Docker container.

Question 3： How to start ShardingSphere-Proxy whose backend databases are MySQL or openGauss.

Answer： Mount the directory where `mysql-connector.jar` or `opengauss-jdbc.jar` stores to `/opt/shardingsphere-proxy/ext-lib`.

Question 4： How to import user-defined sharding strategy?

Answer: Volume the directory where `shardingsphere-strategy.jar` stores to `/opt/shardingsphere-proxy/ext-lib`.

7.2.2 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.

Authority

It is used to set up initial user to login compute node, and authority data of storage node.

Configuration Item Explanation

```
rules:
  - !AUTHORITY
    users:
      - # Username, authorized host and password for compute node. Format:
<username>@<hostname>:<password>, hostname is % or empty string means do not care
about authorized host
    provider:
      type: # authority provider for storage node
```

Example

```yaml
rules:
  - !AUTHORITY
    users:
      - root@localhost:root
      - my_user@pwd
    provider:
      type: FOO_AUTHORITY_PROVIDER
```

Refer to Authority Provider for more implementations.

7.2.3 DistSQL

This chapter will introduce the detailed syntax of DistSQL.

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

Syntax

```sql
ADD RESOURCE dataSource [, dataSource] ...

ALTER RESOURCE dataSource [, dataSource] ...

DROP RESOURCE dataSourceName [, dataSourceName] ... [ignore single tables]

dataSource:
    simpleSource | urlSource

simpleSource:
    dataSourceName(HOST=hostName,PORT=port,DB=dbName,USER=user [,PASSWORD=password]
[,PROPERTIES(poolProperty [,poolProperty] ...)])
```

```
urlSource:
    dataSourceName(URL=url,USER=user [,PASSWORD=password] [,PROPERTIES(poolProperty
[,poolProperty]) ...])

poolProperty:
    "key"= ("value" | value)
```

- 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 added resource can be connected normally, otherwise it will not be added successfully
- Duplicate `dataSourceName` is not allowed to be added
- In the definition of a `dataSource`, the syntax of `simpleSource` and `urlSource` cannot be mixed
- `poolProperty` is used to customize connection pool properties, `key` must be the same as the connection pool property name, `value` supports int and String types
- `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] ...

CREATE DEFAULT SHARDING shardingScope STRATEGY (shardingStrategy)

ALTER SHARDING TABLE RULE shardingTableRuleDefinition [,
shardingTableRuleDefinition] ...

DROP SHARDING TABLE RULE tableName [, tableName] ...

CREATE SHARDING ALGORITHM shardingAlgorithmDefinition [,
shardingAlgorithmDefinition] ...
```

```
ALTER SHARDING ALGORITHM shardingAlgorithmDefinition [,
shardingAlgorithmDefinition] ...

DROP SHARDING ALGORITHM algorithmName [, algorithmName] ...

shardingTableRuleDefinition:
    shardingAutoTableRule | shardingTableRule

shardingAutoTableRule:
    tableName(resources (COMMA shardingColumn)? (COMMA algorithmDefinition)? (COMMA
keyGenerateStrategy)?)

shardingTableRule:
    tableName(dataNodes (COMMA  databaseStrategy)? (COMMA tableStrategy)? (COMMA
keyGenerateStrategy)?)

resources:
    RESOURCES(resource [, resource] ...)

dataNodes:
    DATANODES(dataNode [, dataNode] ...)

resource:
    resourceName | inlineExpression

dataNode:
    resourceName | inlineExpression

shardingColumn:
    SHARDING_COLUMN=columnName

algorithmDefinition:
    TYPE(NAME=shardingAlgorithmType [, PROPERTIES([algorithmProperties])])

keyGenerateStrategy:
    GENERATED_KEY(COLUMN=columnName, strategyDefinition)

shardingScope:
    DATABASE | TABLE

databaseStrategy:
    DATABASE_STRATEGY(shardingStrategy)

tableStrategy:
    TABLE_STRATEGY(shardingStrategy)

shardingStrategy:
```

```
    TYPE=strategyType, shardingColumn, shardingAlgorithm

shardingColumn:
    SHARDING_COLUMN=columnName

shardingAlgorithm:
    SHARDING_ALGORITHM=shardingAlgorithmName

strategyDefinition:
    TYPE(NAME=keyGenerateStrategyType [, PROPERTIES([algorithmProperties])])

shardingAlgorithmDefinition:
    shardingAlgorithmName(algorithmDefinition)

algorithmProperties:
    algorithmProperty [, algorithmProperty] ...

algorithmProperty:
    key=value
```

- `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 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

```
CREATE SHARDING TABLE RULE t_order (
RESOURCES(resource_0,resource_1),
SHARDING_COLUMN=order_id,
TYPE(NAME=hash_mod,PROPERTIES("sharding-count"=4)),
GENERATED_KEY(COLUMN=another_id,TYPE(NAME=snowflake,PROPERTIES("worker-id"=123)))
),t_order_item (
DATANODES("resource_${0..1}.t_order_item_${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),
GENERATED_KEY(COLUMN=another_id,TYPE(NAME=snowflake,PROPERTIES("worker-id"=123)))
);

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)),
GENERATED_KEY(COLUMN=another_id,TYPE(NAME=snowflake,PROPERTIES("worker-id"=123)))
),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),
GENERATED_KEY(COLUMN=another_id,TYPE(NAME=uuid,PROPERTIES("worker-id"=123)))
);

DROP SHARDING TABLE RULE t_order, t_order_item;
```

```
CREATE DEFAULT SHARDING DATABASE STRATEGY (
TYPE = standard,SHARDING_COLUMN=order_id,SHARDING_ALGORITHM=algorithmsName
);

CREATE SHARDING ALGORITHM database_inline (
TYPE(NAME=inline,PROPERTIES("algorithm-expression"="resource_${user_id % 2}"))
),table_inline (
TYPE(NAME=inline,PROPERTIES("algorithm-expression"="resource_${order_id % 2}"))
);

ALTER SHARDING ALGORITHM database_inline (
TYPE(NAME=inline,PROPERTIES("algorithm-expression"="resource_${user_id % 4}"))
),table_inline (
TYPE(NAME=inline,PROPERTIES("algorithm-expression"="resource_${order_id % 4}"))
);

DROP SHARDING ALGORITHM t_order_hash_mod;
```

Sharding Binding Table Rule

```
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

```
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;
```

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

loadBanlancerDefinition:
    TYPE(NAME=loadBanlancerType [, PROPERTIES([algorithmProperties] )] )

algorithmProperties:
    algorithmProperty [, algorithmProperty] ...

algorithmProperty:
    key=value
```

- 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,
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 databaseDiscoveryRuleDefinition [,
databaseDiscoveryRuleDefinition] ...

ALTER DB_DISCOVERY RULE databaseDiscoveryRuleDefinition [,
databaseDiscoveryRuleDefinition] ...

DROP DB_DISCOVERY RULE ruleName [, ruleName] ...

databaseDiscoveryRuleDefinition:
    ruleName(resources, discoveryTypeDefinition)

resources:
    RESOURCES(resourceName [, resourceName] ...)

discoveryTypeDefinition:
    TYPE(NAME=discoveryType [, PROPERTIES([algorithmProperties] )] )

algorithmProperties:
    algorithmProperty [, algorithmProperty] ...
```

```
algorithmProperty:
    key=value
```

- `discoveryType` specifies the database discovery service type, `ShardingSphere` has built-in support for `MGR`
- Duplicate `ruleName` will not be created

Example

```sql
CREATE DB_DISCOVERY RULE ha_group_0 (
RESOURCES(resource_0,resource_1),
TYPE(NAME=mgr,PROPERTIES(groupName='92504d5b-6dec',keepAliveCron=''))
);

ALTER DB_DISCOVERY RULE ha_group_0 (
RESOURCES(resource_0,resource_1,resource_2),
TYPE(NAME=mgr,PROPERTIES(groupName='92504d5b-6dec' ,keepAliveCron=''))
);

DROP DB_DISCOVERY RULE ha_group_0;
```

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
```

- 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

```
-  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](https://shardingsphere.apache.org/document/current/en/user-manual/
shardingsphere-proxy/usage/distsql/syntax/rdl/resource-definition/)
- `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

```sql
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_note_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 SCHEMA RESOURCES [FROM schemaName]
```

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 parameter

Example

```
mysql> show schema resources;
+------+-------+-----------+------+------+--------------------------------------
--------------------------------------------------------------------------------
-------------------------------------------------------------------------+
| name | type  | host      | port | db   | attribute

                                                                      |
+------+-------+-----------+------+------+--------------------------------------
--------------------------------------------------------------------------------
-------------------------------------------------------------------------+
| ds_0 | MySQL | 127.0.0.1 | 3306 | ds_0 | {"minPoolSize":1,
"connectionTimeoutMilliseconds":30000,"maxLifetimeMilliseconds":1800000,"readOnly
":false,"idleTimeoutMilliseconds":60000,"maxPoolSize":50} |
| ds_1 | MySQL | 127.0.0.1 | 3306 | ds_1 | {"minPoolSize":1,
"connectionTimeoutMilliseconds":30000,"maxLifetimeMilliseconds":1800000,"readOnly
":false,"idleTimeoutMilliseconds":60000,"maxPoolSize":50} |
+------+-------+-----------+------+------+--------------------------------------
--------------------------------------------------------------------------------
-------------------------------------------------------------------------+
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 schemaName]

SHOW SHARDING ALGORITHMS [FROM schemaName]

tableRule:
    RULE tableName
```

- Support query all data fragmentation rules and specified table query
- Support query all sharding algorithms

Sharding Binding Table Rule

```
SHOW SHARDING BINDING TABLE RULES [FROM schemaName]
```

Sharding Broadcast Table Rule

```
SHOW SHARDING BROADCAST TABLE RULES [FROM schemaName]
```

Return Value Description

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 parameter
table_strategy_type	Table sharding strategy type
table_sharding_column	Table sharding column
table_ sharding_algorithm_type	Database sharding algorithm type
table_s harding_algorithm_props	Database sharding algorithm parameter
key_generate_column	Distributed primary key generation column
key_generator_type	Distributed primary key generation type
key_generator_props	Distributed primary key generation parameter

Sharding Algorithms

Column	Description
name	Sharding algorithm name
type	Sharding algorithm type
props	Sharding algorithm parameters

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

Example

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        | worker-id:123      |
| 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        | worker-id:123      |
| 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
   | worker-id:123     |
+---------+----------------------------+--------------------+---------------------
+-----------------------+-------------------------------+-------------------------
---------------+--------------------+---------------------+-------------------------
---+-----------------------------------------------+-------------------+------------
------+-------------------+
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)
```

Sharding Binding Table Rule

```
mysql> show sharding binding table rules from sharding_db;
+----------------------+
| 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 (tableRule | RULES) [FROM schemaName]

tableRule:
    RULE tableName
```

Return Value Description

Column	Description
table_name	Single table name
resource_name	Data source name

Example

```
mysql> show single table rules;
+--------------+---------------+
| table_name   | resource_name |
+--------------+---------------+
| t_single_0   | ds_0          |
| t_single_1   | ds_1          |
+--------------+---------------+
2 rows in set (0.02 sec)
```

Readwrite-Splitting

Syntax

```
SHOW READWRITE_SPLITTING RULES [FROM schemaName]
```

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）
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 | NULL                        | 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_name | read_data_source_
names | load_balancer_type | load_balancer_props |
+--------+-----------------------------+-----------------------+-------------------
-----+--------------------+---------------------+
| pr_ds | ms_group_0                  | NULL                  |
    | 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_name | read_data_source_
names | load_balancer_type | load_balancer_props |
+--------+-----------------------------+-----------------------+-------------------
-----+--------------------+---------------------+
| pr_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 schemaName]
```

Return Value Description

Column	Description
name	Rule name
data_source_names	Data source name list
primary_data_source_name	Primary data source name
discover_type	Database discovery service type
discover_props	Database discovery service parameters

Example

```
mysql> show db_discovery rules from database_discovery_db;
+-------+--------------------+-------------------------+---------------+---------
-------------------------------------------------------------------------------------
----------------+
| name  | data_source_names   | primary_data_source_name | discover_type |
discover_props
                        |
+-------+--------------------+-------------------------+---------------+---------
-------------------------------------------------------------------------------------
----------------+
| pr_ds | ds_0, ds_1, ds_2    | ds_0                     | MGR           |
keepAliveCron=0/50 * * * * ?, zkServerLists=localhost:2181, groupName=b13df29e-
90b6-11e8-8d1b-525400fc3996 |
+-------+--------------------+-------------------------+---------------+---------
-------------------------------------------------------------------------------------
----------------+
1 row in set (0.00 sec)
```

Encrypt

Syntax

```
SHOW ENCRYPT RULES [FROM schemaName]

SHOW ENCRYPT TABLE RULE tableName [from schemaName]
```

- 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
cipher_column	Ciphertext column name
plain_column	Plaintext column name
encryptor_type	Encryption algorithm type
encryptor_props	Encryption algorithm parameter

Example

Show Encrypt Rules

```
mysql> show encrypt rules from encrypt_db;
+-----------+--------------+---------------+--------------+----------------+-------
------------------+
| table     | logic_column | cipher_column | plain_column | encryptor_type |
encryptor_props         |
+-----------+--------------+---------------+--------------+----------------+-------
------------------+
| t_encrypt | order_id     | order_cipher  | NULL         | MD5            |
                 |
| t_encrypt | user_id      | user_cipher   | user_plain   | AES            | aes-
key-value=123456abc |
| t_order   | item_id      | order_cipher  | NULL         | MD5            |
                 |
| t_order   | order_id     | user_cipher   | user_plain   | AES            | aes-
key-value=123456abc |
+-----------+--------------+---------------+--------------+----------------+-------
------------------+
4 rows in set (0.01 sec)
```

Show Encrypt Table Rule Table Name

```
mysql> show encrypt table rule t_encrypt;
+-----------+--------------+---------------+--------------+----------------+-------
------------------+
| table     | logic_column | cipher_column | plain_column | encryptor_type |
encryptor_props         |
+-----------+--------------+---------------+--------------+----------------+-------
------------------+
| t_encrypt | order_id     | order_cipher  | NULL         | MD5            |
                 |
| t_encrypt | user_id      | user_cipher   | user_plain   | AES            | aes-
key-value=123456abc |
```

```
+-----------+--------------+---------------+--------------+----------------+-------
-------------------+
2 rows in set (0.00 sec)

mysql> show encrypt table rule t_encrypt from encrypt_db;
+-----------+--------------+---------------+--------------+----------------+-------
-------------------+
| table     | logic_column | cipher_column | plain_column | encryptor_type |
encryptor_props          |
+-----------+--------------+---------------+--------------+----------------+-------
-------------------+
| t_encrypt | order_id     | order_cipher  | NULL         | MD5            |
                 |
| t_encrypt | user_id      | user_cipher   | user_plain   | AES            | aes-
key-value=123456abc |
+-----------+--------------+---------------+--------------+----------------+-------
-------------------+
2 rows in set (0.00 sec)
```

Shadow

Syntax

```
SHOW SHADOW shadowRule | RULES [FROM schemaName]

SHOW SHADOW TABLE RULES [FROM schemaName]

SHOW SHADOW ALGORITHMS [FROM schemaName]

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
properties	Shadow algorithm parameters

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_note_algorithm_1
            |
+--------------+-------------------------------------------------------------------
-------------+
1 rows in set (0.01 sec)
```

SHOW SHADOW ALGORITHMS

```
mysql> show shadow algorithms;
+-------------------------+--------------------+-----------------------------------
--------+
| shadow_algorithm_name   | type               | properties
       |
+-------------------------+--------------------+-----------------------------------
--------+
| user_id_match_algorithm | COLUMN_REGEX_MATCH | operation=insert,column=user_id,
regex=[1] |
| simple_note_algorithm_1 | SIMPLE_NOTE        | shadow=true,foo=bar
       |
+-------------------------+--------------------+-----------------------------------
--------+
2 rows in set (0.01 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.

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 readwr ite_splitting hint 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 scaling list	Query running list	show scaling list
show scaling status xx	Query scaling status, xx: jobId	show scaling status 1234
start scaling xx	Start scaling, xx: jobId	start scaling 1234
stop scaling xx	Stop scaling, xx: jobId	stop scaling 1234
drop scaling xx	Drop scaling, xx: jobId	drop scaling 1234
reset scaling xx	reset progress, xx: jobId	reset scaling 1234
check scaling xx	Data consistency check with algorithm in `server.yaml`, xx: jobId	check scaling 1234
show scaling check algorithms	Show available consistency check algorithms	show scaling check algorithms
check scaling {jobId} by type(name={algorithmType})	Data consistency check with defined algorithm	check scaling 1234 by type(name=DEFAULT)
stop scaling source writing xx	The source ShardingSphere data source is discontinued, xx: jobId	stop scaling source writing 1234
checkout scaling xx	Switch to target ShardingSphere data source, xx: jobId	checkout scaling 1234

Circuit Breaker

Statement	Function	Example
[enable / disable] readwrite_splitting read xxx [from schema]	Enable or disable read data source	enable readwrite_splitting read resource_0
[enable / disable] instance IP=xxx, PORT=xxx	Enable or disable proxy instance	disable instance IP=127.0.0.1, PORT=3307
show instance list	Query proxy instance information	show instance list
show readwrite_splitting read resources [from schema]	Query all read resources status	show readwrite_splitting read resources

Other

Statement	Function	Example
set variable p roxy_property_name = xx	proxy_property_name is one of properties con- fig uration of proxy, name is split by underscore	set variable sql_show = true
set variable transaction_type = xx	Modify transaction_type of the current connection, supports LOCAL, XA, BASE	set variable tr ansaction_type = XA
set variable age nt_plugins_enabled = [true / false]	Set whether the agent plugins are enabled, the default value is false	set variable agent_p lugins_enabled = true
show all variables	Query proxy all properties configuration	show all variable
show variable p roxy_property_name	Query proxy properties configuration, name is split by underscore	show variable sql_show
show variable transaction_type	Query the transaction type of the current connection	show variable tr ansaction_type
show variable cached_connections	Query the number of cached physical database connections in the current connection	show variable cach ed_connections
show variable age nt_plugins_enabled	Query whether the agent plugin are enabled	show variable agent_p lugins_enabled
preview SQL	Preview the actual SQLs	preview select * from t_order

Notice

ShardingSphere-Proxy does not support hint by default, to support it, set `proxy-hint-enabled` to true in `conf/server.yaml`.

Usage

This chapter will introduce how to use DistSQL to manage sharding, readwrite-splitting and other rules in a distributed database.

Sharding

Usage

Pre-work

1. Start the MySQL service
2. Create MySQL database (refer to ShardingSphere-Proxy data source configuration rules)

3. Create a role or user with creation permission for ShardingSphere-Proxy
4. Start Zookeeper service (for persistent configuration)

Start ShardingSphere-Proxy

1. Add `mode` and `authentication` configurations to `server.yaml` (please refer to the example of ShardingSphere-Proxy)
2. Start ShardingSphere-Proxy (Related introduction)

Create a distributed database and sharding tables

1. Connect to ShardingSphere-Proxy
2. Create a distributed database

```
CREATE DATABASE sharding_db;
```

3. Use newly created database

```
USE sharding_db;
```

4. Configure data source information

```
ADD RESOURCE ds_0 (
HOST=127.0.0.1,
PORT=3306,
DB=ds_1,
USER=root,
PASSWORD=root
);

ADD RESOURCE ds_1 (
HOST=127.0.0.1,
PORT=3306,
DB=ds_2,
USER=root,
PASSWORD=root
);
```

5. 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)),
GENERATED_KEY(COLUMN=order_id,TYPE(NAME=snowflake,PROPERTIES("worker-id"=123)))
);
```

6. 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
```

7. Drop sharding table

```
DROP TABLE t_order;
```

8. Drop sharding rule

```
DROP SHARDING TABLE RULE t_order;
```

9. Drop resource

```
DROP RESOURCE ds_0, ds_1;
```

10. Drop distributed database

```
DROP DATABASE sharding_db;
```

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.
4. The `Auto Sharding Algorithm` will continue to increase to cover the user's various sharding scenarios.

readwrite_splitting

Usage

Pre-work

1. Start the MySQL service
2. Create MySQL database (refer to ShardingProxy data source configuration rules)
3. Create a role or user with creation permission for ShardingProxy

4. Start Zookeeper service (for persistent configuration)

Start ShardingProxy

1. Add mode and `authentication` configurations to `server.yaml` (please refer to the example of ShardingProxy)
2. Start ShardingProxy (Related introduction)

Create a distributed database and sharding tables

1. Connect to ShardingProxy
2. Create a distributed database

```
CREATE DATABASE readwrite_splitting_db;
```

3. Use newly created database

```
USE readwrite_splitting_db;
```

4. Configure data source information

```
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
);
```

5. Create readwrite_splitting rule

```
CREATE READWRITE_SPLITTING RULE group_0 (
WRITE_RESOURCE=write_ds,
READ_RESOURCES(read_ds),
TYPE(NAME=random)
);
```

6. Alter readwrite_splitting rule

```
ALTER READWRITE_SPLITTING RULE group_0 (
WRITE_RESOURCE=write_ds,
```

```
READ_RESOURCES(read_ds),
TYPE(NAME=random,PROPERTIES(read_weight='2:0'))
)
```

7. Drop readwrite_splitting rule

```
DROP READWRITE_SPLITTING RULE group_0;
```

8. Drop resource

```
DROP RESOURCE write_ds,read_ds;
```

9. Drop distributed database

```
DROP DATABASE readwrite_splitting_db;
```

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 .

Encrypt

Usage

Pre-work

1. Start the MySQL service
2. Create MySQL database (refer to ShardingProxy data source configuration rules)
3. Create a role or user with creation permission for ShardingProxy
4. Start Zookeeper service (for persistent configuration)

Start ShardingProxy

1. Add mode and `authentication` configurations to `server.yaml` (please refer to the example of ShardingProxy)
2. Start ShardingProxy (Related introduction)

Create a distributed database and sharding tables

1. Connect to ShardingProxy
2. Create a distributed database

```
CREATE DATABASE encrypt_db;
```

3. Use newly created database

```
USE encrypt_db;
```

4. Configure data source information

```
ADD RESOURCE ds_0 (
HOST=127.0.0.1,
PORT=3306,
DB=ds_0,
USER=root,
PASSWORD=root
);
```

5. Create encrypt table

```
CREATE TABLE `t_encrypt` (
  `order_id` int NOT NULL,
  `user_plain` varchar(45) DEFAULT NULL,
  `user_cipher` varchar(45) DEFAULT NULL,
  PRIMARY KEY (`order_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4
```

6. Create 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))
));
```

7. Alter 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'))),
));
```

8. Drop encrypt rule

```sql
DROP ENCRYPT RULE t_encrypt;
```

9. Drop resource

```sql
DROP RESOURCE ds_0;
```

10. Drop distributed database

```sql
DROP DATABASE encrypt_db;
```

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.

DB Discovery

Usage

Pre-work

1. Start the MySQL service
2. Create MySQL database (refer to ShardingProxy data source configuration rules)
3. Create a role or user with creation permission for ShardingProxy
4. Start Zookeeper service (for persistent configuration)

Start ShardingProxy

1. Add mode and authentication configurations to `server.yaml` (please refer to the example of ShardingProxy)
2. Start ShardingProxy (Related introduction)

Create a distributed database and sharding tables

1. Connect to ShardingProxy
2. Create a distributed database

```
CREATE DATABASE discovery_db;
```

3. Use newly created database

```
USE discovery_db;
```

4. Configure data source information

```
ADD RESOURCE ds_0 (
HOST=127.0.0.1,
PORT=3306,
DB=ds_0,
USER=root,
PASSWORD=root
),RESOURCE ds_1 (
HOST=127.0.0.1,
PORT=3306,
DB=ds_1,
USER=root,
PASSWORD=root
),RESOURCE ds_2 (
HOST=127.0.0.1,
PORT=3306,
DB=ds_2,
USER=root,
PASSWORD=root
);
```

5. Create DB discovery rule

```
CREATE DB_DISCOVERY RULE group_0 (
RESOURCES(ds_0,ds_1),
TYPE(NAME=mgr,PROPERTIES(groupName='92504d5b-6dec',keepAliveCron=''))
);
```

6. Alter DB discovery rule

```
ALTER DB_DISCOVERY RULE group_0 (
RESOURCES(ds_0,ds_1,ds_2),
TYPE(NAME=mgr,PROPERTIES(groupName='92504d5b-6dec' ,keepAliveCron=''))
);
```

7. Drop db_discovery rule

```
DROP DB_DISCOVERY RULE group_0;
```

8. Drop resource

```
DROP RESOURCE ds_0,ds_1,ds_2;
```

9. Drop distributed database

```
DROP DATABASE discovery_db;
```

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 .

Shadow

Usage

Pre-work

1. Start the MySQL service
2. Create MySQL database (refer to ShardingProxy data source configuration rules)
3. Create a role or user with creation permission for ShardingProxy
4. Start Zookeeper service (for persistent configuration)

Start ShardingProxy

1. Add mode and `authentication` configurations to `server.yaml` (please refer to the example of ShardingProxy)
2. Start ShardingProxy (Related introduction)

Create a distributed database and sharding tables

1. Connect to ShardingProxy
2. Create a distributed database

```
CREATE DATABASE shadow_db;
```

3. Use newly created database

```
USE shadow_db;
```

4. Configure data source information

```
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
);
```

5. Create shadow rule

```
CREATE SHADOW RULE group_0(
SOURCE=ds_0,
SHADOW=ds_1,
t_order((simple_note_algorithm, TYPE(NAME=SIMPLE_NOTE, PROPERTIES("shadow"="true",
foo="bar"))),(TYPE(NAME=COLUMN_REGEX_MATCH, PROPERTIES("operation"="insert","column
"="user_id", "regex"='[1]')))),
```

```
t_order_item((TYPE(NAME=SIMPLE_NOTE, PROPERTIES("shadow"="true", "foo"="bar")))));
```

6. Alter shadow rule

```
ALTER SHADOW RULE group_0(
SOURCE=ds_0,
SHADOW=ds_2,
t_order_item((TYPE(NAME=SIMPLE_NOTE, PROPERTIES("shadow"="true", "foo"="bar")))));
```

7. Drop shadow rule

```
DROP SHADOW RULE group_0;
```

8. Drop resource

```
DROP RESOURCE ds_0,ds_1,ds_2;
```

9. Drop distributed database

```
DROP DATABASE shadow_db;
```

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 .

7.2.4 Properties

Introduction

Apache ShardingSphere provides the way of property configuration to configure system level configuration.

Configuration Item Explanation

Na me	• DataType*	*Description*	• DefaultValue*	• DynamicUpdate*
sq l-s how (?)	boolean	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.	false	true
s ql- sim ple (?)	boolean	Whether show SQL details in simple style.	false	true
ke rne l-e xec uto r-s ize (?)	int	The max thread size of worker group to execute SQL. One ShardingSphere-DataSource will use a independent thread pool, it does not share thread pool even different data source in same JVM.	infinite	false
max -co nne cti ons -si ze- per -qu ery (?)	int	Max opened connection size for each query.	1	true
c hec k-t abl e-m eta dat a-e nab led (?)	boolean	Whether validate table meta data consistency when application startup or updated.	false	false
proxy- fro nte nd- flu sh- thr esh old (?)	int	Flush threshold for every records from databases for	128	true

Properties can be updated by DistSQL. Dynamic update can take effect immediately, static update can take effect after restarted.

7.3 ShardingSphere-Sidecar

7.3.1 Introduction

ShardingSphere-Sidecar (TODO) defines itself as a cloud native database agent of the Kubernetes environment, in charge of all the access to the database in the form of sidecar.

It provides a mesh layer interacting with the database, we call this as `Database Mesh`.

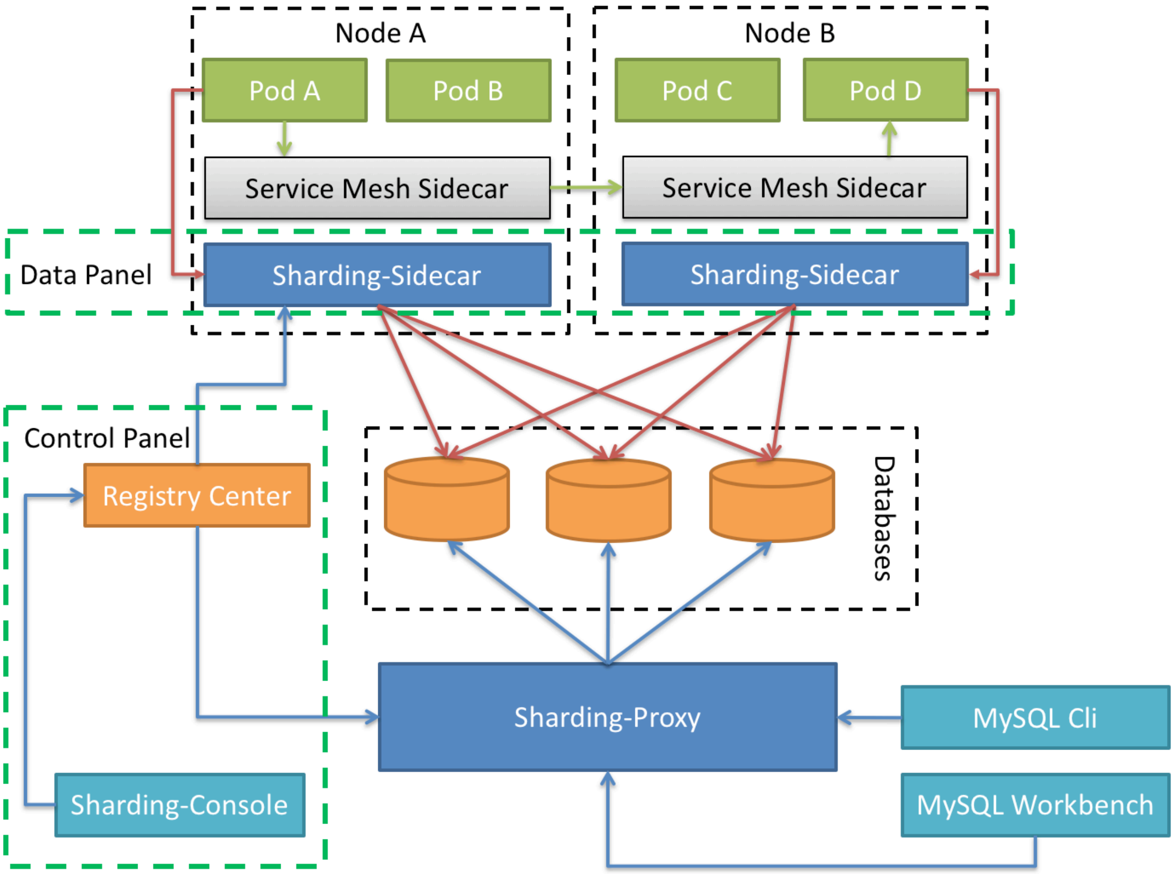

Figure8: ShardingSphere-Sidecar Architecture

7.3.2 Comparison

	Shardi ngSphere-JDBC	*Shardin gSphere-Proxy*	*ShardingS phere-Sidecar*
Database	Any	My SQL/PostgreSQL	MySQL
Connections Count Cost	High	Low	High
Supported Languages	Java Only	Any	Any
Performance	Low loss	Relatively High loss	Low loss
De centralization	Yes	No	Yes
Static Entry	No	Yes	No

The advantage of ShardingSphere-Sidecar lies in its cloud native support for Kubernetes and Mesos.

7.4 ShardingSphere-Scaling

7.4.1 Introduction

ShardingSphere-Scaling is a common solution for migrating data to ShardingSphere or scaling data in Apache ShardingSphere since **4.1.0**, current state is **Experimental** version.

7.4.2 Build

Build&Deployment

1. Execute the following command to compile and generate 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
```

The binary packages: - /shardingsphere-distribution/shardingsphere-proxy-distribution/target/apache-shardingsphere-${latest.release.version}-shardingsphere-proxy-bin.tar.gz

Or get binary package from download page.

2. Unzip the proxy distribution package, modify the configuration file `conf/server.yaml`, enable `scaling` and mode:

```
scaling:
  blockQueueSize: 10000
  workerThread: 40
  clusterAutoSwitchAlgorithm:
    type: IDLE
    props:
```

```
      incremental-task-idle-minute-threshold: 30
  dataConsistencyCheckAlgorithm:
    type: DEFAULT

mode:
  type: Cluster
  repository:
    type: ZooKeeper
    props:
      namespace: governance_ds
      server-lists: localhost:2181
      retryIntervalMilliseconds: 500
      timeToLiveSeconds: 60
      maxRetries: 3
      operationTimeoutMilliseconds: 500
  overwrite: false
```

Enable `clusterAutoSwitchAlgorithm` indicate system will detect when scaling job is finished and switch cluster configuration automatically. Currently, system supply `IDLE` type implementation.

Enable `dataConsistencyCheckAlgorithm` indicate system will use this defined algorithm to do data consistency check when it' s emitted, if it' s disabled, then data consistency check will be ignored. Currently, system supply `DEFAULT` type implementation, it supports following database types: `MySQL`, you could not enable it if you' re running other database types for now, support of other database types is under development.

You could customize an auto switch algorithm by implementing `ScalingClusterAutoSwitchAlgorithm` SPI interface, and customize a check algorithm by implementing `ScalingDataConsistencyCheckAlgorithm` SPI interface. Please refer to Dev Manual#Scaling for more details.

3. Start up ShardingSphere-Proxy:

```
sh bin/start.sh
```

4. See the proxy log file `logs/stdout.log`, ensure startup successfully.

Shutdown

```
sh bin/stop.sh
```

Configuration

The existing configuration items are as follows, we can modify them in `conf/server.yaml`:

Name	Description	Default value
blockQueueSize	Queue size of data transmission channel	10000
workerThread	Worker thread pool size, the number of migration task threads allowed to run concurrently	40

7.4.3 Manual

Manual

Environment

JAVA, JDK 1.8+.

The migration scene we support:

Source	Target
MySQL(5.1.15 ~ 5.7.x)	MySQL
PostgreSQL(9.4 ~)	PostgreSQL
openGauss(2.1.0)	openGauss

Attention:

If the backend database is in following table, please download JDBC driver jar and put it into `${shardingsphere-proxy}/lib` directory.

RDBMS	JDBC driver	Reference
MySQL	**`mysql-connector-java-5.1.47.jar <https://repo1.maven.org/maven2/mysql/mysql-connector-java/5.1.47/mysql-connector-java-5.1.47.jar>`__**	Connector/J Versions
openGauss	opengauss-jdbc-2.0.1-compatibility.jar	

Supported features:

Feature	MySQL	PostgreSQL	openGauss
Inventory migration	Supported	Supported	Supported
Incremental migration	Supported	Supported	Supported
Create table automatically	Supported	Unsupported	Supported
Default data consistency check algorithm	Supported	Unsupported	Unsupported

Attention:

For RDBMS which `Create table automatically` feature is not supported, we need to create sharding tables manually.

Privileges

We need to enable `binlog` for MySQL. Privileges of users scaling used should include Replication privileges.

```
+-----------------------------------------+---------------------------------------+
| Variable_name                           | Value                                 |
+-----------------------------------------+---------------------------------------+
| log_bin                                 | ON                                    |
| binlog_format                           | ROW                                   |
| binlog_row_image                        | FULL                                  |
+-----------------------------------------+---------------------------------------+

+------------------------------------------------------------------------------+
|Grants for ${username}@${host}                                                |
+------------------------------------------------------------------------------+
|GRANT REPLICATION SLAVE, REPLICATION CLIENT ON *.* TO ${username}@${host}     |
|.......                                                                       |
+------------------------------------------------------------------------------+
```

PostgreSQL need to support and open test_decoding feature.

DistSQL API

ShardingSphere-Scaling provides DistSQL API

Preview current sharding rule

Example:

```sql
preview select count(1) from t_order;
```

Response:

```
mysql> preview select count(1) from t_order;
+------------------+-------------------------------+
| data_source_name | sql                           |
+------------------+-------------------------------+
| ds_0             | select count(1) from t_order_0 |
| ds_0             | select count(1) from t_order_1 |
| ds_1             | select count(1) from t_order_0 |
| ds_1             | select count(1) from t_order_1 |
+------------------+-------------------------------+
4 rows in set (0.00 sec)
```

Start scaling job

1. Add new data source resources

Please refer to RDL#Data Source for more details.

Create database on underlying RDBMS first, it will be used in following `DistSQL`.

Example:

```
ADD RESOURCE ds_2 (
    URL="jdbc:mysql://127.0.0.1:3306/db2?serverTimezone=UTC&useSSL=false",
    USER=root,
    PASSWORD=root,
    PROPERTIES("maximumPoolSize"=10,"idleTimeout"="30000")
);
-- ds_3, ds_4
```

2. Alter sharding table rule

Please refer to RDL#Sharding for more details.

`SHARDING TABLE RULE` support two types: `TableRule` and `AutoTableRule`. For each logic table, we could not use mixture of these two types.

Example of alter `AutoTableRule`:

```
ALTER SHARDING TABLE RULE t_order (
RESOURCES(ds_2, ds_3, ds_4),
SHARDING_COLUMN=order_id,
TYPE(NAME=hash_mod,PROPERTIES("sharding-count"=10)),
GENERATED_KEY(COLUMN=another_id,TYPE(NAME=snowflake,PROPERTIES("worker-id"=123)))
);
```

If `RESOURCES` and `sharding-count` is changed, then scaling job will be emitted.

Uncompleted example of alter `TableRule`:

```
ALTER SHARDING TABLE RULE t_order (
DATANODES("ds_${2..4}.t_order_${0..1}"),
```

```
DATABASE_STRATEGY(TYPE=standard,SHARDING_COLUMN=user_id,SHARDING_
ALGORITHM=database_inline),
TABLE_STRATEGY(TYPE=standard,SHARDING_COLUMN=order_id,SHARDING_ALGORITHM=t_order_
inline),
GENERATED_KEY(COLUMN=order_id,TYPE(NAME=snowflake,PROPERTIES("worker-id"=123)))
);
```

Attention: We could not emit scaling job by altering `TableRule` in current version.

List scaling jobs

Please refer to RAL#Scaling for more details.

Example:

```
show scaling list;
```

Response:

```
mysql> show scaling list;
+---------------------+-----------------------+----------------------+--------+-----
----------------+---------------------+
| id                  | tables                | sharding_total_count | active |
create_time         | stop_time           |
+---------------------+-----------------------+----------------------+--------+-----
----------------+---------------------+
| 659853312085983232 | t_order_item, t_order | 2                    | 0      |
2021-10-26 20:21:31 | 2021-10-26 20:24:01 |
| 660152090995195904 | t_order_item, t_order | 2                    | 0      |
2021-10-27 16:08:43 | 2021-10-27 16:11:00 |
+---------------------+-----------------------+----------------------+--------+-----
----------------+---------------------+
2 rows in set (0.04 sec)
```

Get scaling progress

Example:

```
show scaling status {jobId};
```

Response:

```
mysql> show scaling status 660152090995195904;
+------+-------------+----------+-------------------------------+-----------------
--------+
| item | data_source | status   | inventory_finished_percentage | incremental_idle_
minutes |
+------+-------------+----------+-------------------------------+-----------------
--------+
```

```
| 0     | ds_1         | FINISHED | 100                            | 2834
        |
| 1     | ds_0         | FINISHED | 100                            | 2834
        |
+------+-------------+----------+-------------------------------+------------------
--------+
2 rows in set (0.00 sec)
```

Current scaling job is finished, new sharding rule should take effect, and not if scaling job is failed.

`status` values:

Value	Description
PREPARING	preparing
RUNNING	running
EXECUTE_INVENTORY_TASK	inventory task running
EXECUTE_INCREMENTAL_TASK	incremental task running
ALMOST_FINISHED	almost finished
FINISHED	finished
PREPARING_FAILURE	preparation failed
EXECUTE_INVENTORY_TASK_FAILURE	inventory task failed
EXECUTE_INCREMENTAL_TASK_FAILURE	incremental task failed

Preview new sharding rule

Example:

```
preview select count(1) from t_order;
```

Response:

```
mysql> preview select count(1) from t_order;
+------------------+-------------------------------+
| data_source_name | sql                           |
+------------------+-------------------------------+
| ds_2             | select count(1) from t_order_0 |
| ds_2             | select count(1) from t_order_1 |
| ds_3             | select count(1) from t_order_0 |
| ds_3             | select count(1) from t_order_1 |
| ds_4             | select count(1) from t_order_0 |
| ds_4             | select count(1) from t_order_1 |
+------------------+-------------------------------+
6 rows in set (0.01 sec)
```

Other DistSQL

Please refer to RAL#Scaling for more details.

8 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.

8.1 SQL Parser

8.1.1 DatabaseTypedSQLParserFacade

SPI Name	*Description*
DatabaseTypedSQLParserFacade	SQL parser facade for lexer and parser

Implementation Class	*Description*
MySQLParserFacade	SQL parser facade for MySQL
PostgreSQLParserFacade	SQL parser facade for PostgreSQL
SQLServerParserFacade	SQL parser facade for SQLServer
OracleParserFacade	SQL parser facade for Oracle
SQL92ParserFacade	SQL parser facade for SQL92

8.1.2 SQLVisitorFacade

SPI Name	*Description*
SQLVisitorFacade	SQL AST visitor facade

Implementation Class	*Description*
MySQLS tatementSQLVisitorFacade	SQL visitor of statement extracted facade for MySQL
PostgreSQLS tatementSQLVisitorFacade	SQL visitor of statement extracted facade for PostgreSQL
SQLServerS tatementSQLVisitorFacade	SQL visitor of statement extracted facade for SQLServer
OracleS tatementSQLVisitorFacade	SQL visitor of statement extracted facade for Oracle
SQL92S tatementSQLVisitorFacade	SQL visitor of statement extracted facade for SQL92

8.2 Configuration

8.2.1 RuleBuilder

SPI Name	*Description*
RuleBuilder	Used to convert user configurations to rule objects

Implementation Class	*Description*
AlgorithmP rovidedRead-writeSp littingRuleBuilder	Used to convert algorithm-based read-write separation user configuration into read-write separation rule objects
Algorithm Provided-DatabaseDi scoveryRule-Builder	Used to convert algorithm-based database discovery user configuration into database discovery rule objects
AlgorithmProvidedS hard-ingRuleBuilder	Used to convert algorithm-based sharding user configuration into sharding rule objects
AlgorithmProvided Encryp-tRuleBuilder	Used to convert algorithm-based encryption user configuration into encryption rule objects
AlgorithmProvide dShad-owRuleBuilder	Used to convert algorithm-based shadow database user configuration into shadow database rule objects
ReadwriteSp littingRule-Builder	Used to convert read-write separation user configuration into read-write separation rule objects
DatabaseDi scoveryRule-Builder	Used to convert database discovery user configuration into database discovery rule objects
Sing leTableRuleBuilder	Used to convert single-table user configuration into a single-table rule objects
Au thorityRuleBuilder	Used to convert permission user configuration into permission rule objects
S hardingRuleBuilder	Used to convert sharding user configuration into sharding rule objects
EncryptRuleBuilder	Used to convert encrypted user configuration into encryption rule objects
ShadowRuleBuilder	Used to convert shadow database user configuration into shadow database rule objects
Tran sactionRuleBuilder	Used to convert transaction user configuration into transaction rule objects

8.2.2 YamlRuleConfigurationSwapper

SPI Name	*Description*
YamlRu leConfigurationSwap-per	Used to convert YAML configuration to standard user configuration

Implementation Class	*Description*
ReadwriteSplittingR uleAlgorithm-ProviderCo nfigurationYamlSwap-per	Used to convert algorithm-based read-write separation configuration into read-write separation standard configuration
DatabaseDiscoveryR uleAlgorithm-ProviderCo nfigurationYamlSwap-per	Used to convert algorithm-based database discovery configuration into database discovery standard configuration
ShardingR uleAlgorithmProviderCo nfigurationYamlSwapper	Used to convert algorithm-based sharding configuration into sharding standard configuration
EncryptR uleAlgorithmProviderCo nfigurationYamlSwapper	Used to convert algorithm-based encryption configuration into encryption standard configuration
ShadowR uleAlgorithmProviderCo nfigurationYamlSwapper	Used to convert algorithm-based shadow database configuration into shadow database standard configuration
Re adwriteSplittingRuleCo nfigurationYamlSwapper	Used to convert the YAML configuration of read-write separation into the standard configuration of read-write separation
D atabaseDiscoveryRuleCo nfigurationYamlSwapper	Used to convert the YAML configuration of database discovery into the standard configuration of database discovery
AuthorityRuleCo nfigurationYaml-Swapper	Used to convert the YAML configuration of permission rules into standard configuration of permission rules
ShardingRuleCo nfigurationYaml-Swapper	Used to convert the YAML configuration of the shard into the standard configuration of the shard
EncryptRuleCo nfigurationYaml-Swapper	Used to convert encrypted YAML configuration into encrypted standard configuration
ShadowRuleCo nfigurationYaml-Swapper	Used to convert the YAML configuration of the shadow database into the standard configuration of the shadow database
TransactionRuleCo nfigurationYamlSwapper	Used to convert the YAML configuration of the transaction into the standard configuration of the transaction

8.2.3 ShardingSphereYamlConstruct

SPI Name	*Description*
ShardingSphereYamlConstruct	Used to convert customized objects and YAML to each other

Implementation Class	*Description*
NoneShardingStrate gyConfigurationYaml-Construct	Used to convert non sharding strategy and YAML to each other

8.3 Kernel

8.3.1 DatabaseType

SPI Name	*Description*
DatabaseType	Supported database type

Implementation Class	*Description*
SQL92DatabaseType	SQL92 database type
MySQLDatabaseType	MySQL database
MariaDBDatabaseType	MariaDB database
PostgreSQLDatabaseType	PostgreSQL database
OracleDatabaseType	Oracle database
SQLServerDatabaseType	SQLServer database
H2DatabaseType	H2 database
OpenGaussDatabaseType	OpenGauss database

8.3.2 DialectTableMetaDataLoader

SPI Name	*Description*
DialectTableMetaDataLoader	Use SQL dialect to load meta data rapidly

Implementation Class	*Description*
MySQLTableMetaDataLoader	Use MySQL dialect to load meta data
OracleTableMetaDataLoader	Use Oracle dialect to load meta data
PostgreSQLTableMetaDataLoader	Use PostgreSQL dialect to load meta data
SQLServerTableMetaDataLoader	Use SQLServer dialect to load meta data
H2TableMetaDataLoader	Use H2 dialect to load meta data
OpenGaussTableMetaDataLoader	Use OpenGauss dialect to load meta data

8.3.3 SQLRouter

SPI Name	*Description*
SQLRouter	Used to process routing results

Implementation Class	*Description*
Re adwriteSplittingSQLRouter	Used to process read-write separation routing results
D atabaseDiscoverySQLRouter	Used to process database discovery routing results
SingleTableSQLRouter	Used to process single-table routing results
ShardingSQLRouter	Used to process sharding routing results
ShadowSQLRouter	Used to process shadow database routing results

8.3.4 SQLRewriteContextDecorator

SPI Name	*Description*
SQLRewriteContextDecorator	Used to process SQL rewrite results

SPI Name	*Description*
Shardin gSQLRewriteContextDecorator	Used to process sharding SQL rewrite results
Encryp tSQLRewriteContextDecorator	Used to process encryption SQL rewrite results
Shado wSQLRewriteContextDecorator	Used to process shadow SQL rewrite results

8.3.5 SQLExecutionHook

SPI Name	*Description*
SQLExecutionHook	Hook of SQL execution

Implementation Class	*Description*
TransactionalSQLExecutionHook	Transaction hook of SQL execution

8.3.6 ResultProcessEngine

SPI Name	*Description*
ResultProcessEngine	Used by merge engine to process result set

Implementation Class	*Description*
Shard ingResultMergerEngine	Used by merge engine to process sharding result set
Encrypt ResultDecoratorEngine	Used by merge engine to process encryption result set

8.3.7 StoragePrivilegeHandler

SPI Name	*Description*
StoragePrivilegeHandler	Use SQL dialect to process privilege metadata

Implementation Class	*Description*
Postgr eSQLPrivilegeHandler	Use PostgreSQL dialect to process privilege metadata
SQLSe rverPrivilegeHandler	Use SQLServer dialect to process privilege metadata
Or aclePrivilegeHandler	Use Oracle dialect to process privilege metadata
M ySQLPrivilegeHandler	Use MySQL dialect to process privilege metadata

8.4 Data Sharding

8.4.1 ShardingAlgorithm

SPI Name	*Description*
ShardingAlgorithm	Sharding algorithm

Implementation Class	*Description*
Boundar yBasedRangeShardingAlgorithm	Boundary based range sharding algorithm
Volum eBasedRangeShardingAlgorithm	Volume based range sharding algorithm
Co mplexInlineShardingAlgorithm	Complex inline sharding algorithm
A utoIntervalShardingAlgorithm	Mutable interval sharding algorithm
ClassBasedShardingAlgorithm	Class based sharding algorithm
HintInlineShardingAlgorithm	Hint inline sharding algorithm
IntervalShardingAlgorithm	Fixed interval sharding algorithm
HashModShardingAlgorithm	Hash modulo sharding algorithm
InlineShardingAlgorithm	Inline sharding algorithm
ModShardingAlgorithm	Modulo sharding algorithm

8.4.2 KeyGenerateAlgorithm

SPI Name	*Description*
KeyGenerateAlgorithm	Key generate algorithm

Implementation Class	*Description*
SnowflakeKeyGenerateAlgorithm	Snowflake key generate algorithm
UUIDKeyGenerateAlgorithm	UUID key generate algorithm

8.4.3 DatetimeService

SPI Name	*Description*
DatetimeService	Use current time for routing

Implementation Class	*Description*
DatabaseDa tetimeServiceDelegate	Get the current time from the database for routing
SystemDatetimeService	Get the current time from the application system for routing

8.4.4 DatabaseSQLEntry

SPI Name	*Description*
DatabaseSQLEntry	Database dialect for get current time

Implementation Class	*Description*
MySQLDatabaseSQLEntry	MySQL dialect for get current time
PostgreSQLDatabaseSQLEntry	PostgreSQL dialect for get current time
OracleDatabaseSQLEntry	Oracle dialect for get current time
SQLServerDatabaseSQLEntry	SQLServer dialect for get current time

8.5 Readwrite-splitting

8.5.1 ReplicaLoadBalanceAlgorithm

SPI Name	*Description*
ReplicaLoadBalanceAlgorithm	Load balance algorithm of replica databases

Implementation Class	*Description*
RoundRobinRe plicaLoadBalanceAlgorithm	Round robin load balance algorithm of replica databases
RandomRe plicaLoadBalanceAlgorithm	Random load balance algorithm of replica databases

8.6 Data Encryption

8.6.1 EncryptAlgorithm

SPI Name	*Description*
EncryptAlgorithm	Data encrypt algorithm

Implementation Class	*Description*
MD5EncryptAlgorithm	MD5 data encrypt algorithm
AESEncryptAlgorithm	AES data encrypt algorithm
RC4EncryptAlgorithm	Rc4 data encrypt algorithm

8.6.2 QueryAssistedEncryptAlgorithm

SPI Name	*Description*
QueryAss istedEncryptAlgorithm	Data encrypt algorithm which include query assisted column

Implementation Class	*Description*
None	

8.7 SQL Checker

8.7.1 SQLChecker

SPI Name	*Description*
SQLChecker	SQL checker

Implementation Class	*Description*
AuthorityChecker	Authority checker

8.8 Distributed Transaction

8.8.1 ShardingSphereTransactionManager

SPI Name	*Description*
ShardingSphereTransactionManager	Distributed transaction manager

Implementation Class	*Description*
XAShardingSphereTransactionManager	XA distributed transaction manager
Seata ATShardingSphereTransactionManager	Seata distributed transaction manager

8.8.2 XATransactionManagerProvider

SPI Name	*Description*
XATransactionManagerProvider	XA distributed transaction manager

Implementation Class	*Description*
Atomikos TransactionManagerProvider	XA distributed transaction manager based on Atomikos
NarayanaXA TransactionManager-Provider	XA distributed transaction manager based on Narayana
BitronixXA TransactionManagerProvider	XA distributed transaction manager based on Bitronix

8.8.3 XADataSourceDefinition

SPI Name	*Description*
XADataSourceDefinition	Auto convert Non XA data source to XA data source

Implementation Class	*Description*
MySQLXAD ataSourceDefini-tion	Auto convert Non XA MySQL data source to XA MySQL data source
MariaDBXAD ataSourceDefini-tion	Auto convert Non XA MariaDB data source to XA MariaDB data source
PostgreSQLXAD ataSourceDefi-nition	Auto convert Non XA PostgreSQL data source to XA PostgreSQL data source
OracleXAD ataSourceDefinition	Auto convert Non XA Oracle data source to XA Oracle data source
SQLServerXAD ataSourceDefi-nition	Auto convert Non XA SQLServer data source to XA SQLServer data source
H2XAD ataSourceDefinition	Auto convert Non XA H2 data source to XA H2 data source

8.8.4 DataSourcePropertyProvider

SPI Name	*Description*
DataS ourcePropertyProvider	Used to get standard properties of data source pool

Implementation Class	*Description*
HikariCPPropertyProvider	Used to get standard properties of HikariCP

8.9 Mode

8.9.1 StandalonePersistRepository

SPI Name	*Description*
StandalonePersistRepository	Standalone mode Configuration persistence

Implementation Class	*Description*
FileRepository	File persistence

8.9.2 ClusterPersistRepository

SPI Name	*Description*
ClusterPersistRepository	Registry center repository

Implementation Class	*Description*
CuratorZookeeperRepository	ZooKeeper registry center repository
EtcdRepository	Etcd registry center repository

8.9.3 GovernanceWatcher

SPI Name	*Description*
GovernanceWatcher	Governance watcher

Implementation Class	*Description*
StorageNodeStateChangedWatcher	Storage node changed watcher
ComputeNodeStateChangedWatcher	Compute node changed watcher
PropertiesChangedWatcher	Properties changed watcher
PrivilegeNodeChangedWatcher	Privilege changed watcher
GlobalRuleChangedWatcher	Global rule changed watcher
MetaDataChangedWatcher	Meta data changed watcher

8.10 Scaling

8.10.1 ScalingEntry

SPI Name	*Description*
ScalingEntry	Entry of scaling

Implementation Class	*Description*
MySQLScalingEntry	MySQL entry of scaling
PostgreSQLScalingEntry	PostgreSQL entry of scaling

8.10.2 ScalingClusterAutoSwitchAlgorithm

SPI Name	*Description*
ScalingClusterAutoSwitchAlgorithm	Scaling job completion check algorithm

Implementation Class	*Description*
Scal ingIdleClusterAutoSwitchAlgorithm	Incremental task idle time based algorithm

8.10.3 ScalingDataConsistencyCheckAlgorithm

SPI Name	*Description*
Sca lingDataConsistencyCheck-Algorithm	Data consistency check algorithm on source and target database cluster

Implementation Class	*Description*
ScalingDef aultDataConsistencyCheckAlgo-rithm	Default implementation with CRC32 of all records.

8.11 Proxy

8.11.1 DatabaseProtocolFrontendEngine

SPI Name	*Description*
DatabaseProtocolFrontendEngine	Regulate parse and adapter protocol of database access for ShardingSphere-Proxy

Implementation Class	*Description*
MySQLFrontendEngine	Base on MySQL database protocol
PostgreSQLFrontendEngine	Base on PostgreSQL database protocol
OpenGaussFrontendEngine	Base on openGauss database protocol

8.11.2 JDBCDriverURLRecognizer

SPI Name	*Description*
JDBCDriverURLRecognizer	Use JDBC driver to execute SQL

Implementation Class	*Description*
MySQLRecognizer	Use MySQL JDBC driver to execute SQL
PostgreSQLRecognizer	Use PostgreSQL JDBC driver to execute SQL
OracleRecognizer	Use Oracle JDBC driver to execute SQL
SQLServerRecognizer	Use SQLServer JDBC driver to execute SQL
H2Recognizer	Use H2 JDBC driver to execute SQL
P6SpyDriverRecognizer	Use P6Spy JDBC driver to execute SQL
OpenGaussRecognizer	Use openGauss JDBC driver to execute SQL

8.11.3 AuthorityProvideAlgorithm

SPI Name	*Description*
AuthorityProvideAlgorithm	User authority loading logic

Implementation Class	*Type*	*Description*
NativeAuthorityP roviderAlgorithm	NATIVE	Persist user authority defined in server.yaml into the backend database. An admin user will be created if not existed.
AllPrivilegesPer mittedAuthorityP roviderAlgorithm	ALL_PR IVILEG ES_PER MITTED	All privileges granted to user by default (No authentication). Will not interact with the actual database.
Sch emaPrivilegesPer mittedAuthorityP roviderAlgorithm	SCH EMA_PR IVILEG ES_PER MITTED	Permissions configured through the attribute user-schema-mappings.

8.12 Shadow DB

8.12.1 ShadowAlgorithm

SPI Name	*Description*
ShadowAlgorithm	shadow routing algorithm

Implementation Class	*Description*
ColumnValueMatchShadowAlgorithm	Column value match shadow algorithm
ColumnRegexMatchShadowAlgorithm	Column regex match shadow algorithm
SimpleSQLNoteShadowAlgorithm	Simple SQL note shadow algorithm

9

Reference

This chapter contains a section of technical implementation and test process with Apache Sharding-Sphere, which provide the reference with users and developers.

9.1 Management

9.1.1 Data Structure in Registry Center

Under defined namespace, `rules`, `props` and `metadata` nodes persist in YAML, modifying nodes can dynamically refresh configurations. `status` 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
    ├         ├──${schema_1}                          # Schema name 1
    ├         ├         ├──dataSources                # Datasource configuration
    ├         ├         ├──rules                      # Rule configuration
    ├         ├         ├──schema                     # Table configuration
    ├         ├──${schema_2}                          # Schema name 2
    ├         ├         ├──dataSources                # Datasource configuration
    ├         ├         ├──rules                      # Rule configuration
    ├         ├         ├──schema                     # Table configuration
    ├──status
    ├     ├──compute_nodes
    ├     ├     ├──online
    ├     ├     ├     ├──${your_instance_ip_a}@${your_instance_port_x}
    ├     ├     ├     ├──${your_instance_ip_b}@${your_instance_port_y}
    ├     ├     ├     ├──....
    ├     ├     ├──circuit_breaker
    ├     ├     ├     ├──${your_instance_ip_c}@${your_instance_port_v}
    ├     ├     ├     ├──${your_instance_ip_d}@${your_instance_port_w}
```

```
    ├    ├     ├     ├─....
    ├    ├─storage_nodes
    ├    ├     ├─disable
    ├    ├     ├      ├─${schema_1.ds_0}
    ├    ├     ├      ├─${schema_1.ds_1}
    ├    ├     ├      ├─....
    ├    ├     ├─primary
    ├    ├     ├      ├─${schema_2.ds_0}
    ├    ├     ├      ├─${schema_2.ds_1}
    ├    ├     ├      ├─....
```

/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_PRIVILEGES_PERMITTED
```

/props

Properties configuration. Please refer to Configuration Manual for more details.

```
kernel-executor-size: 20
sql-show: true
```

/metadata/${schemaName}/dataSources

A collection of multiple database connection pools, whose properties (e.g. DBCP, C3P0, Druid and HikariCP) are configured by users themselves.

```
ds_0:
  dataSourceClassName: com.zaxxer.hikari.HikariDataSource
  props:
    url: jdbc:mysql://127.0.0.1:3306/demo_ds_0?serverTimezone=UTC&useSSL=false
    password: null
    maxPoolSize: 50
    connectionTimeoutMilliseconds: 30000
    idleTimeoutMilliseconds: 60000
    minPoolSize: 1
    username: root
    maxLifetimeMilliseconds: 1800000
```

```
ds_1:
  dataSourceClassName: com.zaxxer.hikari.HikariDataSource
  props:
    url: jdbc:mysql://127.0.0.1:3306/demo_ds_1?serverTimezone=UTC&useSSL=false
    password: null
    maxPoolSize: 50
    connectionTimeoutMilliseconds: 30000
    idleTimeoutMilliseconds: 60000
    minPoolSize: 1
    username: root
    maxLifetimeMilliseconds: 1800000
```

/metadata/${schemaName}/rules

Rule configurations, including sharding, readwrite-splitting, data encryption, shadow DB configurations.

```
- !SHARDING
  xxx

- !READWRITE_SPLITTING
  xxx

- !ENCRYPT
  xxx
```

/metadata/${schemaName}/schema

Dynamic modification of metadata content is not supported currently.

```
tables:                                              # Tables
  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:                                          # Indexes
```

```
      t_user_order_id_index:                 # index_name
        name: t_user_order_id_index
  t_order_item:
    columns:
      order_id:
        caseSensitive: false
        dataType: 0
        generated: false
        name: order_id
        primaryKey: false
```

/status/compute_nodes

It includes running instance information of database access object, with sub-nodes as the identifiers of currently running instance, which consist of IP and PORT. 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.

/status/storage_nodes

It is able to orchestrate replica database, delete or disable data dynamically.

9.2 Sharding

The major sharding processes of all the three ShardingSphere products are identical. According to whether query optimization is performed, they can be divided into standard kernel process and federation executor engine process. The standard kernel process consists of `SQL Parse => SQL Route => SQL Rewrite => SQL Execute => Result Merge`, which is used to process SQL execution in standard sharding scenarios. The federation executor engine process consists of `SQL Parse => Logical Plan Optimize => Physical Plan Optimize => Plan Execute => Standard Kernel Process`. The federation executor engine perform logical plan optimization and physical plan optimization. In the optimization execution phase, it relies on the standard kernel process to route, rewrite, execute, and merge the optimized logical SQL.

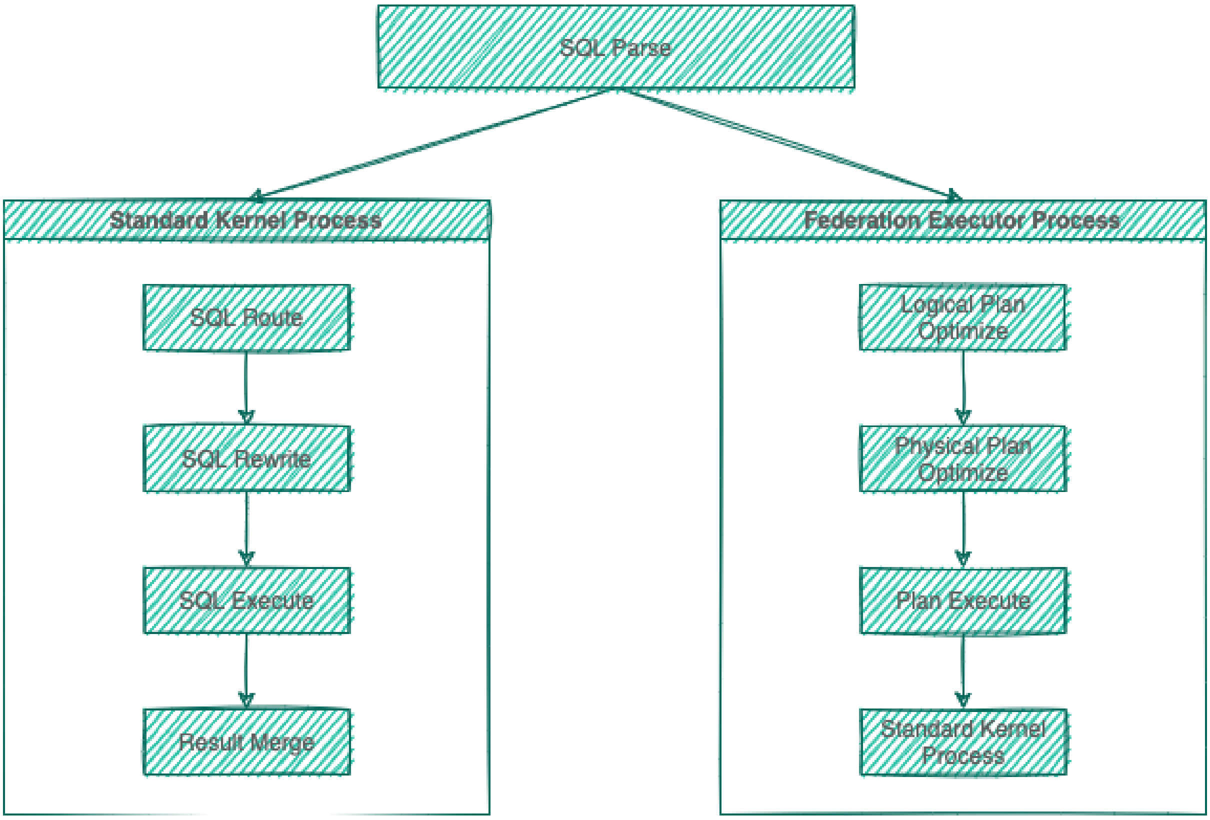

Figure1: Sharding Architecture Diagram

9.2.1 SQL Parsing

It is divided into lexical parsing and syntactic parsing. The lexical parser will split SQL into inseparable words, and then the syntactic parser will analyze SQL and extract the parsing context, which can include tables, options, ordering items, grouping items, aggregation functions, pagination information, query conditions and placeholders that may be revised.

9.2.2 SQL Route

It is the sharding strategy that matches users' configurations according to the parsing context and the route path can be generated. It supports sharding route and broadcast route currently.

9.2.3 SQL Rewrite

It rewrites SQL as statement that can be rightly executed in the real database, and can be divided into correctness rewrite and optimization rewrite.

9.2.4 SQL Execution

Through multi-thread executor, it executes asynchronously.

9.2.5 Result Merger

It merges multiple execution result sets to output through unified JDBC interface. Result merger includes methods as stream merger, memory merger and addition merger using decorator merger.

9.2.6 Query Optimization

Supported by federation executor engine(under development), optimization is performed on complex query such as join query and subquery. It also supports distributed query across multiple database instances. It uses relational algebra internally to optimize query plan, and then get query result through the best query plan.

9.2.7 Parse Engine

Compared to other programming languages, SQL is relatively simple, but it is still a complete set of programming language, so there is no essential difference between parsing SQL grammar and parsing other languages (Java, C and Go, etc.).

Abstract Syntax Tree

The parsing process can be divided into lexical parsing and syntactic parsing. Lexical parser is used to divide SQL into indivisible atomic signs, i.e., Token. According to the dictionary provided by different database dialect, it is categorized into keyword, expression, literal value and operator. SQL is then converted into abstract syntax tree by syntactic parser.

For example, the following SQL:

```
SELECT id, name FROM t_user WHERE status = 'ACTIVE' AND age > 18
```

Its parsing AST (Abstract Syntax Tree) is this:

To better understand, the Token of keywords in abstract syntax tree is shown in green; that of variables is shown in red; what' s to be further divided is shown in grey.

At last, through traversing the abstract syntax tree, the context needed by sharding is extracted and the place that may need to be rewritten is also marked out. Parsing context for the use of sharding includes select items, table information, sharding conditions, auto-increment primary key information, Order By information, Group By information, and pagination information (Limit, Rownum and Top). One-time SQL parsing process is irreversible, each Token is parsed according to the original order of SQL in a high performance. Considering similarities and differences between SQL of all kinds of database dialect, SQL dialect dictionaries of different types of databases are provided in the parsing module.

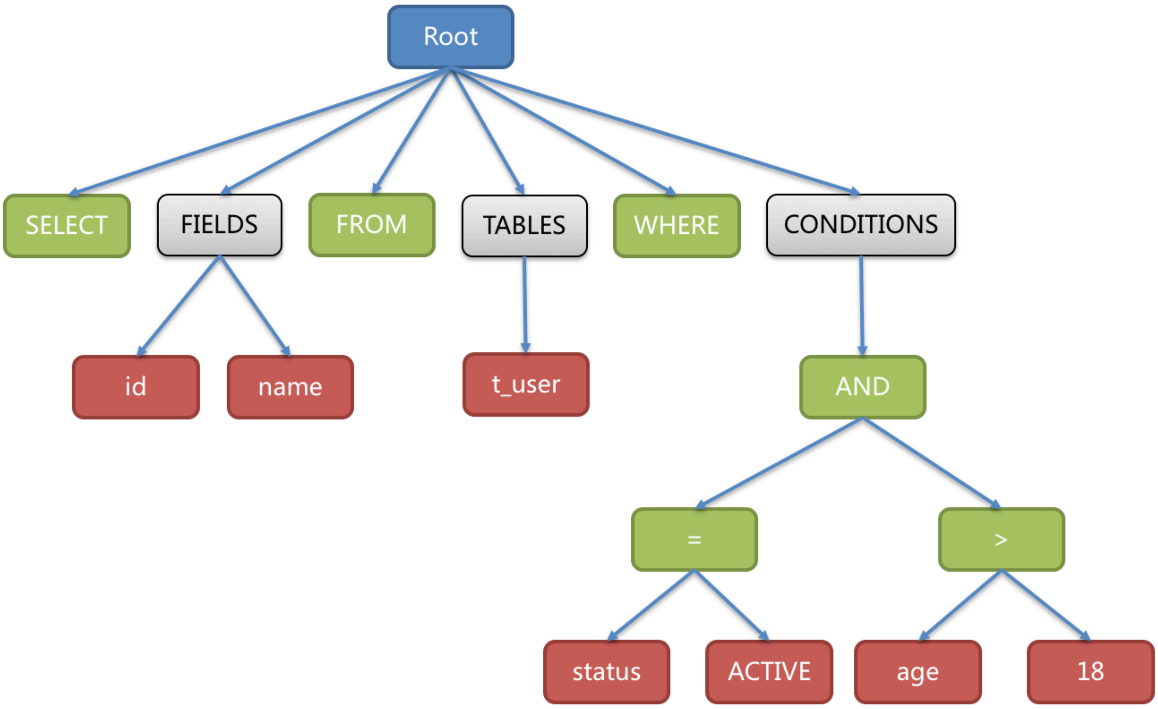

Figure2: SQL AST

SQL Parser

History

As the core of database sharding and table sharding, SQL parser takes the performance and compatibility as its most important index. ShardingSphere SQL parser has undergone the upgrade and iteration of 3 generations of products.

To pursue good performance and quick achievement, the first generation of SQL parser uses `Druid` before 1.4.x version. As tested in practice, its performance exceeds other parsers a lot.

The second generation of SQL parsing engine begins from 1.5.x version, ShardingSphere has adopted fully self-developed parsing engine ever since. Due to different purposes, ShardingSphere does not need to transform SQL into a totally abstract syntax tree or traverse twice through visitor. Using `half parsing` method, it only extracts the context required by data sharding, so the performance and compatibility of SQL parsing is further improved.

The third generation of SQL parsing engine begins from 3.0.x version. ShardingSphere tries to adopts ANTLR as a generator for the SQL parsing engine, and uses Visit to obtain SQL Statement from AST. Starting from version 5.0.x, the architecture of the parsing engine has been refactored. At the same time, it is convenient to directly obtain the parsing results of the same SQL to improve parsing efficiency by putting the AST obtained from the first parsing into the cache. Therefore, we recommend that users adopt `PreparedStatement` this SQL pre-compilation method to improve performance. Currently, users can also use ShardingSphere's SQL parsing engine independently to obtain AST and SQL Statements for a variety of mainstream relational databases. In the future, the SQL parsing engine will continue to provide powerful functions such as SQL formatting and SQL templating.

Features

- Independent SQL parsing engine
- The syntax rules can be easily expanded and modified (using ANTLR)
- Support multiple dialects

DB	Status
MySQL	supported
PostgreSQL	supported
SQLServer	supported
Oracle	supported
SQL92	supported

- SQL format (developing)
- SQL parameterize (developing)

API Usage

Maven config

```
<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>
```

demo:

- Get AST

```
/**
 * databaseType type:String values: MySQL, Oracle, PostgreSQL, SQL92, SQLServer,
openGauss
 * sql type:String SQL to be parsed
 * useCache type:boolean whether use cache
 * @return parse context
 */
ParseContext parseContext = new SQLParserEngine(databaseType).parse(sql, useCache)
```

- GET SQLStatement

```
/**
 * databaseType type:String values: MySQL, Oracle, PostgreSQL, SQL92, SQLServer,
openGauss
 * useCache type:boolean whether use cache
 * @return SQLStatement
 */
ParseContext parseContext = new SQLParserEngine(databaseType).parse(sql, useCache);
SQLVisitorEngine sqlVisitorEngine = new SQLVisitorEngine(databaseType, "STATEMENT
");
SQLStatement sqlStatement = sqlVisitorEngine.visit(parseContext);
```

- SQL Format

```
/**
 * databaseType type:String values MySQL
 * useCache type:boolean whether use cache
 * @return String
 */
ParseContext parseContext = new SQLParserEngine(databaseType).parse(sql, useCache);
SQLVisitorEngine sqlVisitorEngine = new SQLVisitorEngine(databaseType, "FORMAT",
new Properties());
String formatedSql = sqlVisitorEngine.visit(parseContext);
```

example:

sql	formatedSql
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 , 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 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;

9.2.8 Route Engine

It refers to the sharding strategy that matches databases and tables according to the parsing context and generates route path. SQL with sharding keys can be divided into single-sharding route (equal mark as the operator of sharding key), multiple-sharding route (IN as the operator of sharding key) and range sharding route (BETWEEN as the operator of sharding key). SQL without sharding key adopts broadcast route.

Sharding strategies can usually be set in the database or by users. Strategies built in the database are relatively simple and can generally be divided into last number modulo, hash, range, tag, time and so on. More flexible, sharding strategies set by users can be customized according to their needs. Together with automatic data migration, database middle layer can automatically shard and balance the data without users paying attention to sharding strategies, and thereby the distributed database can have the elastic scaling-out ability. In ShardingSphere' s roadmap, elastic scaling-out ability will start from 4.x version.

Sharding Route

It is used in the situation to route according to the sharding key, and can be sub-divided into 3 types, direct route, standard route and Cartesian product route.

Direct Route

The conditions for direct route are relatively strict. It requires to shard through Hint (use HintAPI to appoint the route to databases and tables directly). On the premise of having database sharding but not table sharding, SQL parsing and the following result merging can be avoided. Therefore, with the highest compatibility, it can execute any SQL in complex situations, including sub-queries, self-defined functions. Direct route can also be used in the situation where sharding keys are not in SQL. For example, set sharding key as `3`.

```
hintManager.setDatabaseShardingValue(3);
```

If the routing algorithm is `value % 2`, when a logical database `t_order` corresponds to two physical databaset`_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

Standard route is ShardingSphere's most recommended sharding method. Its application range is the SQL that does not include joint query or only includes joint query between binding tables. When the sharding operator is equal mark, the route result will fall into a single database (table); when sharding operators are BETWEEN or IN, the route result will not necessarily fall into the only database (table). So one logic SQL can finally be split into multiple real SQL to execute. For example, if sharding is according to the odd number or even number of `order_id`, a single table query SQL is as the following:

```
SELECT * FROM t_order WHERE order_id IN (1, 2);
```

The route result will be:

```
SELECT * FROM t_order_0 WHERE order_id IN (1, 2);
SELECT * FROM t_order_1 WHERE order_id IN (1, 2);
```

The complexity and performance of the joint query are comparable with those of single-table query. For instance, if a joint query SQL that contains binding tables is as this:

```
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 route result will 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_1 o JOIN t_order_item_1 i ON o.order_id=i.order_id  WHERE
order_id IN (1, 2);
```

It can be seen that, the number of divided SQL is the same as the number of single tables.

Cartesian Route

Cartesian route has the most complex situation, it cannot locate sharding rules according to the binding table relationship, so the joint query between non-binding tables needs to be split into Cartesian product combination to execute. If SQL in the last case is not configured with binding table relationship, the route result will 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);
```

Cartesian product route has a relatively low performance, so it should be careful to use.

Broadcast Route

For SQL without sharding key, broadcast route is used. According to SQL types, it can be divided into five types, schema & table route, database schema route, database instance route, unicast route and ignore route.

Schema & Table Route

Schema & table route is used to deal with all the operations of physical tables related to its logic table, including DQL and DML without sharding key and DDL, etc. For example.

```
SELECT * FROM t_order WHERE good_prority IN (1, 10);
```

It will traverse all the tables in all the databases, match the logical table and the physical table name one by one and execute them if succeeded. After routing, they are:

```
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);
```

Database Schema Route

Database schema route is used to deal with database operations, including the SET database management order used to set the database and transaction control statement as TCL. In this case, all physical databases matched with the name are traversed according to logical database name, and the command is executed in the physical database. For example:

```
SET autocommit=0;
```

If this command is executed in `t_order`, `t_order` will have 2 physical databases. And it will actually be executed in both `t_order_0` and `t_order_1`.

Database Instance Route

Database instance route is used in DCL operation, whose authorization statement aims at database instances. No matter how many schemas are included in one instance, each one of them can only be executed once. For example:

```
CREATE USER customer@127.0.0.1 identified BY '123';
```

This command will be executed in all the physical database instances to ensure customer users have access to each instance.

Unicast Route

Unicast route is used in the scenario of acquiring the information from some certain physical table. It only requires to acquire data from any physical table in any database. For example:

```
DESCRIBE t_order;
```

The descriptions of the two physical tables, t_order_0 and t_order_1 of t_order have the same structure, so this command is executed once on any physical table.

Ignore Route

Ignore route is used to block the operation of SQL to the database. For example:

```
USE order_db;
```

This command will not be executed in physical database. Because ShardingSphere uses logic Schema, there is no need to send the Schema shift order to the database.

The overall structure of route engine is as the following:

9.2.9 Rewrite Engine

The SQL written by engineers 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.

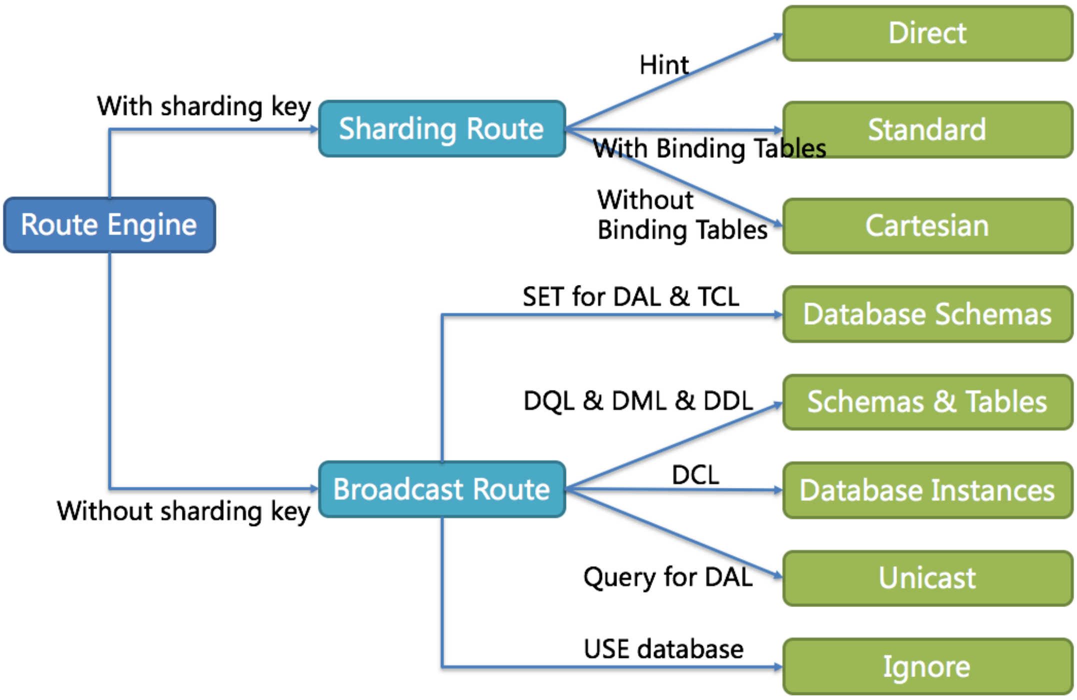

Figure3: Route Engine

Correctness Rewrite

In situation with sharding tables, it requires to rewrite logic table names in sharding settings into actual table names acquired after routing. Database sharding does not require to rewrite table names. In addition to that, there are also column derivation, pagination information revision and other content.

Identifier Rewrite

Identifiers that need to be rewritten include table name, index name and schema name. Table name rewrite refers to the process to locate the position of logic tables in the original SQL and rewrite it as the physical table. Table name rewrite is one typical situation that requires to parse SQL. From a most plain case, if the logic SQL is as follow:

```
SELECT order_id FROM t_order WHERE order_id=1;
```

If the SQL is configured with sharding key order_id=1, it will be routed to Sharding Table 1. Then, the SQL after rewrite should be:

```
SELECT order_id FROM t_order_1 WHERE order_id=1;
```

In this most simple kind of SQL, whether parsing SQL to abstract syntax tree seems unimportant, SQL can be rewritten only by searching for and substituting characters. But in the following situation, it is unable to rewrite SQL rightly merely by searching for and substituting characters:

```sql
SELECT order_id FROM t_order WHERE order_id=1 AND remarks=' t_order xxx';
```

The SQL rightly rewritten is supposed to be:

```sql
SELECT order_id FROM t_order_1 WHERE order_id=1 AND remarks=' t_order xxx';
```

Rather than:

```sql
SELECT order_id FROM t_order_1 WHERE order_id=1 AND remarks=' t_order_1 xxx';
```

Because there may be similar characters besides the table name, the simple character substitute method cannot be used to rewrite SQL. Here is another more complex SQL rewrite situation:

```sql
SELECT t_order.order_id FROM t_order WHERE t_order.order_id=1 AND remarks=' t_order
xxx';
```

The SQL above takes table name as the identifier of the field, so it should also be revised when SQL is rewritten:

```sql
SELECT t_order_1.order_id FROM t_order_1 WHERE t_order_1.order_id=1 AND remarks='
t_order xxx';
```

But if there is another table name defined in SQL, it is not necessary to revise that, even though that name is the same as the table name. For example:

```sql
SELECT t_order.order_id FROM t_order AS t_order WHERE t_order.order_id=1 AND
remarks=' t_order xxx';
```

SQL rewrite only requires to revise its table name:

```sql
SELECT t_order.order_id FROM t_order_1 AS t_order WHERE t_order.order_id=1 AND
remarks=' t_order xxx';
```

Index name is another identifier that can be rewritten. In some databases (such as MySQL/SQLServer), the index is created according to the table dimension, and its names in different tables can repeat. In some other databases (such as PostgreSQL/Oracle), however, the index is created according to the database dimension, index names in different tables are required to be one and the only.

In ShardingSphere, schema management method is similar to that of the table. It uses logic schema to manage a set of data sources, so it requires to replace the logic schema written by users in SQL with physical database schema.

ShardingSphere only supports to use schema in database management statements but not in DQL and DML statements, for example:

```sql
SHOW COLUMNS FROM t_order FROM order_ds;
```

Schema rewrite refers to rewriting logic schema as a right and real schema found arbitrarily with unicast route.

Column Derivation

Column derivation in query statements usually results from two situations. First, ShardingSphere needs to acquire the corresponding data when merging results, but it is not returned through the query SQL. This kind of situation aims mainly at GROUP BY and ORDER BY. Result merger requires sorting and ranking according to items of `GROUP BY` and `ORDER BY`field. But if sorting and ranking items are not included in the original SQL, it should be rewritten. Look at the situation where the original SQL has the information required by result merger:

```
SELECT order_id, user_id FROM t_order ORDER BY user_id;
```

Since user_id is used in ranking, the result merger needs the data able to acquire user_id. The SQL above is able to acquire user_id data, so there is no need to add columns.

If the selected item does not contain the column required by result merger, it will need to add column, as the following SQL:

```
SELECT order_id FROM t_order ORDER BY user_id;
```

Since the original SQL does not contain user_id needed by result merger, the SQL needs to be rewritten by adding columns, and after that, it will be:

```
SELECT order_id, user_id AS ORDER_BY_DERIVED_0 FROM t_order ORDER BY user_id;
```

What's to be mentioned, column derivation will only add the missing column rather than all of them; the SQL that includes * in SELECT will also selectively add columns according to the meta-data information of tables. Here is a relatively complex SQL column derivation case:

```
SELECT o.* FROM t_order o, t_order_item i WHERE o.order_id=i.order_id ORDER BY
user_id, order_item_id;
```

Suppose only t_order_item table contains order_item_id column, according to the meta-data information of tables, the user_id in sorting item exists in table t_order as merging result, but order_item_id does not exist in t_order, so it needs to add columns. The SQL after that will be:

```
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;
```

Another situation of column derivation is using AVG aggregation function. In distributed situations, it is not right to calculate the average value with avg1 + avg2 + avg3 / 3, and it should be rewritten as (sum1 + sum2 + sum3) / (count1 + count2 + count3). This requires to rewrite the SQL that contains AVG as SUM and COUNT and recalculate the average value in result merger. Such as the following SQL:

```
SELECT AVG(price) FROM t_order WHERE user_id=1;
```

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 it can calculate the right average value through result merger.

The last kind of column derivation is in SQL with INSERT. With database auto-increment key, there is no need to fill in primary key field. But database auto-increment key cannot satisfy the requirement of only one primary key being in the distributed situation. So ShardingSphere provides a distributed auto-increment key generation strategy, enabling users to replace the current auto-increment key invisibly with a distributed one without changing existing codes through column derivation. Distributed auto-increment key generation strategy will be expounded in the following part, here we only explain the content related to SQL rewrite. For example, if the primary key of t_order is order_id, and the original SQL is:

```
INSERT INTO t_order (`field1`, `field2`) VALUES (10, 1);
```

It can be seen that the SQL above does not include an auto-increment key, which will be filled by the database itself. After ShardingSphere set an auto-increment key, the SQL will be rewritten as:

```
INSERT INTO t_order (`field1`, `field2`, order_id) VALUES (10, 1, xxxxx);
```

Rewritten SQL will add auto-increment key name and its value generated automatically in the last part of INSERT FIELD and INSERT VALUE. `xxxxx` in the SQL above stands for the latter one.

If INSERT SQL does not contain the column name of the table, ShardingSphere can also automatically generate auto-increment key by comparing the number of parameter and column in the table meta-information. For example, the original SQL is:

```
INSERT INTO t_order VALUES (10, 1);
```

The rewritten SQL only needs to add an auto-increment key in the column where the primary key is:

```
INSERT INTO t_order VALUES (xxxxx, 10, 1);
```

When auto-increment key derives column, if the user writes SQL with placeholder, he only needs to rewrite parameter list but not SQL itself.

Pagination Revision

The scenarios 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 right to take, acquire LIMIT 10, 10 under sharding situations, and take out the first 10 pieces of data according to sorting conditions after merging. For example, if the SQL is:

```
SELECT score FROM t_score ORDER BY score DESC LIMIT 1, 2;
```

The following picture shows the pagination execution results without SQL rewrite.

As shown in the picture, if you want to acquire the second and the third piece of data ordered by score common in both tables, and they are supposed to be `95` and `90`. Since the 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

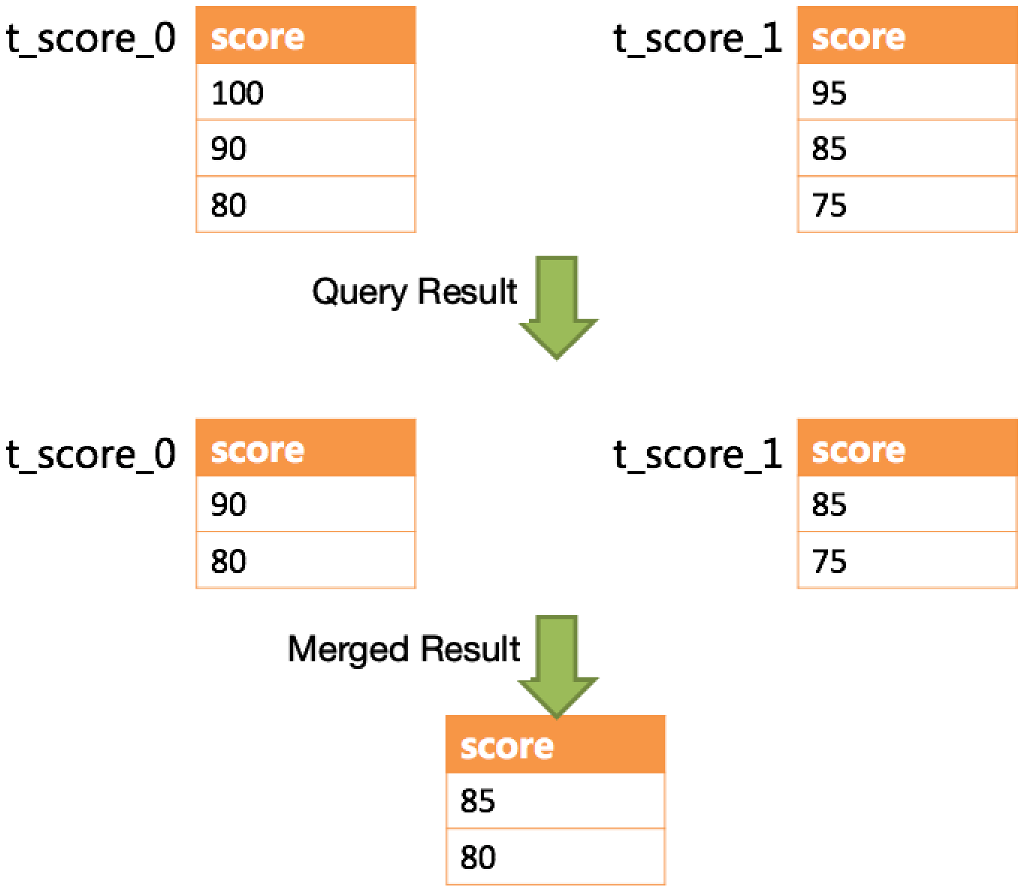

Figure4: Pagination without rewrite

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 combine sorting conditions to calculate the right data. The following picture shows the execution of pagination results after SQL rewrite.

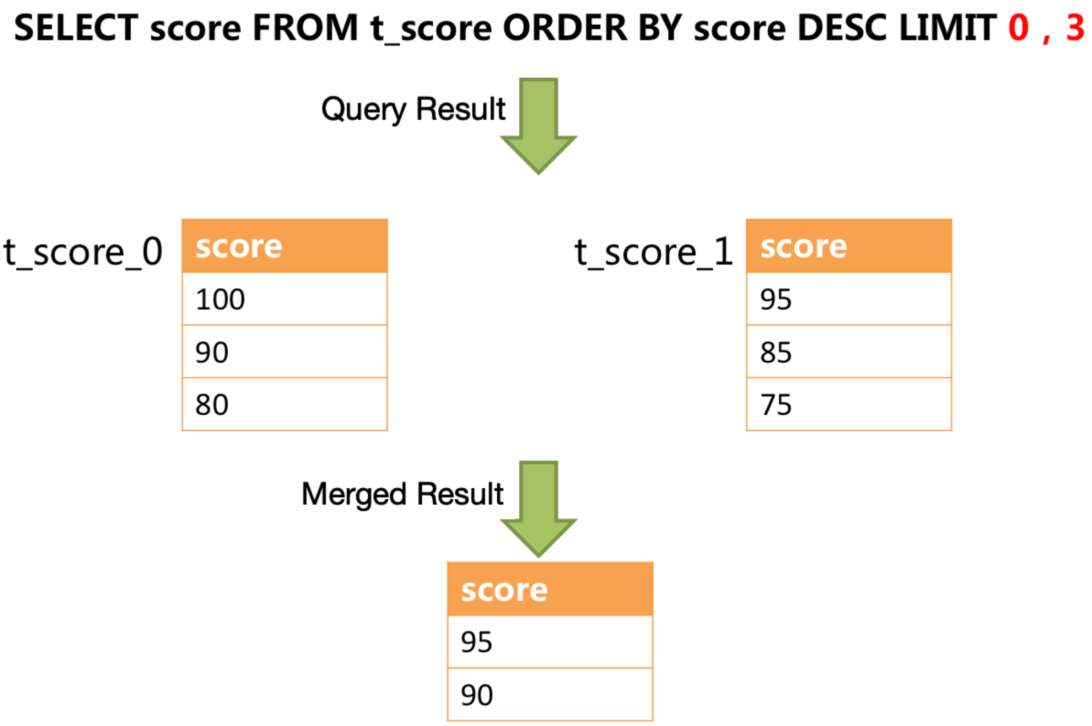

Figure5: Pagination with rewrite

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 record line record number and line offset amount, or using the tail ID of last pagination data as the pagination method of conditions of the next query.

When revising pagination information, if the user uses placeholder method to write SQL, he only needs to rewrite parameter list rather than SQL itself.

Batch Split

When using batch inserted SQL, if the inserted data crosses sharding, the user needs to rewrite SQL to avoid writing excessive data into the database. The differences between insert operation and query operation are: though the query sentence has used sharding keys that do not exist in current sharding, they will not have any influence on data, but insert operation has to delete extra sharding keys. Take the following SQL for example:

```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 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 taken out from query statement, it is not reasonable for the schema to have excessive data. So the 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. Like the following SQL:

```
SELECT * FROM t_order WHERE order_id IN (1, 2, 3);
```

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, but it has not achieved the most optimized query efficiency.

Optimization Rewrite

Its purpose is to effectively improve the 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, result merging is unnecessary to be involved, so there is no need for rewrites as derived column, pagination information and others. In particular, there is no need to read from the first piece of information, which reduces the pressure for the database to a large extent and saves meaningless consumption of the network bandwidth.

Stream Merger Optimization

It only adds sorting items and sorting orders identical with grouping items and `ORDER BY` to `GROUP BY` SQL, and they are used to transfer memory merger to stream merger. In the result merger part, stream merger and memory merger will be explained in detail.

The overall structure of rewrite engine is shown in the following picture.

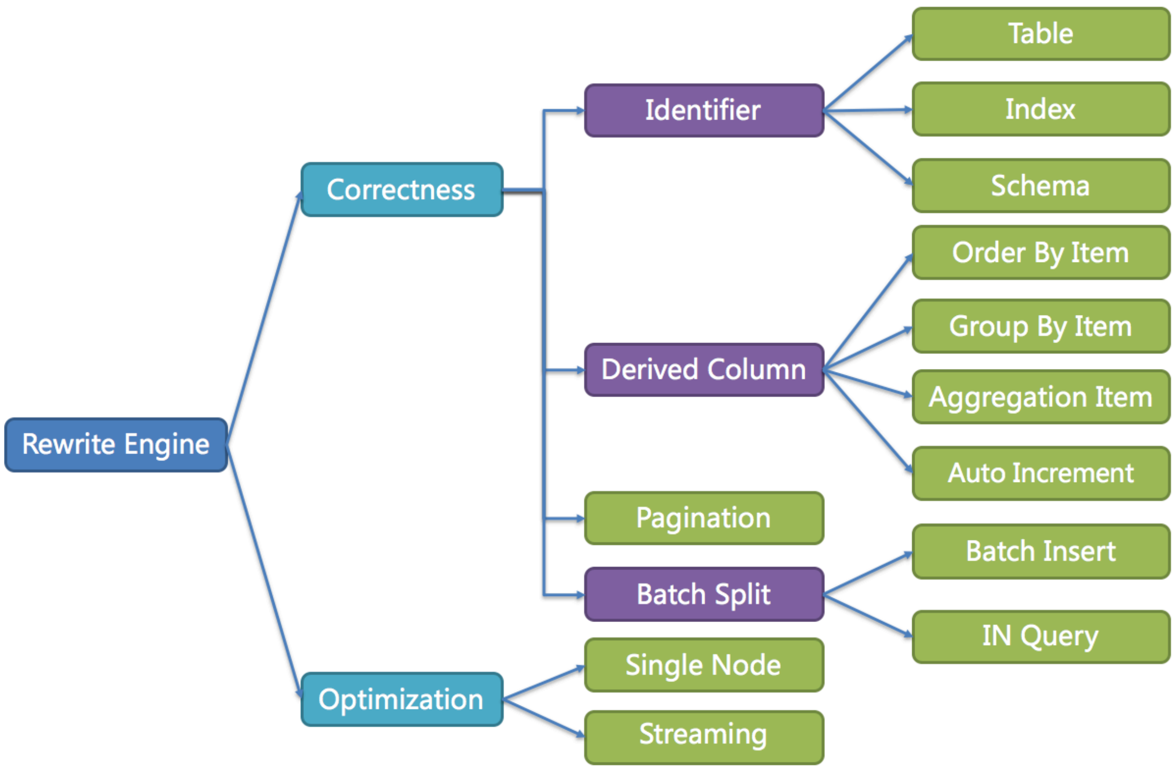

Figure6: Rewrite Engine

9.2.10 Execute Engine

ShardingSphere adopts a set of automatic execution engine, responsible for sending the true SQL, which has been routed and rewritten, to execute in the underlying data source safely and effectively. It does not simply send the SQL through JDBC to directly execute in the underlying data source, or put execution requests directly to the thread pool to concurrently execute, but focuses more on the creation of a balanced data source connection, the consumption generated by the memory usage, the maximum utilization of the concurrency and other problems. The objective of the execution engine is to automatically balance between the resource control and the execution efficiency.

Connection Mode

From the perspective of resource control, the connection number of the business side' s visit of the database should be limited. It can effectively prevent some certain business from occupying excessive resource, exhausting database connection resources and influencing the normal use of other businesses. Especially when one database contains many tables, a logic SQL that does not contain any sharding key will produce a large amount of physical SQLs that fall into different tables in one database. If each physical SQL takes an independent connection, a query will undoubtedly take up excessive resources.

From the perspective of execution efficiency, holding an independent database connection for each sharding query can make effective use of multi-thread to improve execution efficiency. Opening an independent thread for each database connection can parallelize IO produced consumption. Holding an independent database connection for each sharding query can also avoid loading the query result to the memory too early. It is enough for independent database connections to maintain result set quotation and cursor position, and move the cursor when acquiring corresponding data.

Merging result set by moving down its cursor is called stream merger. It does not require to load all the query results to the memory. Thus, it is able to save memory resource effectively and reduce trash recycle frequency. When it is not able to make sure each sharding query holds an independent database connection, it requires to load all the current query results to the memory before reusing that database connection to acquire the query result from the next sharding table. Therefore, though the stream merger can be used, under this kind of circumstances, it will also degenerate to the memory merger.

The control and protection of database connection resources is one thing, adopting better merging model to save the memory resources of middleware is another thing. How to deal with the relationship between them is a problem that ShardingSphere execution engine should solve. To be accurate, if a sharding SQL needs to operate 200 tables under some database case, should we choose to create 200 parallel connection executions or a serial connection execution? Or to say, how to choose between efficiency and resource control?

Aiming at the above situation, ShardingSphere has provided a solution. It has put forward a Connection Mode concept divided into two types, `MEMORY_STRICTLY` mode and `CONNECTION_STRICTLY` mode.

MEMORY_STRICTLY Mode

The prerequisite to use 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 the SQL is up to standard, it will choose stream merger in priority to avoid memory overflow or frequent garbage recycle.

CONNECTION_STRICTLY Mode

The prerequisite to use 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 database instance, it will create one database connection and operate them serially. If shards exist in different databases, it will still be multi-thread operations for different databases, but with only one database connection being created for each operation in each database. It can prevent the problem brought by excessive occupation of database connection from one request. The mode chooses memory merger all the time.

The MEMORY_STRICTLY mode is applicable to OLAP operation and can increase the system capacity by removing database connection restrictions. It is also applicable to OLTP operation, which usually has sharding keys and can be routed to a single shard. So it is a wise choice to control database connection strictly to make sure resources of online system databases can be used by more applications.

Automatic Execution Engine

ShardingSphere uses which mode at first is up to users' setting and they can choose to use MEMORY_STRICTLY mode or CONNECTION_STRICTLY mode according to their actual business scenarios.

The solution gives users the right to choose, requiring them to know the advantages and disadvantages of both modes and make decision according to the actual business situations. No doubt, it is not the best solution due to increasing users' study cost and use cost.

This kind of dichotomy solution lacks flexible coping ability to switch between two modes with static initialization. In practical situations, route results of each time may differ with different SQL and placeholder indexes. It means some operations may need to use memory merger, while others are better to use stream merger. Connection modes should not be set by users before initializing ShardingSphere, but should be decided dynamically by the situation of SQL and placeholder indexes.

To reduce users' use cost and solve the dynamic connection mode problem, ShardingSphere has extracted the thought of automatic execution engine in order to eliminate the connection mode concept inside. Users do not need to know what are so called MEMORY_STRICTLY mode and CONNECTION_STRICTLY mode, but let the execution engine to choose the best solution according to current situations.

Automatic execution engine has narrowed the selection scale of connection mode to each SQL operation. Aiming at each SQL request, 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 most optimized balance between resource control and efficiency. For 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 can be 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 realize the internal connection model concept. According to the configuration option of `maxConnectionSizePerQuery`, execution engine will choose an appropriate connection mode combined with current route result.

Detailed steps are as follow:

1. Group SQL route results according to data source names.
2. Through the equation in the following picture, users can acquire the SQL route result group to be executed by each database case within the `maxConnectionSizePerQuery` permission range and calculate the most optimized connection mode of this request.

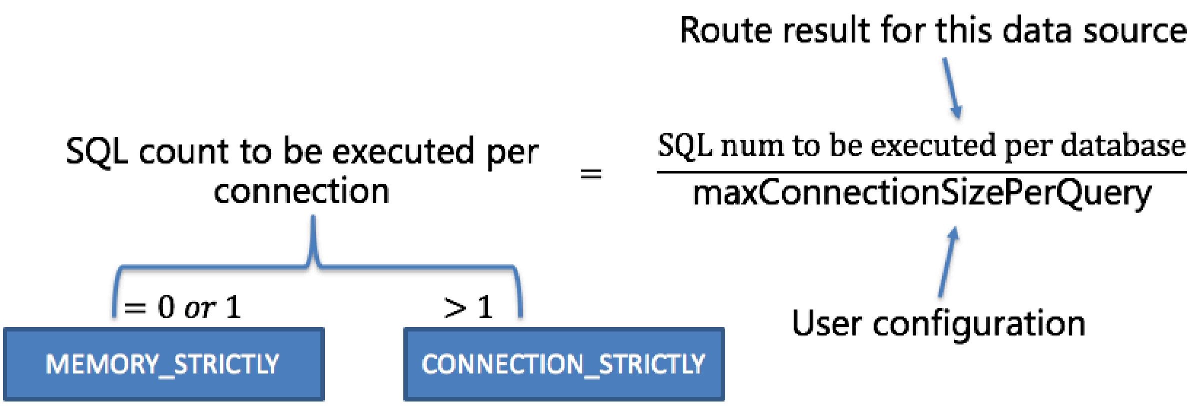

Figure7: Connection mode calculate formula

Within the range that maxConnectionSizePerQuery permits, when the request number that one connection needs to execute is more than 1, meaning current database connection cannot hold the corresponding data result set, it must uses memory merger. On the contrary, when it equals to 1, meaning current database connection can hold the according data result set, it can use stream merger.

Each choice of connection mode aims at each physical database; that is to say, if it is routed to more than one databases, the connection mode of each database may mix with each other and not be the same in one query.

Users can use the route group result acquired from the last step to create the execution unit. When the data source uses technologies, such as database connection pool, to control database connection number, there is some chance for deadlock, if it has not dealt with concurrency properly. As multiple requests waiting for each other to release database connection resources, it will generate hunger wait and cause the crossing deadlock problem.

For example, suppose one query needs to acquire two database connections from a data source and apply them in two table sharding queries routed to one database. It is possible that Query A has already acquired a database connection from that data source and waits to acquire another connection; but in the same time, Query B has also finished it and waits. If the maximum connection number that the connection pool permits is 2, those two query requests will wait forever. The following picture has illustrated the deadlock situation:

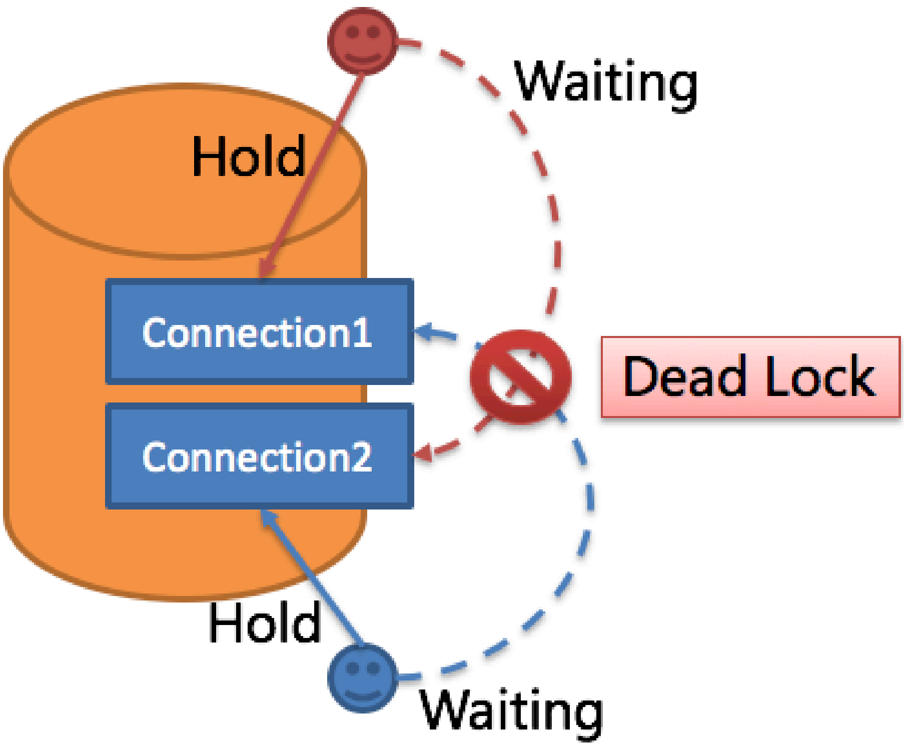

Figure8: Dead lock

To avoid the deadlock, ShardingSphere will go through synchronous processing when acquiring database connection. When creating execution units, it acquires all the database connections that this SQL requires for once with atomic method and reduces the possibility of acquiring only part of the resources. Due to the high operation frequency, locking the connection each time when acquiring it can decrease ShardingSphere' s concurrency. Therefore, it has improved two aspects here:

1. Avoid the setting that locking only takes one database connection each time. Because under this kind of circumstance, two requests waiting for each other will not happen, so there is no need for locking. Most OLTP operations use sharding keys to route to the only data node, which will make the system in a totally unlocked state, thereby improve the concurrency efficiency further. In addition to routing to a single shard, readwrite-splitting also belongs to this category.

2. Only aim at MEMORY_STRICTLY mode to lock resources. 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

Applied in actually SQL execution, this phrase can be divided into two steps: group execution and merger result generation.

Group execution can distribute execution unit groups generated in preparation phrase to the underlying concurrency engine and send events according to each key steps during the execution process, such as starting, successful and failed execution events. Execution engine only focuses on message sending rather than subscribers of the event. Other ShardingSphere modules, such as distributed transactions, invoked chain tracing and so on, will subscribe focusing events and do corresponding operations. Through the connection mode acquired in preparation phrase, ShardingSphere will generate memory merger result set or stream merger result set, and transfer it to the result merger engine for the next step.

The overall structure of execution engine is shown as the following picture:

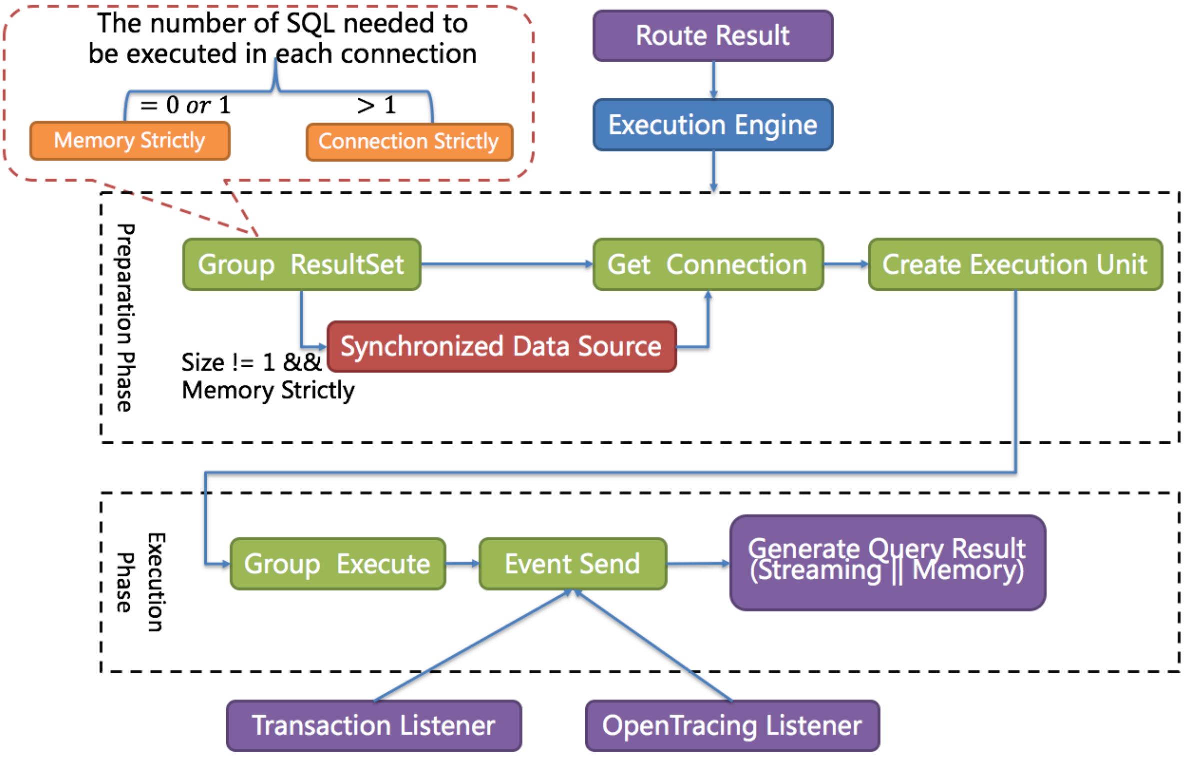

Figure9: Execute engine architecture

9.2.11 Merger Engine

Result merger refers to merging multi-data result set acquired from all the data nodes as one result set and returning it to the request end rightly.

In function, the result merger supported by ShardingSphere can be divided into five kinds, iteration, order-by, group-by, pagination and aggregation, which are in composition relation rather than clash relation. In structure, it can be divided into stream merger, memory merger and decorator merger, among which, stream merger and memory merger clash with each other; decorator merger can be further processed based on stream merger and memory merger.

Since the result set is returned from database line by line instead of being loaded to the memory all at once, the most prior choice of merger method is to follow the database returned result set, for it is able to reduce the memory consumption to a large extend.

Stream merger means, each time, the data acquired from the result set is able to return the single piece of right data line by line.

It is the most suitable one for the method that the database returns original result set. Iteration, order-by, and stream group-by belong to stream merger.

Memory merger needs to iterate all the data in the result set and store it in the memory first. after unified grouping, ordering, aggregation and other computations, it will pack it into data result set, which is visited line by line, and return that result set.

Decorator merger merges and reinforces all the result sets function uniformly. Currently, decorator merger has pagination merger and aggregation merger these two kinds.

Iteration Merger

As the simplest merger method, iteration merger only requires the combination of multiple data result sets into a single-direction chain table. After iterating current data result sets in the chain table, it only needs to move the element of chain table to the next position and iterate the next data result set.

Order-by Merger

Because there is `ORDER BY` statement in SQL, each data result has its own order. So it is enough only to order data value that the result set cursor currently points to, which is equal to sequencing multiple already ordered arrays, and therefore, order-by merger is the most suitable ordering algorithm in this situation.

When merging order inquiries, ShardingSphere will compare current data values in each result set (which is realized by 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 in the top end of the line, renter 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 one of them has already been ordered according to the score, but there is no order between 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 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.

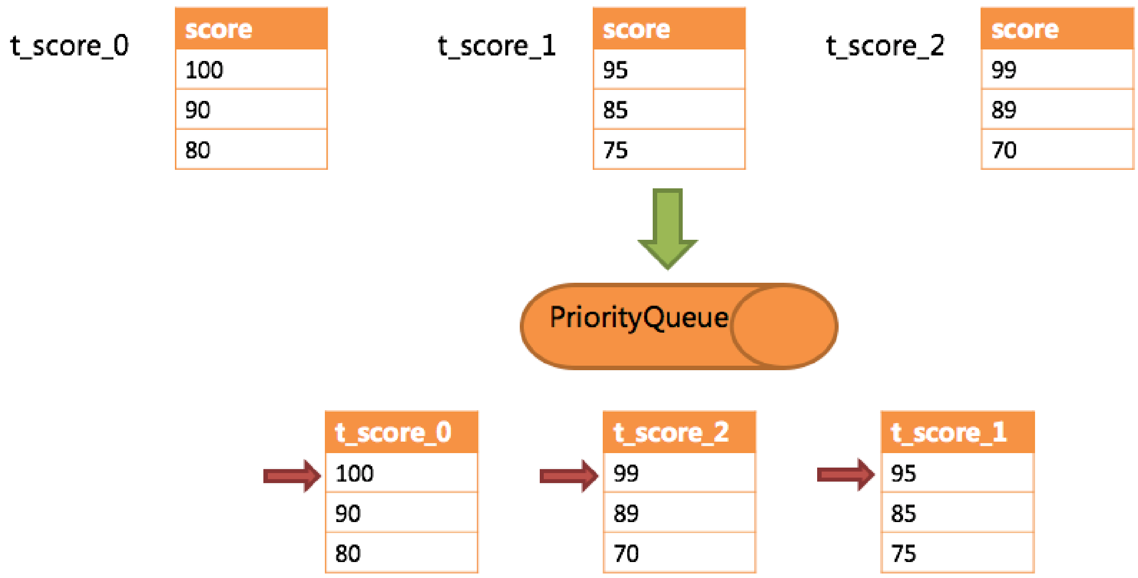

Figure10: Order by merger example 1

This diagram illustrates how the order-by merger works when using next invocation. We can see from the diagram that when using next invocation, 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 client end, the cursor will be moved down and t_score_0 will be put back to 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 current data result set. According to the current value, t_score_0 is at the last of the queue, and in the second place of the queue formerly, the data result set of t_score_2, automatically moves to the first place of the queue.

In the second next operation, t_score_2 in the first position is popped out of the queue. 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 in the same way. If there is no data in the result set, it will not rejoin the queue.

It can be seen that, under the circumstance that data in each result set is ordered while result sets are disordered, ShardingSphere does not need to upload all the data to the memory to order. In the order-by merger method, each next operation only acquires the right piece of data each time, which saves the memory consumption to a large extent.

On the other hand, the order-by merger has maintained the orderliness on horizontal axis and vertical axis of the data result set. Naturally ordered, vertical axis refers to each data result set itself, which is acquired by SQL with `ORDER BY`. Horizontal axis refers to the current value pointed by each data result

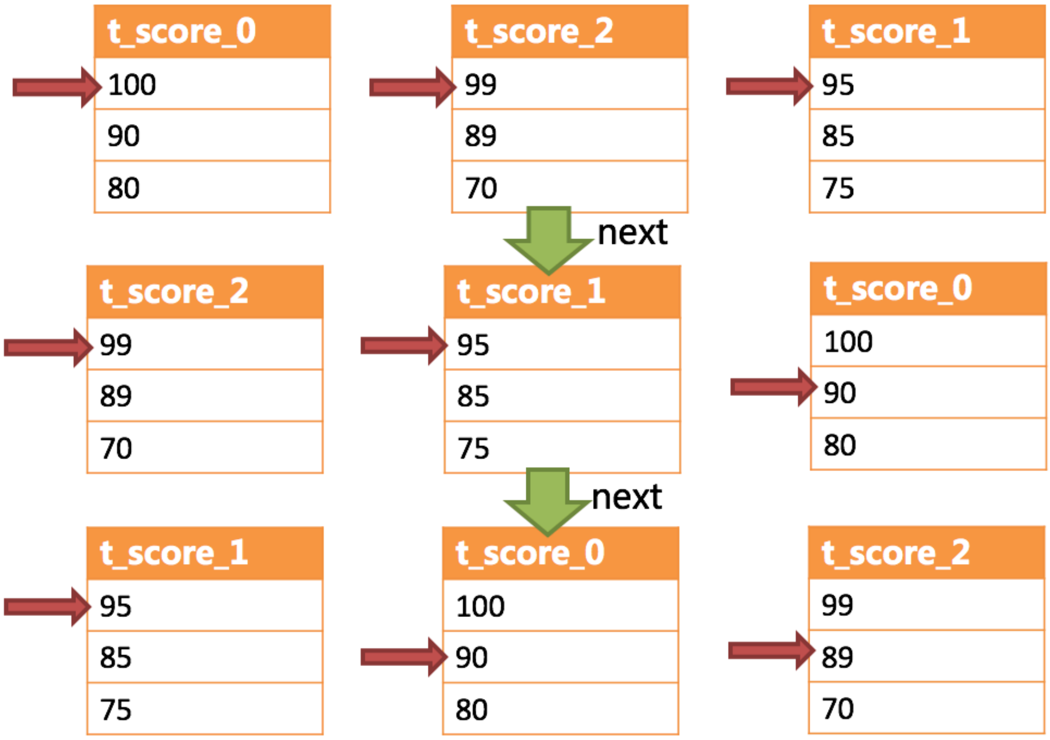

Figure11: Order by merger example 2

set, and its order needs to be maintained by the priority queue.Each time when the current cursor moves down, it requires to put 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

With the most complicated situation, group-by merger can be divided into stream group-by merger and memory group-by merger. Stream group-by merger requires SQL field and order item type (ASC or DESC) to be the same with group-by item. Otherwise, its data accuracy can only be maintained by memory merger.

For instance, if it is sharded by subject, table structure contains examinees' name (to simplify, name repetition is not taken into consideration) and score. The SQL used to acquire each examinee' s total score is as follow:

```
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 to group are all stored in the data value that data result set cursor currently points to, stream group-by merger can be used, as illustrated by the diagram:

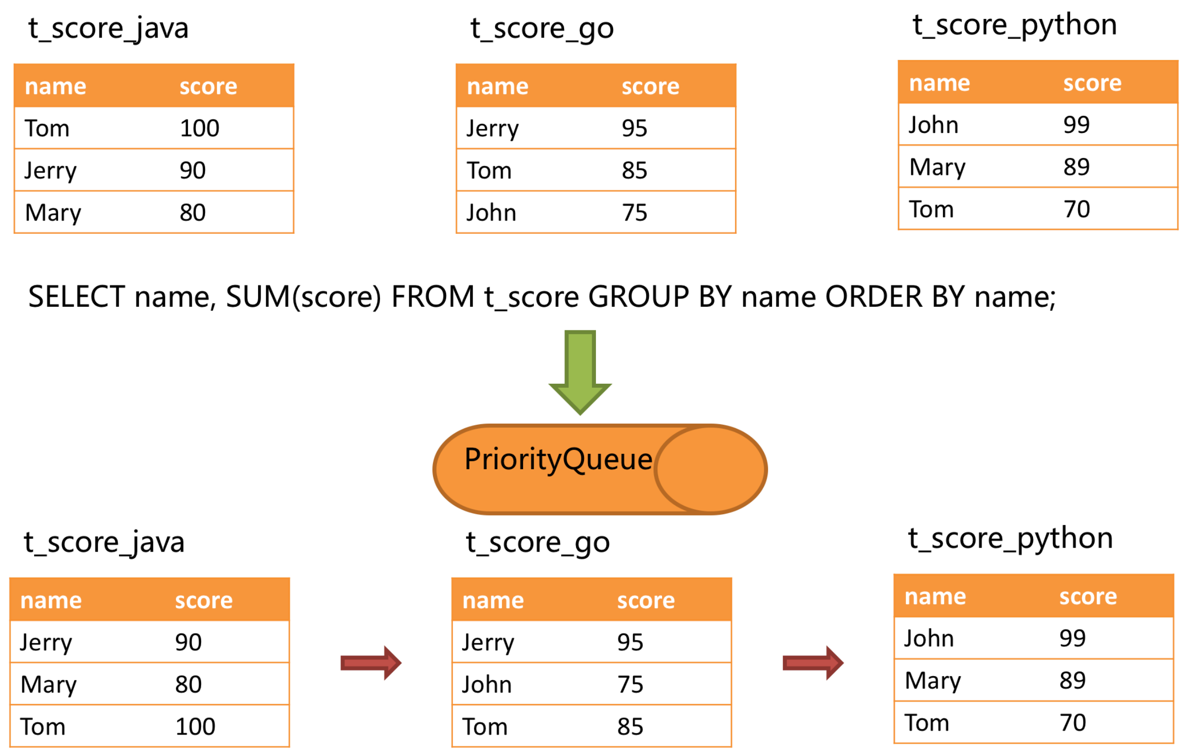

Figure12: Group by merger example 1

The merging logic is similar to that of order-by merger. The following picture shows how stream group-by merger works in next invocation.

We can see from the picture, in the first next invocation, t_score_java in the first position, along with other result set data also having the grouping value of "Jerry", will be popped out of the queue. After

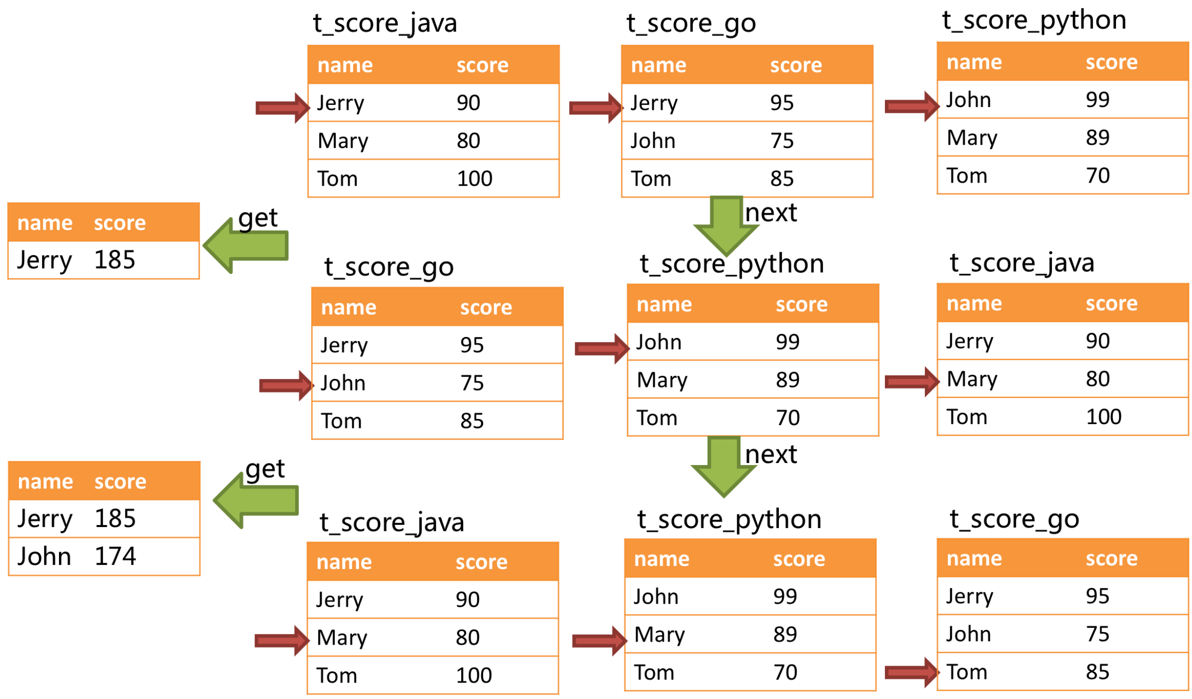

Figure13: Group by merger example 2

acquiring all the students' scores with the name of "Jerry", the accumulation operation will be proceeded. Hence, after the first next invocation is finished, the result set acquired is the sum of Jerry's scores. In the same time, all the cursors in data result sets will be moved down to a different data value next to "Jerry" and rearranged according to 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 points:

1. It will take out all the data with the same group item from multiple data result sets for once.
2. It does the aggregation calculation according to aggregation function type.

For the inconsistency between the group item and the order item, it requires to upload all the data to the memory to group and aggregate, since the relevant data value needed to acquire group information is not continuous, and stream merger is not able to use. 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 diagram ordered by score in the upper half part structure.

When SQL only contains group-by statement, according to different database implementation, its sequencing order may not be the same as the group order. The lack of ordering statement indicates the order is not important in this SQL. Therefore, through SQL optimization re-write, ShardingSphere can automatically add the ordering item same as grouping item, converting it from the memory merger that consumes memory to stream merger.

Aggregation Merger

Whether stream group-by merger or memory group-by merger processes the aggregation function in the same way. Therefore, aggregation merger is an additional merging ability based on what have been introduced above, i.e., the decorator mode. The aggregation function can be categorized into three types, comparison, sum and average.

Comparison aggregation function refers to `MAX` and `MIN`. They need to compare all the result set data and return its maximum or minimum value directly.

Sum aggregation function refers to `SUM` and `COUNT`. They need to sum up all the result set data.

Average aggregation function refers only to `AVG`. It must be calculated through `SUM` and `COUNT` of SQL re-write, which has been mentioned in SQL re-write, so we will state no more here.

Pagination Merger

All the merger types above can be paginated. Pagination is the decorator added on other kinds of mergers. ShardingSphere augments its ability to paginate the data result set through the decorator mode. Pagination merger is responsible for filtering the data unnecessary to acquire.

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 accuracy by rewriting `LIMIT 10000000, 10` to `LIMIT 0, 10000010`. Users can easily have the misconception 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, only memory group-by merger will upload all the data to the memory. Generally speaking, however, SQL used for OLAP grouping, is applied more frequently to massive calculation or small result generation rather than vast result data generation. Except for memory group-by merger, other cases use stream merger to acquire data result set. So ShardingSphere would skip unnecessary data through next method in result set, rather than storing them in the memory.

What's to be noticed, pagination with `LIMIT` is not the best practice actually, because a large amount of data still needs to be transmitted to ShardingSphere's memory space for ordering. `LIMIT` cannot search for data by index, so paginating with ID is a better solution on the premise that the ID continuity can be guaranteed. For example:

```sql
SELECT * FROM t_order WHERE id > 100000 AND id <= 100010 ORDER BY id;
```

Or search the next page through the ID of the last query result, for example:

```sql
SELECT * FROM t_order WHERE id > 10000000 LIMIT 10;
```

The overall structure of merger engine is shown in the following diagram:

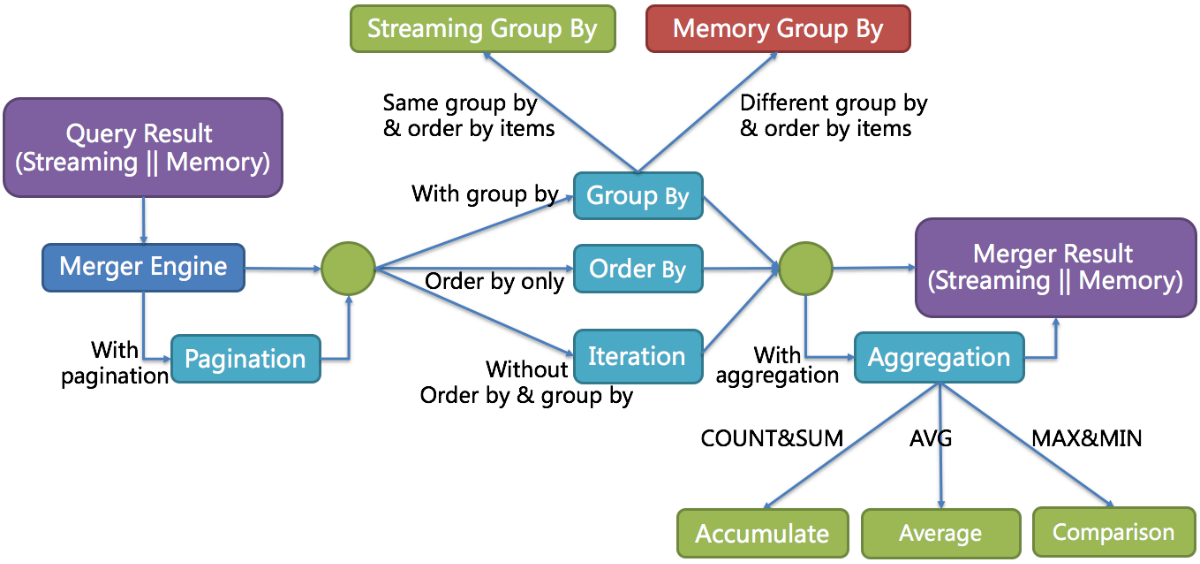

Figure14: Merge Architecture

9.3 Transaction

9.3.1 Navigation

This chapter mainly introduces the principles of the distributed transactions:

- 2PC transaction with XA
- BASE transaction with Seata

9.3.2 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.

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.

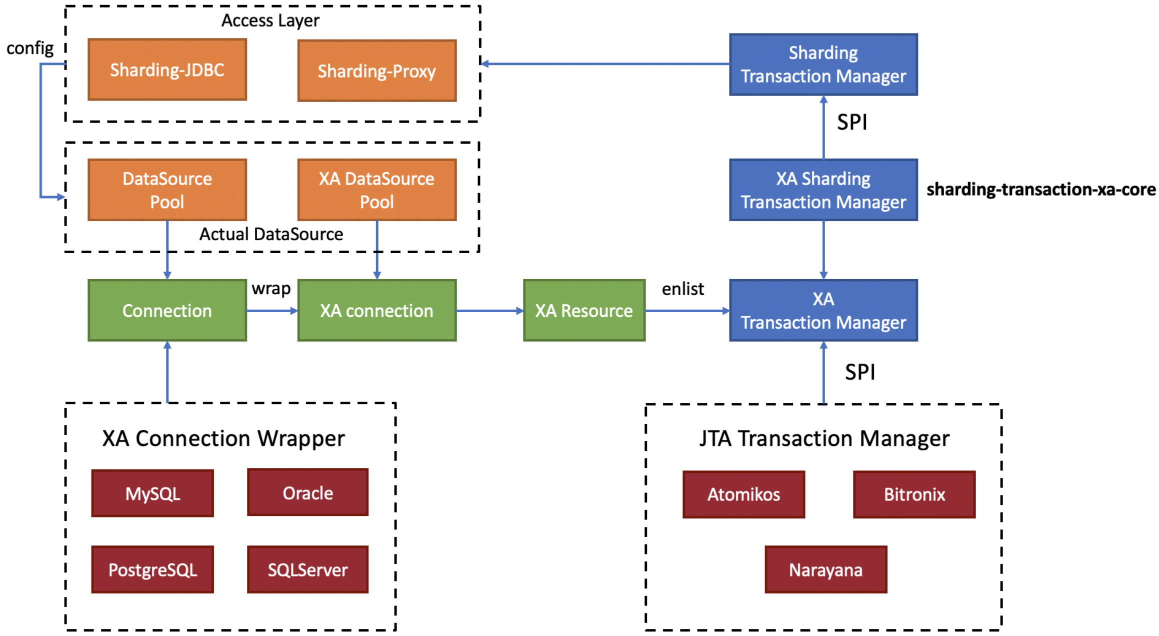

Figure15: Principle of ShardingSphere transaction XA

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 roll-back 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
```

9.3.3 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.

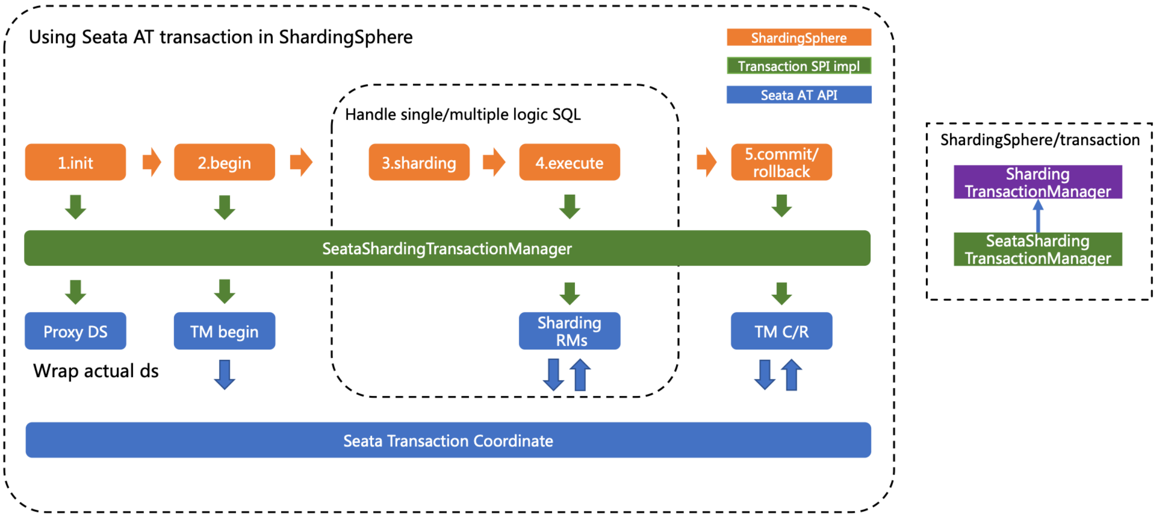

Figure16: Seata BASE transaction

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.

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.

9.4 Scaling

9.4.1 Principle Description

Consider about these challenges of ShardingSphere-Scaling, the solution is: Use two database clusters temporarily, and switch after the scaling is completed.

Advantages:

1. No effect for origin data during scaling.
2. No risk for scaling failure.
3. No limited by sharding strategies.

Disadvantages：

1. Redundant servers during scaling.
2. All data needs to be moved.

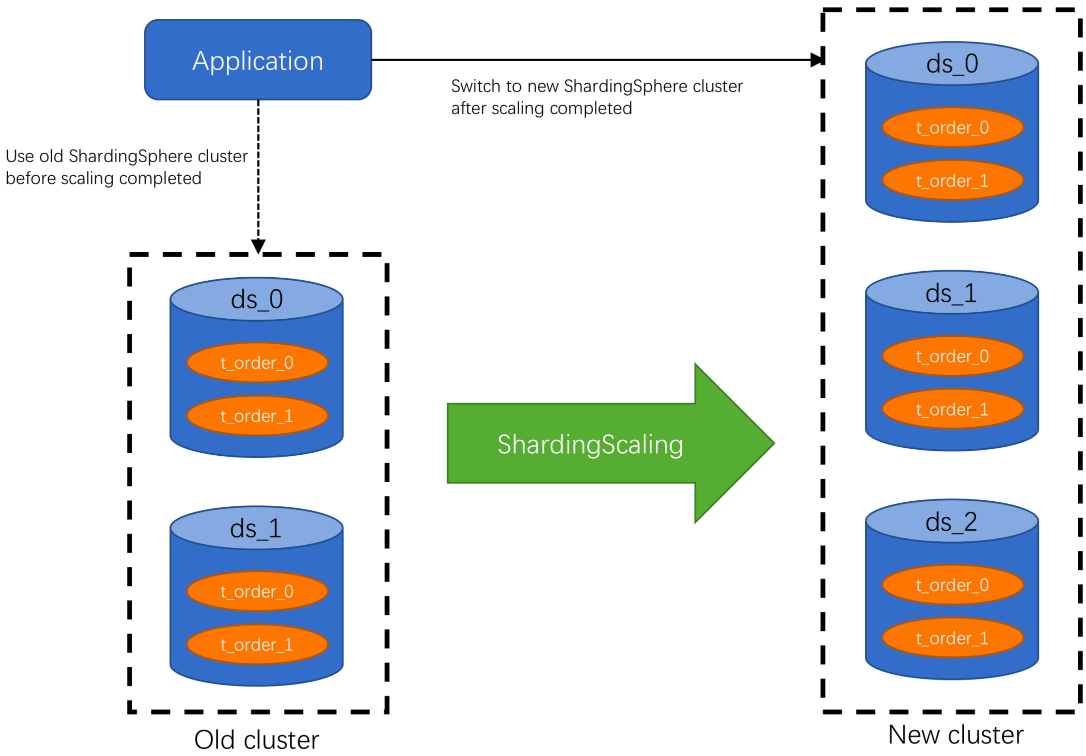

Figure17: Scaling Principle Overview

ShardingSphere-Scaling will analyze the sharding rules and extract information like datasource and data nodes. According the sharding rules, ShardingSphere-Scaling create a scaling job with 4 main phases.

1. Preparing Phase.
2. Inventory Phase.
3. Incremental Phase.
4. Switching Phase.

9.4.2 Phase Description

Preparing Phase

ShardingSphere-Scaling will check the datasource connectivity and permissions, statistic the amount of inventory data, record position of log, shard tasks based on amount of inventory data and the parallelism set by the user.

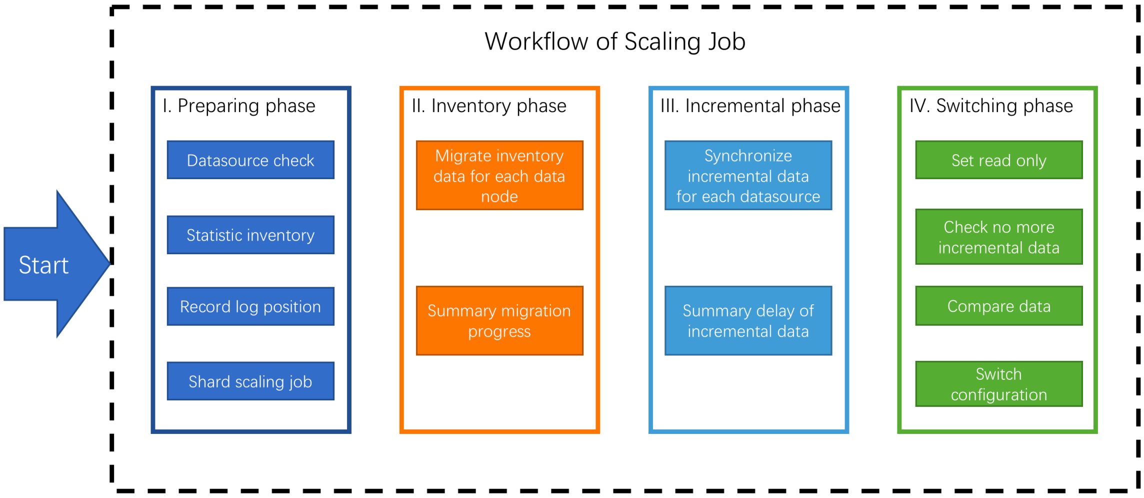

Figure18: Workflow

Inventory Phase

Executing the Inventory data migration tasks sharded in preparing phase. ShardingSphere-Scaling uses JDBC to query inventory data directly from data nodes and write to the new cluster using new rules.

Incremental Phase

The data in data nodes is still changing during the inventory phase, so ShardingSphere-Scaling need to synchronize these incremental data to new data nodes. Different databases have different implementations, but generally implemented by change data capture function based on replication protocols or WAL logs.

- MySQL: subscribe and parse binlog.
- PostgreSQL: official logic replication test_decoding.

These captured incremental data, Apache ShardingSphere also write to the new cluster using new rules.

Switching Phase

In this phase, there may be a temporary read only time, make the data in old data nodes static so that the incremental phase complete fully. The read only time is range seconds to minutes, it depends on the amount of data and the checking data. After finished, Apache ShardingSphere can switch the configuration by register-center and config-center, make application use new sharding rule and new data nodes.

9.5 Encryption

9.5.1 Process Details

Apache ShardingSphere can encrypt the plaintext by parsing and rewriting SQL according to the encryption rule, and store the plaintext (optional) and ciphertext data to the database at the same time. Queries data only extracts the ciphertext data from database and decrypts it, and finally returns the plaintext to user. Apache ShardingSphere transparently process of data encryption, so that users do not need to know to the implementation details of it, use encrypted data just like as regular data. In addition, Apache ShardingSphere can provide a relatively complete set of solutions whether the online business system has been encrypted or the new online business system uses the encryption function.

Overall Architecture

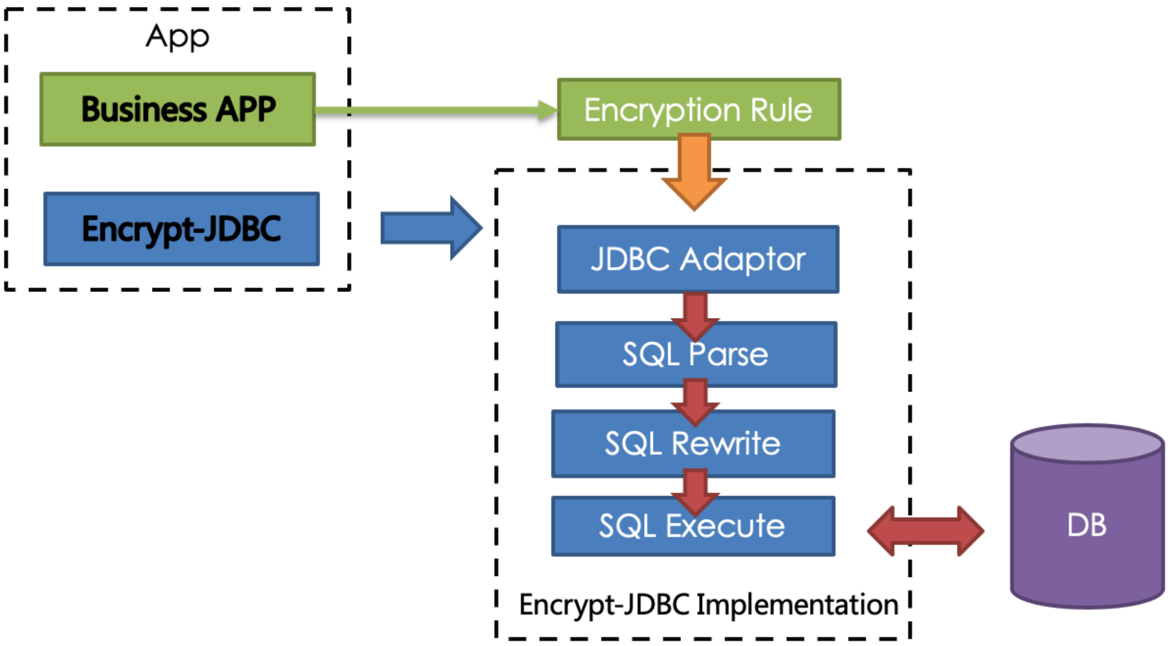

Figure19: 1

Encrypt module intercepts SQL initiated by user, analyzes and understands SQL behavior through the SQL syntax parser. According to the encryption rules passed by the user, find out the fields that need to be encrypted/decrypted and the encryptor/decryptor used to encrypt/decrypt the target fields, and then interact with the underlying database. ShardingSphere will encrypt the plaintext requested by the user and store it in the underlying database; and when the user queries, the ciphertext will be taken out of the database for decryption and returned to the end user. ShardingSphere shields the encryption of data, so that users do not need to perceive the process of parsing SQL, data encryption, and data decryption, just like using ordinary data.

Encryption Rule

Before explaining the whole process in detail, we need to understand the encryption rules and configuration, which is the basis of understanding the whole process. The encryption configuration is mainly divided into four parts: data source configuration, encrypt algorithm configuration, encryption table rule configuration, and query attribute configuration. The details are shown in the following figure:

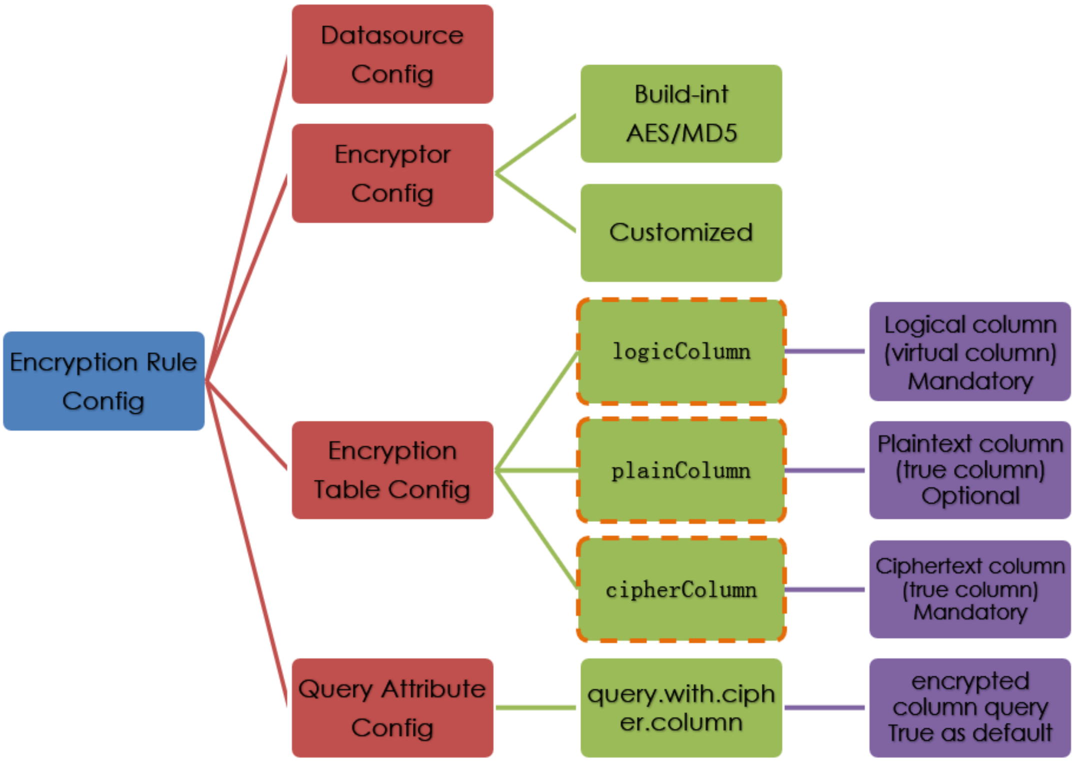

Figure20: 2

Datasource Configuration: The configuration of DataSource.

Encrypt Algorithm Configuration: What kind of encryption strategy to use for encryption and decryption. Currently ShardingSphere has three built-in encryption/decryption strategies: AES, MD5, RC4. Users can also implement a set of encryption/decryption algorithms by implementing the interface provided by Apache ShardingSphere.

Encryption Table Configuration: Show the ShardingSphere data table which column is used to store cipher column data (cipherColumn), which column is used to store plain text data (plainColumn), and which column users want to use for SQL writing (logicColumn)

> How to understand `Which column do users want to use to write SQL (logicColumn)?`
>
> We can understand according to the meaning of Apache ShardingSphere. The ultimate goal of Apache ShardingSphere is to shield the encryption of the underlying data, that is, we do not want users to know how the data is encrypted/decrypted, how to store plaintext data in plainColumn, and ciphertext data in cipherColumn. In other words, we do not even want users to know the existence and use of plainColumn and cipherColumn. Therefore, we need

to provide users with a column in conceptual. This column can be separated from the real column of the underlying database. It can be a real column in the database table or not, so that the user can freely change the plainColumn and The column name of cipherColumn. Or delete plainColumn and choose to never store plain text and only store cipher text. As long as the user's SQL is written according to this logical column, and the correct mapping relationship between logicColumn and plainColumn, cipherColumn is given in the encryption rule.

Why do you do this? The answer is at the end of the article, that is, to enable the online services to seamlessly, transparently, and safely carry out data encryption migration.

Query Attribute configuration: When the plaintext data and ciphertext data are stored in the underlying database table at the same time, this attribute switch is used to decide whether to directly query the plaintext data in the database table to return, or to query the ciphertext data and decrypt it through Apache ShardingSphere to return.

Encryption Process

For example, if there is a table in the database called t_user, there are actually two fields pwd_plain in this table, used to store plain text data, pwd_cipher, used to store cipher text data, and define logicColumn as pwd. Then, when writing SQL, users should write to logicColumn, that is, `INSERT INTO t_user SET pwd = '123'`. Apache ShardingSphere receives the SQL, and through the encryption configuration provided by the user, finds that pwd is a logicColumn, so it decrypt the logical column and its corresponding plaintext data. As can be seen that ** Apache ShardingSphere has carried out the column-sensitive and data-sensitive mapping conversion of the logical column facing the user and the plaintext and ciphertext columns facing the underlying database. As shown below:

This is also the core meaning of Apache ShardingSphere, which is to separate user SQL from the underlying data table structure according to the encryption rules provided by the user, so that the SQL writer by user no longer depends on the actual database table structure. The connection, mapping, and conversion between the user and the underlying database are handled by Apache ShardingSphere. Why should we do this? It is still the same : in order to enable the online business to seamlessly, transparently and safely perform data encryption migration.

In order to make the reader more clearly understand the core processing flow of Apache ShardingSphere, the following picture shows the processing flow and conversion logic when using Apache ShardingSphere to add, delete, modify and check, as shown in the following figure.

9.5.2 Detailed Solution

After understanding the Apache ShardingSphere encryption process, you can combine the encryption configuration and encryption process with the actual scenario. All design and development are to solve the problems encountered in business scenarios. So for the business scenario requirements mentioned earlier, how should ShardingSphere be used to achieve business requirements?

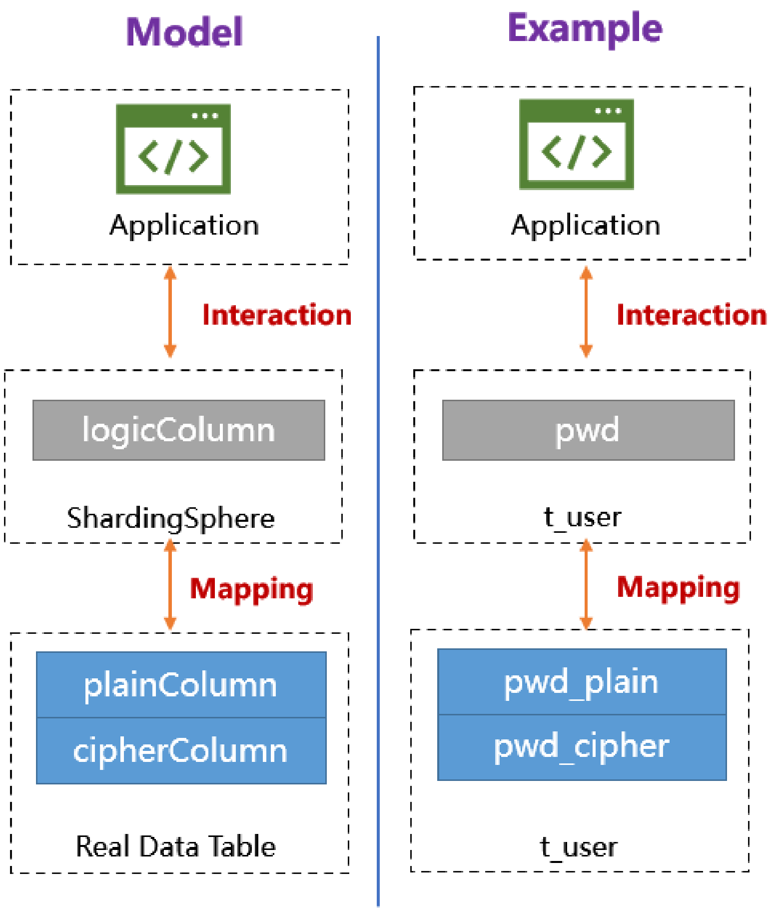
Model
Application
Interaction
logicColumn
ShardingSphere
Mapping
plainColumn
cipherColumn
Real Data Table
Example
Application
Interaction
pwd
t_user
Mapping
pwd_plain
pwd_cipher
t_user

Figure21: 3

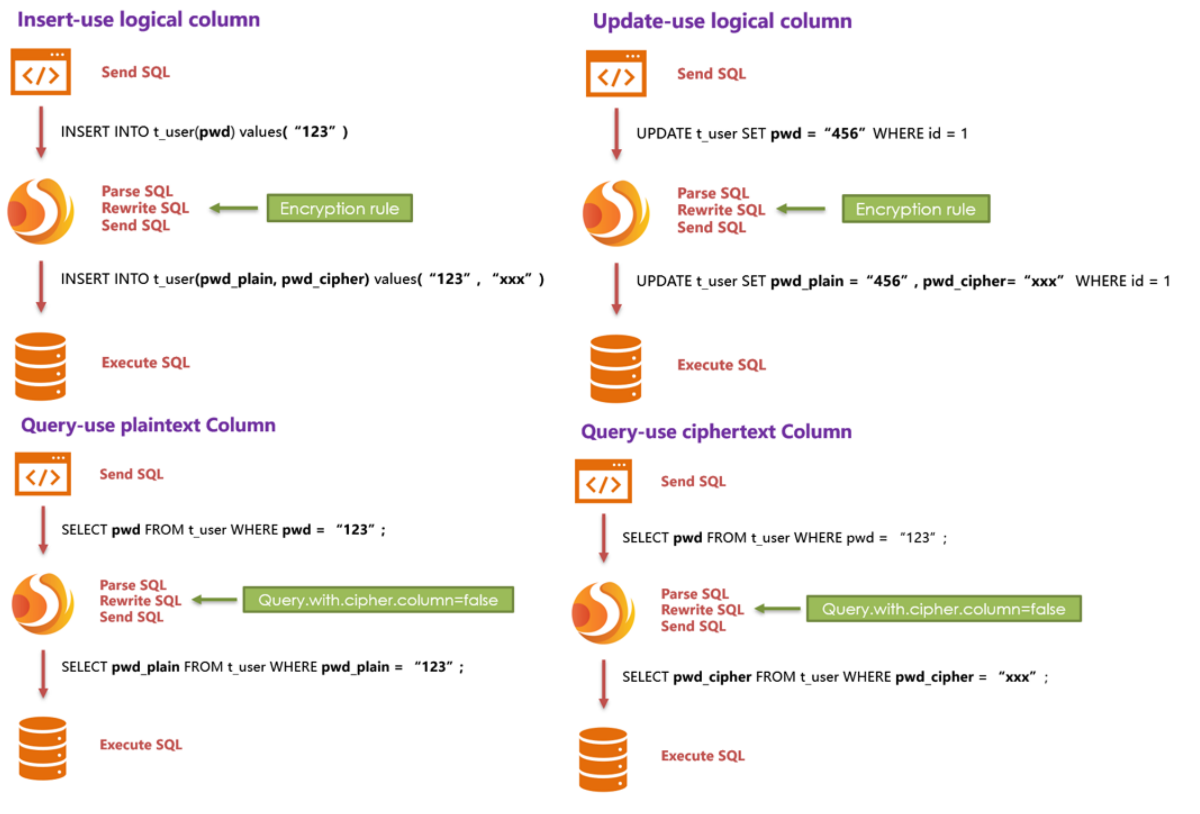

Figure22: 4

New Business

Business scenario analysis: The newly launched business is relatively simple because everything starts from scratch and there is no historical data cleaning problem.

Solution description: After selecting the appropriate encrypt 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). It can also be different **. The recommended configuration is as follows (shown in Yaml format):

```
-!ENCRYPT
  encryptors:
    aes_encryptor:
      type: AES
      props:
        aes-key-value: 123456abc
  tables:
    t_user:
      columns:
        pwd:
          cipherColumn: pwd
          encryptorName: aes_encryptor
```

With this configuration, Apache ShardingSphere only needs to convert logicColumn and cipherColumn.

The underlying data table does not store plain text, only cipher text. This is also a requirement of the security audit part. If users want to store plain text and cipher text together in the database, they just need to add plainColumn configuration. The overall processing flow is shown below:

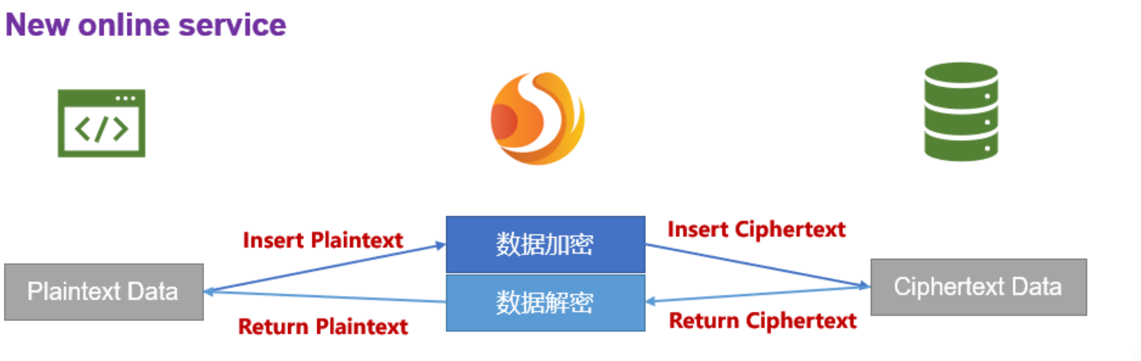

Figure23: 5

Online Business Transformation

Business scenario analysis: As the business is already running online, there must be a large amount of plain text historical data stored in the database. The current challenges are how to enable historical data to be encrypted and cleaned, how to enable incremental data to be encrypted, and how to allow businesses to seamlessly and transparently migrate between the old and new data systems.

Solution description: Before providing a solution, let ' s brainstorm: First, if the old business needs to be desensitized, it must have stored very important and sensitive information. This information has a high gold content and the business is relatively important. If it is broken, the whole team KPI is over. Therefore, it is impossible to suspend business immediately, prohibit writing of new data, encrypt and clean all historical data with an encrypt algorithm, and then deploy the previously reconstructed code online, so that it can encrypt and decrypt online and incremental data. Such a simple and rough way, based on historical experience, will definitely not work.

Then another relatively safe approach is to rebuild a pre-release environment exactly like the production environment, and then encrypt the **Inventory plaintext data** of the production environment through the relevant migration and washing tools and store it in the pre-release environment. The **Increment data** is encrypted by tools such as MySQL replica query and the business party ' s own development, encrypted and stored in the database of the pre-release environment, and then the refactored code can be deployed to the pre-release environment. In this way, the production environment is a set of environment for **modified/queries with plain text as the core**; the pre-release environment is a set of **encrypt/decrypt queries modified with ciphertext as the core**. After comparing for a period of time, the production flow can be cut into the pre-release environment at night. This solution is relatively safe and reliable, but it takes more time, manpower, capital, and costs. It mainly includes: pre-release environment construction, production code rectification, and related auxiliary tool development. Unless there is no way to go, business developers generally go from getting started to giving up.

Business developers must hope: reduce the burden of capital costs, do not modify the business code, and be able to safely and smoothly migrate the system. So, the encryption function module of ShardingSphere was born. 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 achieve different services demand). In addition, 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
  queryWithCipherColumn: false
```

According to the above encryption rules, we need to add a column called pwd_cipher in the t_user table, that is, cipherColumn, 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, that is, the SQL was written for logical columns, so the business code did not need to be changed. Through Apache ShardingSphere, for the incremental data, the plain text will be written to the pwd column, and the plain text will be encrypted and stored in the pwd_cipher column. At this time, because `queryWithCipherColumn` is set to false, for business applications, the plain text column of pwd is still used for query storage, but the cipher text data of the new data is additionally stored on the underlying database table pwd_cipher. The processing flow is shown below:

When the newly added data is inserted, it is encrypted as ciphertext data through Apache ShardingSphere and stored in the cipherColumn. Now it is necessary to process historical plaintext inventory data. **As Apache ShardingSphere currently does not provide the corresponding migration and washing tools, the business party needs to encrypt and store the plain text data in pwd to pwd_cipher.**

2. During system migration

The incremental data has been stored by Apache ShardingSphere in the ciphertext column and the plaintext is stored in the plaintext column; after the historical data is encrypted and cleaned by the business party itself, the ciphertext is also stored in the ciphertext column. That is to say, the plaintext and the ciphertext are stored in the current database. Since the `queryWithCipherColumn = false` in the configuration item, the ciphertext has never been used. Now we need to set the `queryWith-`

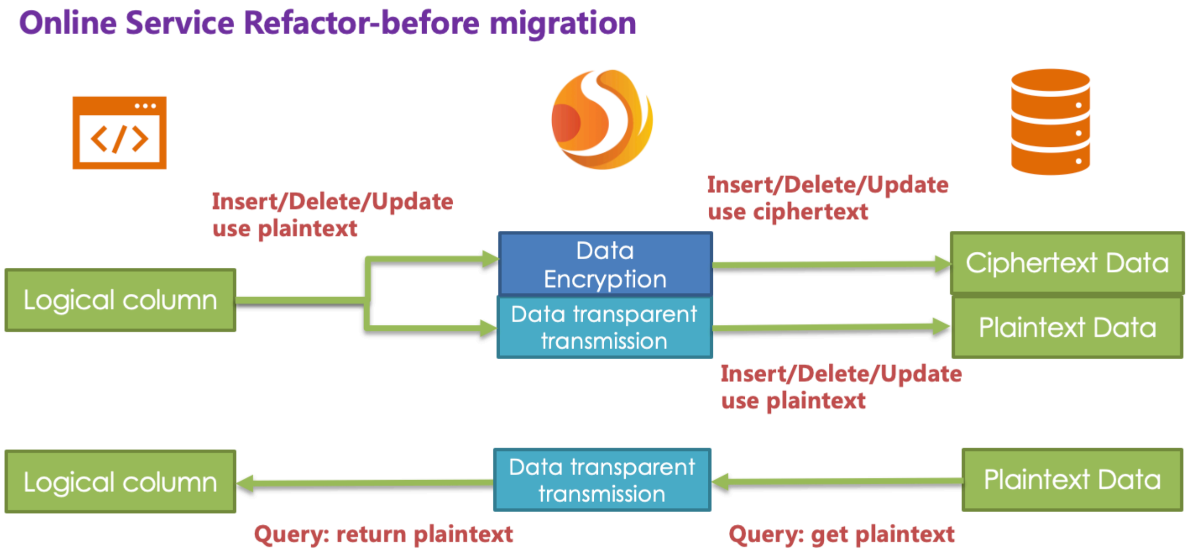

Figure24: 6

`CipherColumn` in the encryption configuration to true in order for the system to cut the ciphertext data for query. After restarting the system, we found that the system business is normal, but Apache ShardingSphere has started to extract the ciphertext data from the database, decrypt it and return it to the user; and for the user' s insert, delete and update requirements, the original data will still be stored The plaintext column, the encrypted ciphertext data is stored in the ciphertext column.

Although the business system extracts the data in the ciphertext column and returns it after decryption; however, it will still save a copy of the original data to the plaintext column during storage. Why? The answer is: in order to be able to roll back the system. **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 just set `queryWithCipherColumn = false`, Apache ShardingSphere will restore, that is, start using plainColumn to query again. The processing flow is shown in the following figure:

3. After system migration

Due to the requirements of the security audit department, it is generally impossible for the business system to keep the plaintext and ciphertext columns of the database permanently synchronized. We need to delete the plaintext data after the system is stable. That is, we need to delete plainColumn (ie pwd) after system migration. The problem is that now the business code is written for pwd SQL, delete the pwd in the underlying data table stored in plain text, and use pwd_cipher to decrypt to get the original data, does that mean that the business side needs to rectify all SQL, thus Do not use the pwd column that is about to be deleted? Remember the core meaning of our encrypt module?

> This is also the core meaning of encrypt 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.

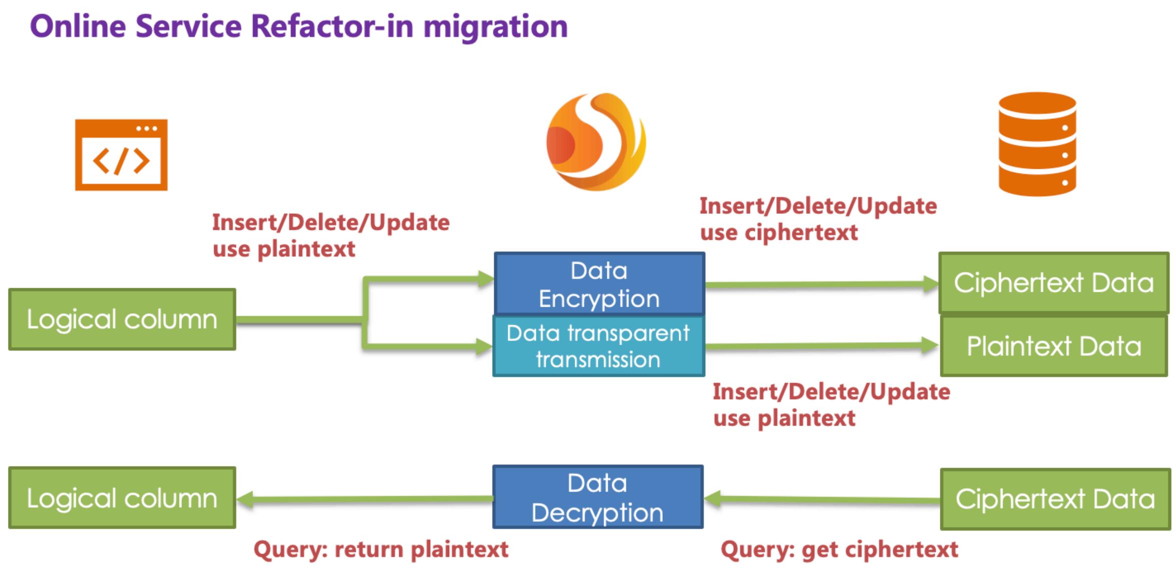

Figure25: 7

Yes, because of the existence of logicColumn, 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 migration is:

```
-!ENCRYPT
  encryptors:
    aes_encryptor:
      type: AES
      props:
        aes-key-value: 123456abc
  tables:
    t_user:
      columns:
        pwd: # pwd 与 pwd_cipher 的转换映射
          cipherColumn: pwd_cipher
          encryptorName: aes_encryptor
```

The processing flow is as follows:

So far, the online service encryption and rectification solutions have all been demonstrated. We provide Java, YAML, Spring Boot Starter, Spring Namespace multiple ways for users to choose to use, and strive to fulfill business requirements. The solution has been continuously launched on JD Digits, providing internal basic service support.

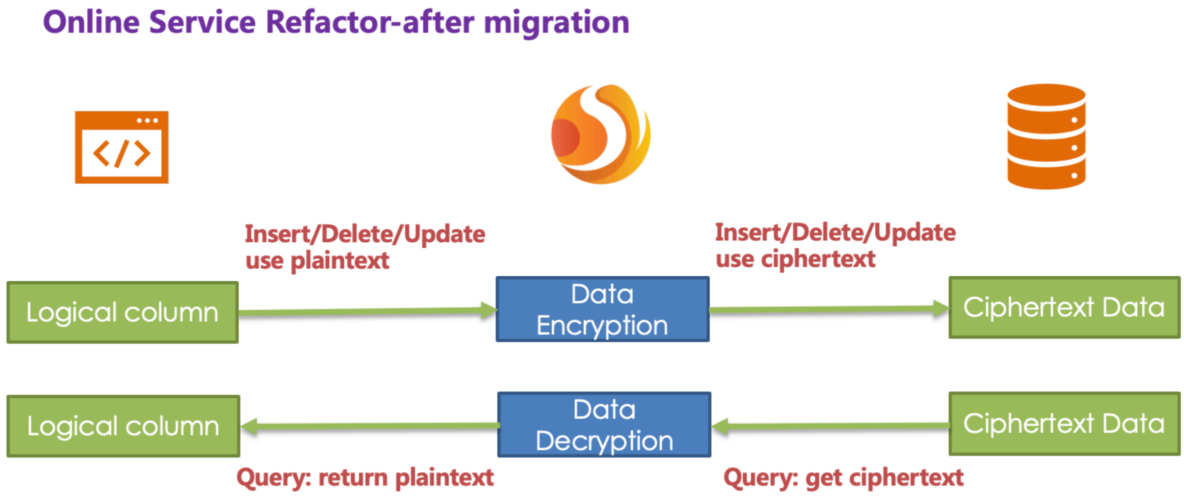

Figure26: 8

9.5.3 The advantages of Middleware encryption service

1. Transparent data encryption process, users do not need to pay attention to the implementation details of encryption.
2. Provide a variety of built-in, third-party (AKS) encryption strategies, users only need to modify the configuration to use.
3. Provides a encryption strategy API interface, users can implement the interface to use a custom encryption strategy for data encryption.
4. Support switching different encryption strategies.
5. For online services, it is possible to store plaintext data and ciphertext data synchronously, and decide whether to use plaintext or ciphertext columns for query through configuration. Without changing the business query SQL, the on-line system can safely and transparently migrate data before and after encryption.

9.5.4 Solution

Apache ShardingSphere has provided two data encryption solutions, corresponding to two ShardingSphere encryption and decryption interfaces, i.e., `EncryptAlgorithm` and `QueryAssistedEncryptAlgorithm`.

On the one hand, Apache ShardingSphere has provided internal encryption and decryption implementations for users, which can be used by them only after configuration. On the other hand, to satisfy users' requirements for different scenarios, we have also opened relevant encryption and decryption interfaces, according to which, users can provide specific implementation types. Then, after simple configurations, Apache ShardingSphere can use encryption and decryption solutions defined by users themselves to desensitize data.

EncryptAlgorithm

The solution has provided two methods `encrypt()` and `decrypt()` to encrypt/decrypt data for encryption.

When users `INSERT`, `DELETE` and `UPDATE`, 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()` reversely and return them to users at last.

Currently, Apache ShardingSphere has provided three types of implementations for this kind of encrypt solution, MD5 (irreversible), AES (reversible) and RC4 (reversible), which can be used after configuration.

QueryAssistedEncryptAlgorithm

Compared with the first encrypt scheme, this one is more secure and complex. Its concept is: even the same data, two same user passwords for example, should not be stored as the same desensitized form in the database. It can help to protect user information and avoid credential stuffing.

This scheme provides three functions to implement, `encrypt()`, `decrypt()` and `queryAssistedEncrypt()`. In `encrypt()` phase, users can set some variable, timestamp for example, and encrypt a combination of original data + variable. This method can make sure the encrypted data of the same original data are different, due to the existence of variables. In `decrypt()` phase, users can use variable data to decrypt according to the encryption algorithms set formerly.

Though this method can indeed increase data security, another problem can appear with it: as the same data is stored in the database in different content, users may not be able to find out all the same original data with equivalent query (`SELECT FROM table WHERE encryptedColumnn = ?`) according to this encryption column. Because of it, we have brought out assistant query column, which is generated by `queryAssistedEncrypt()`. Different from `decrypt()`, this method uses another way to encrypt the original data; but for the same original data, it can generate consistent encryption data. Users can store data processed by `queryAssistedEncrypt()` to assist the query of original data. So there may be one more assistant query column in the table.

`queryAssistedEncrypt()` and `encrypt()` can generate and store different encryption data; `decrypt()` is reversible and `queryAssistedEncrypt()` is irreversible. So when querying the original data, we will parse, rewrite and route SQL automatically. We will also use assistant query column to do `WHERE` queries and use `decrypt()` to decrypt `encrypt()` data and return them to users. All these can not be felt by users.

For now, ShardingSphere has abstracted the concept to be an interface for users to develop rather than providing accurate implementation for this kind of encrypt solution. ShardingSphere will use the accurate implementation of this solution provided by users to desensitize data.

9.6 Shadow

9.6.1 Overall Architecture

Apache ShardingSphere makes shadow judgments on incoming SQL by parsing SQL, according to the shadow rules set by the user in the configuration file, route to production DB or shadow DB.

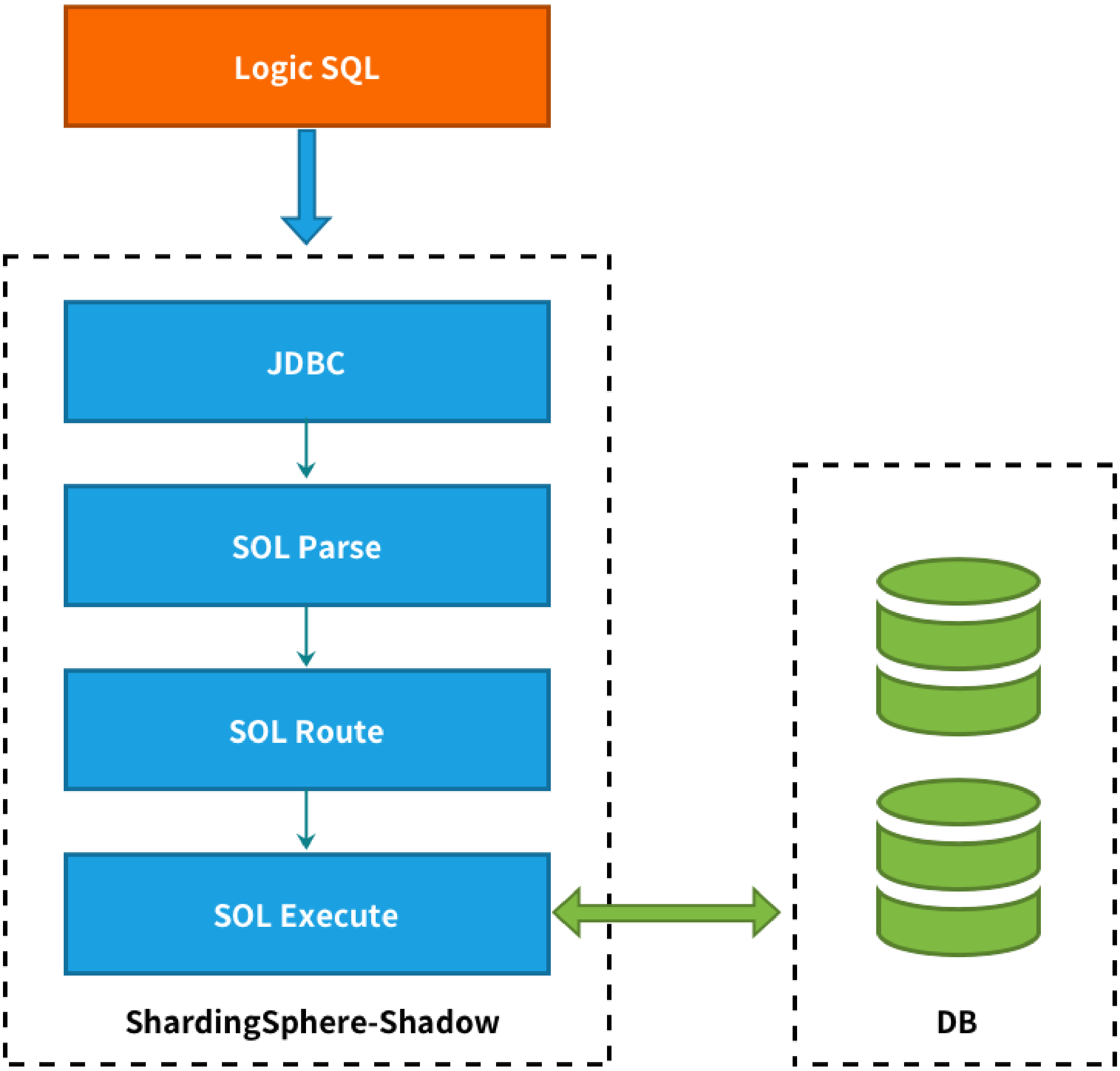

Figure27: Execute Process

9.6.2 Shadow Rule

Shadow rules include shadow data source mapping, shadow tables, and shadow algorithms.

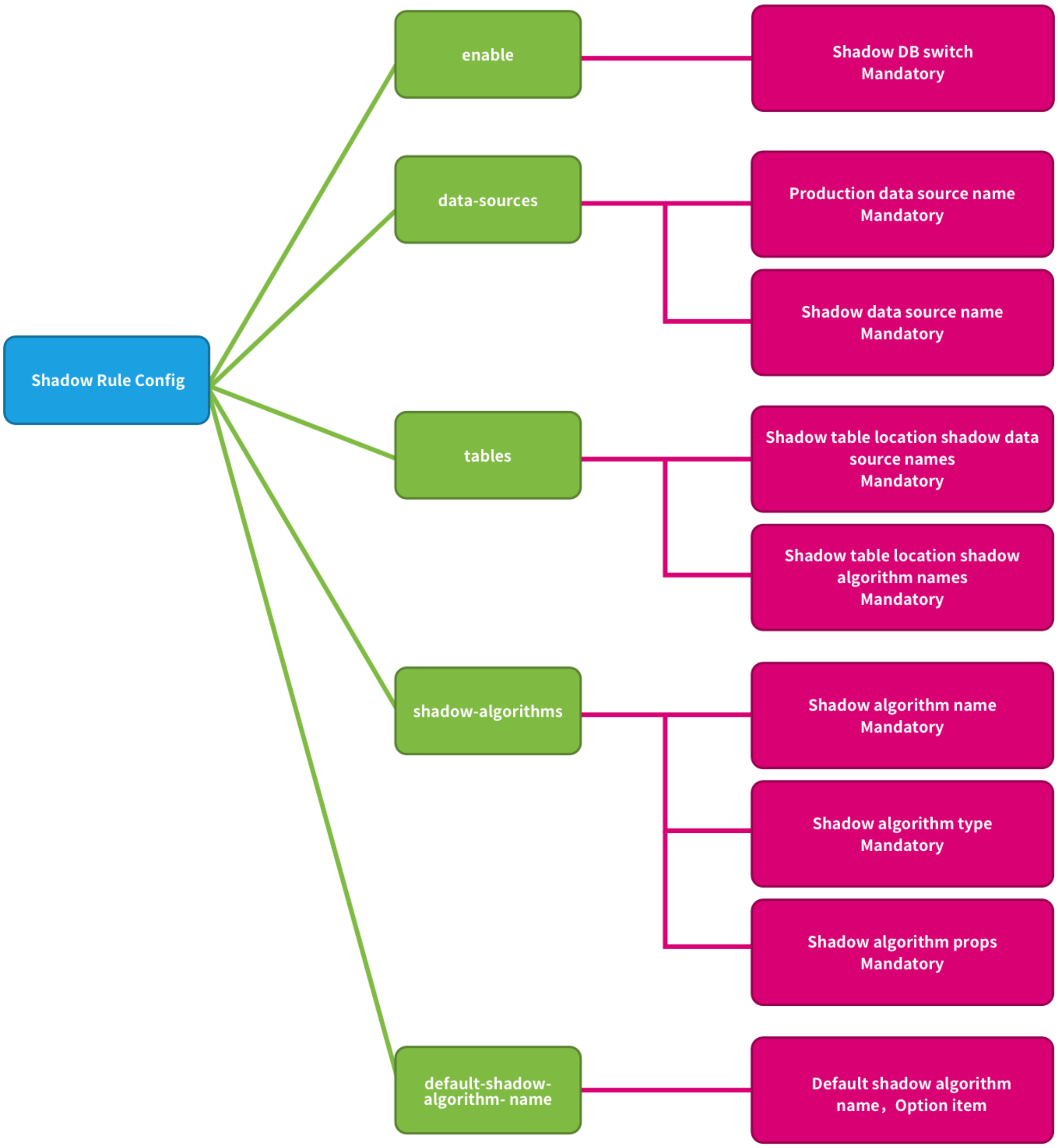

Figure28: Shadow Rule

enable: Shadow DB switch. Optional value `true/false`, default value is `false`.

data-sources: Production data source name and shadow data source name mappings.

tables: Shadow tables related to stress testing. Shadow tables must exist in the specified shadow DB, and the shadow algorithm needs to be specified.

shadow-algorithms: SQL routing shadow algorithm.

default-shadow-algorithm-name: Default shadow algorithm. Optional item, the default matching algorithm for tables that not configured with the shadow algorithm.

9.6.3 Routing Process

Take the INSERT statement as an example. When writing data Apache ShardingSphere will parse the SQL, and then construct a routing chain according to the rules in the configuration file.

In the current version of the function, the shadow function is the last execution unit in the routing chain, that is, if there are other rules that require routing, such as sharding, Apache ShardingSphere will first route to a certain database according to the sharding rules, and then perform the shadow routing decision process.

It determined that the execution of SQL satisfies the configuration of the shadow rule, the data routed to the corresponding shadow database, and the production data remains unchanged.

9.6.4 Shadow Judgment Process

When the shadow DB switch turned on, shadow judgment will be made on the executed SQL statements.

Shadow judgment supports two types of algorithms, users can choose one or combine them according to actual business needs.

DML Statement

Support two type shadow algorithms.

The shadow judgment first judges whether there is an intersection between SQL related tables and configured shadow tables.

If there is an intersection, determine the shadow algorithm associated with the shadow table of the intersection in turn, and any one of them was successful. SQL statement executed shadow DB.

If shadow tables have no intersection, or shadow algorithms are unsuccessful, SQL statement executed production DB.

DDL Statement

Only support note shadow algorithm.

In the pressure testing scenarios, DDL statements are not need tested generally. It is mainly used when initializing or modifying the shadow table in the shadow DB.

The shadow judgment first judges whether the executed SQL contains notes.

If contains notes, determine the note shadow algorithms in the shadow rule in turn, and any one of them was successful. SQL statement executed shadow DB.

The executed SQL does not contain notes, or shadow algorithms are unsuccessful, SQL statement executed production DB.

9.6.5 Shadow Algorithm

Shadow algorithm details, please refer to List of built-in shadow algorithms

9.6.6 Use Example

Scenario

Assume that the e-commerce website wants to perform pressure testing on the order business,

the pressure testing related table `t_order` is a shadow table, the production data executed to the `ds` production DB, and the pressure testing data executed to the database `ds_shadow` shadow DB.

Shadow DB configuration

The shadow configuration for example(YAML):

```yaml
enable: true
  data-sources:
    shadow-data-source:
      source-data-source-name: ds
      shadow-data-source-name: ds-shadow
tables:
  t_order:
    data-source-names: shadow-data-source
    shadow-algorithm-names:
      - simple-hint-algorithm
      - user-id-value-match-algorithm
shadow-algorithms:
  simple-hint-algorithm:
    type: SIMPLE_HINT
    props:
      shadow: true
      foo: bar
  user-id-value-match-algorithm:
    type: VALUE_MATCH
    props:
      operation: insert
      column: user_id
      value: 0

props:
  sql-comment-parse-enabled: true
```

Note: If you use the annotation shadow algorithm, the parse SQL comment configuration item `sql-comment-parse-enabled: true` need to be turned on. turned off by default. please refer to Configuration Props

Shadow DB environment

- Create the shadow DB ds_shadow.
- Create shadow tables, tables structure must be consistent with the production environment. Assume that the t_order table created in the shadow DB. Create table statement need to add SQL note /*shadow:true,foo:bar,.. .*/.

```
CREATE TABLE t_order (order_id INT(11) primary key, user_id int(11) not null, ...)
/*shadow:true,foo:bar,...*/
```

Execute to the shadow DB.

Shadow algorithm example

1. Column shadow algorithm example

Assume that the t_order table contains a list of user_id to store the order user ID. The data of the order created by the user whose user ID is 0 executed to shadow DB, other data executed to production DB.

```
INSERT INTO t_order (order_id, user_id, ...) VALUES (xxx..., 0, ...)
```

No need to modify any SQL or code, only need to control the data of the testing to realize the pressure testing.

Column Shadow algorithm configuration (YAML):

```
shadow-algorithms:
  user-id-value-match-algorithm:
    type: VALUE_MATCH
    props:
      operation: insert
      column: user_id
      value: 0
```

Note: When the shadow table uses the column shadow algorithm, the same type of shadow operation (INSERT, UPDATE, DELETE, SELECT) currently only supports a single column.

2. Hint shadow algorithm example

Assume that the t_order table does not contain columns that can matching. Executed SQL statement need to add SQL note /*shadow:true,foo:bar,.. .*/

```
SELECT * FROM t_order WHERE order_id = xxx /*shadow:true,foo:bar,...*/
```

SQL executed to shadow DB, other data executed to production DB.

Note Shadow algorithm configuration (YAML):

```
shadow-algorithms:
  simple-hint-algorithm:
    type: SIMPLE_HINT
    props:
      shadow: true
      foo: bar
```

3. Hybrid two shadow algorithm example

Assume that the pressure testing of the `t_order` gauge needs to cover the above two scenarios.

```
INSERT INTO t_order (order_id, user_id, ...) VALUES (xxx..., 0, ...);

SELECT * FROM t_order WHERE order_id = xxx /*shadow:true,foo:bar,...*/;
```

Both will be executed to shadow DB, other data executed to production DB.

2 type of shadow algorithm example (YAML):

```
shadow-algorithms:
  user-id-value-match-algorithm:
    type: VALUE_MATCH
    props:
      operation: insert
      column: user_id
      value: 0
  simple-hint-algorithm:
    type: SIMPLE_HINT
    props:
      shadow: true
      foo: bar
```

4. Default shadow algorithm example

Assume that the column shadow algorithm used for the `t_order`, all other shadow tables need to use the note shadow algorithm.

```
INSERT INTO t_order (order_id, user_id, ...) VALUES (xxx..., 0, ...);

INSERT INTO t_xxx_1 (order_item_id, order_id, ...) VALUES (xxx..., xxx..., ...) /
*shadow:true,foo:bar,...*/;

SELECT * FROM t_xxx_2 WHERE order_id = xxx /*shadow:true,foo:bar,...*/;

SELECT * FROM t_xxx_3 WHERE order_id = xxx /*shadow:true,foo:bar,...*/;
```

Both will be executed to shadow DB, other data executed to production DB.

Default shadow algorithm configuration (YAML):

```yaml
enable: true
  data-sources:
    shadow-data-source:
      source-data-source-name: ds
      shadow-data-source-name: ds-shadow
tables:
  t_order:
    data-source-names: shadow-data-source
    shadow-algorithm-names:
      - simple-hint-algorithm
      - user-id-value-match-algorithm
default-shadow-algorithm-name: simple-note-algorithm
shadow-algorithms:
  simple-hint-algorithm:
    type: SIMPLE_HINT
    props:
      shadow: true
      foo: bar
  user-id-value-match-algorithm:
    type: VALUE_MATCH
    props:
      operation: insert
      column: user_id
      value: 0

props:
  sql-comment-parse-enabled: true
```

Note: The default shadow algorithm only supports note shadow algorithm.

9.7 Test

Apache ShardingSphere provides test engines for integration, module and performance.

9.7.1 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.

9.7.2 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.

9.7.3 Performance Test

Provide multiple performance test methods, includes Sysbench, JMH or TPCC and so on.

9.7.4 Integration Test

The SQL parsing unit test covers both SQL placeholder and literal dimension. Integration test can be further divided into two dimensions of strategy and JDBC; the former one includes strategies as Sharding, table Sharding, database Sharding, and readwrite-splitting while the latter one includes `Statement` and `PreparedStatement`.

Therefore, one SQL can drive 5 kinds of database parsing * 2 kinds of parameter transmission modes + 5 kinds of databases * 5 kinds of Sharding strategies * 2 kinds of JDBC operation modes = 60 test cases, to enable ShardingSphere to achieve the pursuit of high quality.

Process

The `Parameterized` in JUnit will collect all test data, and pass to test method to assert one by one. The process of handling test data is just like a leaking hourglass:

Configuration

- environment type
  - /shardingsphere-integration-test-suite/src/test/resources/env-native.properties
  - /shardingsphere-integration-test-suite/src/test/resources/env/SQL-TYPE/dataset.xml
  - /shardingsphere-integration-test-suite/src/test/resources/env/SQL-TYPE/schema.xml
- test case type
  - /shardingsphere-integration-test-suite/src/test/resources/cases/SQL-TYPE/SQL-TYPE-integration-test-cases.xml
  - /shardingsphere-integration-test-suite/src/test/resources/cases/SQL-TYPE/dataset/FEATURE-TYPE/*.xml
- sql-case
  - /sharding-sql-test/src/main/resources/sql/sharding/SQL-TYPE/*.xml

Environment Configuration

Integration test depends on existed database environment, developer need to setup the configuration file for corresponding database to test:

Firstly, setup configuration file `/shardingsphere-integration-test-suite/src/test/resources/env-native.properties`, for example:

```
# the switch for PK, concurrent, column index testing and so on
it.run.additional.cases=false

# test scenarios, could define multiple rules
it.scenarios=db,tbl,dbtbl_with_replica_query,replica_query

# database type, could define multiple databases(H2,MySQL,Oracle,SQLServer,
PostgreSQL)
it.databases=MySQL,PostgreSQL

# MySQL configuration
it.mysql.host=127.0.0.1
it.mysql.port=13306
it.mysql.username=root
it.mysql.password=root

## PostgreSQL configuration
it.postgresql.host=db.psql
it.postgresql.port=5432
it.postgresql.username=postgres
it.postgresql.password=postgres

## SQLServer configuration
it.sqlserver.host=db.mssql
it.sqlserver.port=1433
it.sqlserver.username=sa
it.sqlserver.password=Jdbc1234

## Oracle configuration
it.oracle.host=db.oracle
it.oracle.port=1521
it.oracle.username=jdbc
it.oracle.password=jdbc
```

Secondly, setup configuration file `/shardingsphere-integration-test-suite/src/test/resources/env/SQL-TYPE/dataset.xml`. Developer can set up metadata and expected data to start the data initialization in `dataset.xml`. For example:

```
<dataset>
    <metadata data-nodes="tbl.t_order_${0..9}">
        <column name="order_id" type="numeric" />
```

```
        <column name="user_id" type="numeric" />
        <column name="status" type="varchar" />
    </metadata>
    <row data-node="tbl.t_order_0" values="1000, 10, init" />
    <row data-node="tbl.t_order_1" values="1001, 10, init" />
    <row data-node="tbl.t_order_2" values="1002, 10, init" />
    <row data-node="tbl.t_order_3" values="1003, 10, init" />
    <row data-node="tbl.t_order_4" values="1004, 10, init" />
    <row data-node="tbl.t_order_5" values="1005, 10, init" />
    <row data-node="tbl.t_order_6" values="1006, 10, init" />
    <row data-node="tbl.t_order_7" values="1007, 10, init" />
    <row data-node="tbl.t_order_8" values="1008, 10, init" />
    <row data-node="tbl.t_order_9" values="1009, 10, init" />
</dataset>
```

Developer can customize DDL to create databases and tables in `schema.xml`.

Assertion Configuration

So far have confirmed what kind of sql execute in which environment in upon configuration, here define the data for assert. There are two kinds of config for assert, one is at `/shardingsphere-integration-test-suite/src/test/resources/cases/SQL-TYPE/SQL-TYPE-integration-test-cases.xml`. This file just like an index, defined the sql, parameters and expected index position for execution. the SQL is the value for `sql-case-id`. For example:

```
<integration-test-cases>
    <dml-test-case sql-case-id="insert_with_all_placeholders">
       <assertion parameters="1:int, 1:int, insert:String" expected-data-file=
"insert_for_order_1.xml" />
       <assertion parameters="2:int, 2:int, insert:String" expected-data-file=
"insert_for_order_2.xml" />
    </dml-test-case>
</integration-test-cases>
```

Another kind of config for assert is the data, as known as the corresponding expected-data-file in SQL-TYPE-integration-test-cases.xml, which is at `/shardingsphere-integration-test-suite/src/test/resources/cases/SQL-TYPE/dataset/FEATURE-TYPE/*.xml`.

This file is very like the dataset.xml mentioned before, and the difference is that expected-data-file contains some other assert data, such as the return value after a sql execution. For examples:

```
<dataset update-count="1">
    <metadata data-nodes="db_${0..9}.t_order">
        <column name="order_id" type="numeric" />
        <column name="user_id" type="numeric" />
```

```
        <column name="status" type="varchar" />
    </metadata>
    <row data-node="db_0.t_order" values="1000, 10, update" />
    <row data-node="db_0.t_order" values="1001, 10, init" />
    <row data-node="db_0.t_order" values="2000, 20, init" />
    <row data-node="db_0.t_order" values="2001, 20, init" />
</dataset>
```

Util now, all config files are ready, just launch the corresponding test case is fine.With no need to modify any Java code, only set up some config files. This will reduce the difficulty for ShardingSphere testing.

Notice

1. If Oracle needs to be tested, please add Oracle driver dependencies to the pom.xml.
2. 10 splitting-databases and 10 splitting-tables are used in the integrated test to ensure the test data is full, so it will take a relatively long time to run the test cases.

9.7.5 Performance Test

Provides result for each performance test tools.

Performance Test with Sysbench

Target

The performance of ShardingSphere-JDBC, ShardingSphere-Proxy and MySQL would be compared here. INSERT & UPDATE & DELETE which regarded as a set of associated operation and SELECT which focus on sharding optimization are used to evaluate performance for the basic scenarios (single route, readwrite-splitting & encrypt & sharding, full route). While another set of associated operation, INSERT & SELECT & DELETE, is used to evaluate performance for readwrite-splitting. To achieve the result better, these tests are performed with jmeter which based on a certain amount of data with 20 concurrent threads for 30 minutes, and one MySQL has been deployed on one machine, while the scenario of MySQL used for comparison is deployed on one machine with one instance.

Test Scenarios

Single Route

On the basis of one thousand data volume, four databases that are deployed on the same machine and each contains 1024 tables with `id` used for database sharding and `k` used for table sharding are designed for this scenario, single route select sql statement is chosen here. While as a comparison, MySQL runs with INSERT & UPDATE & DELETE statement and single route select sql statement on the basis of one thousand data volume.

Readwrite-splitting

One primary database and one replica database, which are deployed on different machines, are designed for this scenario based on ten thousand data volume. While as a comparison, MySQL runs with INSERT & SELECT & DELETE sql statement on the basis of ten thousand data volume.

Readwrite-splitting & Encrypt & Sharding

On the basis of one thousand data volume, four databases that are deployed on different machines and each contains 1024 tables with `id` used for database sharding, `k` used for table sharding, `c` encrypted with aes and `pad` encrypted with md5 are designed for this scenario, single route select sql statement is chosen here. While as a comparison, MySQL runs with INSERT & UPDATE & DELETE statement and single route select sql statement on the basis of one thousand data volume.

Full Route

On the basis of one thousand data volume, four databases that are deployed on different machines and each contains one table are designed for this scenario, field `id` is used for database sharding and `k` is used for table sharding, full route select sql statement is chosen here. While as a comparison, MySQL runs with INSERT & UPDATE & DELETE statement and full route select sql statement on the basis of one thousand data volume.

Testing Environment

Table Structure of Database

The structure of table here refer to `sbtest` in `sysbench`

```
CREATE TABLE `tbl` (
  `id` bigint(20) NOT NULL AUTO_INCREMENT,
  `k` int(11) NOT NULL DEFAULT 0,
  `c` char(120) NOT NULL DEFAULT '',
  `pad` char(60) NOT NULL DEFAULT '',
  PRIMARY KEY (`id`)
);
```

Test Scenarios Configuration

The same configurations are used for ShardingSphere-JDBC and ShardingSphere-Proxy, while MySQL with one database connected is designed for comparision. The details for these scenarios are shown as follows.

Single Route Configuration

```yaml
schemaName: sharding_db

dataSources:
  ds_0:
    url: jdbc:mysql://***.***.***.***:****/ds?serverTimezone=UTC&useSSL=false
    username: test
    password:
    connectionTimeoutMilliseconds: 30000
    idleTimeoutMilliseconds: 60000
    maxLifetimeMilliseconds: 1800000
    maxPoolSize: 200
  ds_1:
    url: jdbc:mysql://***.***.***.***:****/ds?serverTimezone=UTC&useSSL=false
    username: test
    password:
    connectionTimeoutMilliseconds: 30000
    idleTimeoutMilliseconds: 60000
    maxLifetimeMilliseconds: 1800000
    maxPoolSize: 200
  ds_2:
    url: jdbc:mysql://***.***.***.***:****/ds?serverTimezone=UTC&useSSL=false
    username: test
    password:
    connectionTimeoutMilliseconds: 30000
    idleTimeoutMilliseconds: 60000
    maxLifetimeMilliseconds: 1800000
    maxPoolSize: 200
  ds_3:
    url: jdbc:mysql://***.***.***.***:****/ds?serverTimezone=UTC&useSSL=false
    username: test
    password:
    connectionTimeoutMilliseconds: 30000
    idleTimeoutMilliseconds: 60000
    maxLifetimeMilliseconds: 1800000
    maxPoolSize: 200
rules:
- !SHARDING
  tables:
    tbl:
```

```
    actualDataNodes: ds_${0..3}.tbl${0..1023}
    tableStrategy:
      standard:
        shardingColumn: k
        shardingAlgorithmName: tbl_table_inline
    keyGenerateStrategy:
        column: id
        keyGeneratorName: snowflake
defaultDatabaseStrategy:
  inline:
    shardingColumn: id
    shardingAlgorithmName: default_db_inline
defaultTableStrategy:
  none:
shardingAlgorithms:
  tbl_table_inline:
    type: INLINE
    props:
      algorithm-expression: tbl${k % 1024}
  default_db_inline:
    type: INLINE
    props:
      algorithm-expression: ds_${id % 4}
keyGenerators:
  snowflake:
    type: SNOWFLAKE
    props:
      worker-id: 123
```

Readwrite-splitting Configuration

```
schemaName: sharding_db

dataSources:
  primary_ds:
    url: jdbc:mysql://***.***.***.***:****/ds?serverTimezone=UTC&useSSL=false
    username: test
    password:
    connectionTimeoutMilliseconds: 30000
    idleTimeoutMilliseconds: 60000
    maxLifetimeMilliseconds: 1800000
    maxPoolSize: 200
  replica_ds_0:
    url: jdbc:mysql://***.***.***.***:****/ds?serverTimezone=UTC&useSSL=false
    username: test
    password:
    connectionTimeoutMilliseconds: 30000
```

```yaml
    idleTimeoutMilliseconds: 60000
    maxLifetimeMilliseconds: 1800000
    maxPoolSize: 200
rules:
- !READWRITE_SPLITTING
  dataSources:
    pr_ds:
      writeDataSourceName: primary_ds
      readDataSourceNames:
        - replica_ds_0
```

Readwrite-splitting & Encrypt & Sharding Configuration

```yaml
schemaName: sharding_db

dataSources:
  primary_ds_0:
    url: jdbc:mysql://***.***.***.***:****/ds?serverTimezone=UTC&useSSL=false
    username: test
    password:
    connectionTimeoutMilliseconds: 30000
    idleTimeoutMilliseconds: 60000
    maxLifetimeMilliseconds: 1800000
    maxPoolSize: 200
  replica_ds_0:
    url: jdbc:mysql://***.***.***.***:****/ds?serverTimezone=UTC&useSSL=false
    username: test
    password:
    connectionTimeoutMilliseconds: 30000
    idleTimeoutMilliseconds: 60000
    maxLifetimeMilliseconds: 1800000
    maxPoolSize: 200
  primary_ds_1:
    url: jdbc:mysql://***.***.***.***:****/ds?serverTimezone=UTC&useSSL=false
    username: test
    password:
    connectionTimeoutMilliseconds: 30000
    idleTimeoutMilliseconds: 60000
    maxLifetimeMilliseconds: 1800000
    maxPoolSize: 200
  replica_ds_1:
    url: jdbc:mysql://***.***.***.***:****/ds?serverTimezone=UTC&useSSL=false
    username: test
    password:
    connectionTimeoutMilliseconds: 30000
    idleTimeoutMilliseconds: 60000
    maxLifetimeMilliseconds: 1800000
```

```yaml
    maxPoolSize: 200
  primary_ds_2:
    url: jdbc:mysql://***.***.***.***:****/ds?serverTimezone=UTC&useSSL=false
    username: test
    password:
    connectionTimeoutMilliseconds: 30000
    idleTimeoutMilliseconds: 60000
    maxLifetimeMilliseconds: 1800000
    maxPoolSize: 200
  replica_ds_2:
    url: jdbc:mysql://***.***.***.***:****/ds?serverTimezone=UTC&useSSL=false
    username: test
    password:
    connectionTimeoutMilliseconds: 30000
    idleTimeoutMilliseconds: 60000
    maxLifetimeMilliseconds: 1800000
    maxPoolSize: 200
  primary_ds_3:
    url: jdbc:mysql://***.***.***.***:****/ds?serverTimezone=UTC&useSSL=false
    username: test
    password:
    connectionTimeoutMilliseconds: 30000
    idleTimeoutMilliseconds: 60000
    maxLifetimeMilliseconds: 1800000
    maxPoolSize: 200
  replica_ds_3:
    url: jdbc:mysql://***.***.***.***:****/ds?serverTimezone=UTC&useSSL=false
    username: test
    password:
    connectionTimeoutMilliseconds: 30000
    idleTimeoutMilliseconds: 60000
    maxLifetimeMilliseconds: 1800000
    maxPoolSize: 200
rules:
- !SHARDING
  tables:
    tbl:
      actualDataNodes: pr_ds_${0..3}.tbl${0..1023}
      databaseStrategy:
        standard:
          shardingColumn: id
          shardingAlgorithmName: tbl_database_inline
      tableStrategy:
        standard:
          shardingColumn: k
          shardingAlgorithmName: tbl_table_inline
      keyGenerateStrategy:
        column: id
```

```yaml
      keyGeneratorName: snowflake
  bindingTables:
    - tbl
  defaultDataSourceName: primary_ds_1
  defaultTableStrategy:
    none:
  shardingAlgorithms:
    tbl_database_inline:
      type: INLINE
      props:
        algorithm-expression: pr_ds_${id % 4}
    tbl_table_inline:
      type: INLINE
      props:
        algorithm-expression: tbl${k % 1024}
  keyGenerators:
    snowflake:
      type: SNOWFLAKE
      props:
          worker-id: 123
- !READWRITE_SPLITTING
  dataSources:
    pr_ds_0:
      writeDataSourceName: primary_ds_0
      readDataSourceNames:
        - replica_ds_0
      loadBalancerName: round_robin
    pr_ds_1:
      writeDataSourceName: primary_ds_1
      readDataSourceNames:
        - replica_ds_1
      loadBalancerName: round_robin
    pr_ds_2:
      writeDataSourceName: primary_ds_2
      readDataSourceNames:
        - replica_ds_2
      loadBalancerName: round_robin
    pr_ds_3:
      writeDataSourceName: primary_ds_3
      readDataSourceNames:
        - replica_ds_3
      loadBalancerName: round_robin
  loadBalancers:
    round_robin:
      type: ROUND_ROBIN
- !ENCRYPT:
  encryptors:
    aes_encryptor:
```

```yaml
      type: AES
      props:
        aes-key-value: 123456abc
    md5_encryptor:
      type: MD5
  tables:
    sbtest:
      columns:
        c:
          plainColumn: c_plain
          cipherColumn: c_cipher
          encryptorName: aes_encryptor
        pad:
          cipherColumn: pad_cipher
          encryptorName: md5_encryptor
```

Full Route Configuration

```yaml
schemaName: sharding_db

dataSources:
  ds_0:
    url: jdbc:mysql://***.***.***.***:****/ds?serverTimezone=UTC&useSSL=false
    username: test
    password:
    connectionTimeoutMilliseconds: 30000
    idleTimeoutMilliseconds: 60000
    maxLifetimeMilliseconds: 1800000
    maxPoolSize: 200
  ds_1:
    url: jdbc:mysql://***.***.***.***:****/ds?serverTimezone=UTC&useSSL=false
    username: test
    password:
    connectionTimeoutMilliseconds: 30000
    idleTimeoutMilliseconds: 60000
    maxLifetimeMilliseconds: 1800000
    maxPoolSize: 200
  ds_2:
    url: jdbc:mysql://***.***.***.***:****/ds?serverTimezone=UTC&useSSL=false
    username: test
    password:
    connectionTimeoutMilliseconds: 30000
    idleTimeoutMilliseconds: 60000
    maxLifetimeMilliseconds: 1800000
    maxPoolSize: 200
  ds_3:
    url: jdbc:mysql://***.***.***.***:****/ds?serverTimezone=UTC&useSSL=false
```

```
    username: test
    password:
    connectionTimeoutMilliseconds: 30000
    idleTimeoutMilliseconds: 60000
    maxLifetimeMilliseconds: 1800000
    maxPoolSize: 200
rules:
- !SHARDING
  tables:
    tbl:
      actualDataNodes: ds_${0..3}.tbl1
      tableStrategy:
        standard:
          shardingColumn: k
          shardingAlgorithmName: tbl_table_inline
      keyGenerateStrategy:
        column: id
        keyGeneratorName: snowflake
  defaultDatabaseStrategy:
    standard:
      shardingColumn: id
      shardingAlgorithmName: default_database_inline
  defaultTableStrategy:
    none:
  shardingAlgorithms:
    default_database_inline:
      type: INLINE
      props:
        algorithm-expression: ds_${id % 4}
    tbl_table_inline:
      type: INLINE
      props:
        algorithm-expression: tbl1
  keyGenerators:
    snowflake:
      type: SNOWFLAKE
      props:
        worker-id: 123
```

Test Result Verification

SQL Statement

```
INSERT+UPDATE+DELETE sql statements:
INSERT INTO tbl(k, c, pad) VALUES(1, '###-###-###', '###-###');
UPDATE tbl SET c='####-####-####', pad='####-####' WHERE id=?;
DELETE FROM tbl WHERE id=?

SELECT sql statement for full route:
SELECT max(id) FROM tbl WHERE id%4=1

SELECT sql statement for single route:
SELECT id, k FROM tbl ignore index(`PRIMARY`) WHERE id=1 AND k=1

INSERT+SELECT+DELETE sql statements:
INSERT INTO tbl1(k, c, pad) VALUES(1, '###-###-###', '###-###');
SELECT count(id) FROM tbl1;
SELECT max(id) FROM tbl1 ignore index(`PRIMARY`);
DELETE FROM tbl1 WHERE id=?
```

Jmeter Class

Consider the implementation of shardingsphere-benchmark Notes: the notes in shardingsphere-benchmark/README.md should be taken attention to

Compile & Build

```
git clone https://github.com/apache/shardingsphere-benchmark.git
cd shardingsphere-benchmark/shardingsphere-benchmark
mvn clean install
```

Perform Test

```
cp target/shardingsphere-benchmark-1.0-SNAPSHOT-jar-with-dependencies.jar apache-
jmeter-4.0/lib/ext
jmeter -n -t test_plan/test.jmx
test.jmx example:https://github.com/apache/shardingsphere-benchmark/tree/master/
report/script/test_plan/test.jmx
```

Process Result Data

Make sure the location of result.jtl file is correct.

```
sh shardingsphere-benchmark/report/script/gen_report.sh
```

Display of Historical Performance Test Data

In progress, please wait.

9.7.6 Module Test

Provides test engine with each complex modules.

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.

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:

```yaml
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
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
      props:
        worker-id: 123
```

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.

9.8 FAQ

9.8.1 1. [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, then conflict occur when ShardingSphere-JDBC create datasources.
2. A simple way to solve this issue is removing the spring-boot-starter of certain datasource pool, shardingsphere-jdbc create datasources with suitable pools.

9.8.2 2. [JDBC] Why is xsd unable to be found when Spring Namespace is used?

Answer:

The use norm of Spring Namespace does not require to deploy 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.

9.8.3 3. [JDBC] Found a JtaTransactionManager in spring boot project when integrating with transaction of XA

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.

9.8.4 4. [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
```

9.8.5 5. [Proxy] How to add a new logic schema dynamically when use ShardingSphere-Proxy?

Answer:

When using ShardingSphere-Proxy, users can dynamically create or drop logic schema through `DistSQL`, the syntax is as follows:

```
CREATE (DATABASE | SCHEMA) [IF NOT EXISTS] schemaName;

DROP (DATABASE | SCHEMA) [IF EXISTS] schemaName;
```

Example:

```
CREATE DATABASE sharding_db;

DROP SCHEMA sharding_db;
```

9.8.6 6. [Proxy] How to use a suitable database tools connecting ShardingSphere-Proxy?

Answer:

1. ShardingSphere-Proxy could be considered as a mysql sever, so we recommend using mysql command line tool to connect to and operate it.
2. If users would like 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.

9.8.7 7. [Proxy] When using a client such as Navicat to connect to Sharding Sphere-Proxy, if Sharding Sphere-Proxy does not create a Schema 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 `schema` or does not add a `resource`, ShardingSphere-Proxy cannot execute SQL.
2. It is recommended to create `schema` and `resource` first, and then use third-party database tools to connect.
3. Please refer to Related introduction the details about `resource`.

9.8.8 8. [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.

9.8.9 9. [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).

9.8.10 10. [Sharding] If sharding database is partial, should tables without sharding database & table be configured in sharding rules?

Answer:

No, ShardingSphere will recognize it automatically.

9.8.11 11. [Sharding] When generic Long type `SingleKeyTableShardingAlgorithm` is used, why does `ClassCastException: Integer can not cast to Long` exception appear?

Answer:

You must make sure the field in database table consistent with that in 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.

9.8.12 12. [Sharding] Why are 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

9.8.13 13. [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`).

9.8.14 14. [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.

9.8.15 15. [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.

9.8.16 16. [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.

9.8.17 17. [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.

9.8.18 18. [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]
```

9.8.19 19. [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.

9.8.20 20. [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`.

9.8.21 21. [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:

```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:

```sql
SELECT SUM(num) AS sum_num, SUM(num2) AS sum_num2 FROM tablexxx;
```

9.8.22 22. [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;
}
```

9.8.23 23. [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

9.8.24 24. [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.

9.8.25 25. [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).

9.8.26 26. [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:

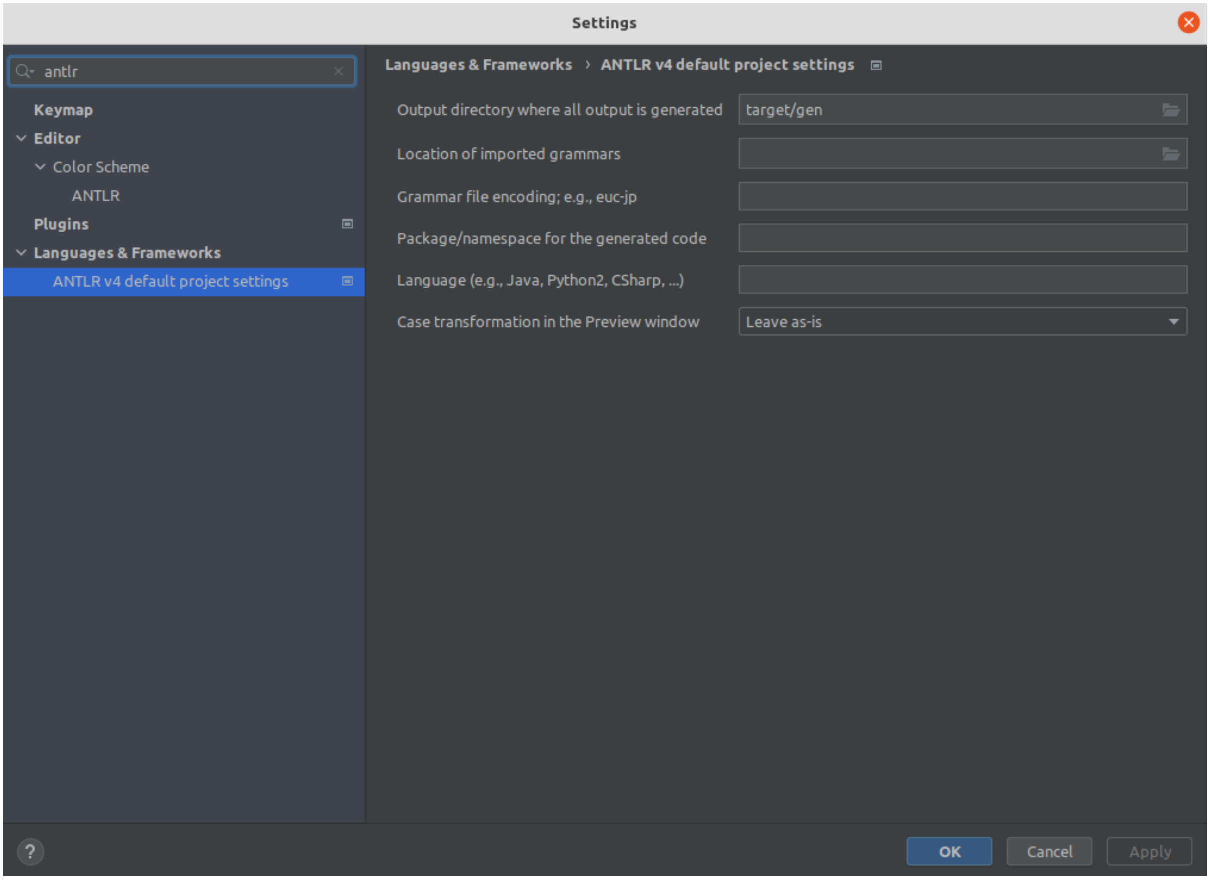

Figure29: Configure ANTLR plugin

9.8.27 27. [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.

9.9 API Change Histories

This chapter contains a section of API change histories of different projects of Apache ShardingSphere: ShardingSphere-JDBC, ShardingSphere-Proxy and ShardingSphere-Sidecar.

9.9.1 ShardingSphere-JDBC

This chapter contains a section of API change histories of Apache ShardingSphere-JDBC.

YAML configuration

5.0.0-alpha

Data Sharding

Configuration Item Explanation

```
dataSources: # Omit the data source configuration, please refer to the usage

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.
```

```
        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

  # 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
      # ...

props:
  # ...
```

Replica Query

Configuration Item Explanation

```yaml
dataSources: # Omit the data source configuration, please refer to the usage

rules:
- !REPLICA_QUERY
  dataSources:
    <data-source-name> (+): # Logic data source name of replica query
      primaryDataSourceName: # Primary data source name
      replicaDataSourceNames:
        - <replica-data_source-name> (+) # Replica data source name
      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
        # ...

props:
  # ...
```

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

Encryption

Configuration Item Explanation

```yaml
dataSource: # Omit the data source configuration, please refer to the usage

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

  # Encrypt algorithm configuration
  encryptors:
    <encrypt-algorithm-name> (+): # Encrypt algorithm name
```

```yaml
      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.

Shadow DB

Configuration Item Explanation

```yaml
dataSources: # Omit the data source configuration, please refer to the usage

rules:
- !SHADOW
  column: # Shadow column name
  sourceDataSourceNames: # Source Data Source names
     # ...
  shadowDataSourceNames: # Shadow Data Source names
     # ...

props:
  # ...
```

Governance

Configuration Item Explanation

```yaml
governance:
  name: # Governance name
  registryCenter: # Registry center
    type: # Governance instance type. Example:Zookeeper, etcd
    serverLists: # The list of servers that connect to governance instance,
including IP and port number; use commas to separate
  overwrite: # Whether to overwrite local configurations with config center
configurations; if it can, each initialization should refer to local configurations
```

ShardingSphere-4.x

Data Sharding

Configuration Item Explanation

```yaml
dataSources:
  ds0: !!org.apache.commons.dbcp.BasicDataSource
    driverClassName: com.mysql.jdbc.Driver
    url: jdbc:mysql://localhost:3306/ds0
    username: root
    password:
  ds1: !!org.apache.commons.dbcp.BasicDataSource
    driverClassName: com.mysql.jdbc.Driver
    url: jdbc:mysql://localhost:3306/ds1
    username: root
    password:

shardingRule:
  tables:
    t_order:
      actualDataNodes: ds${0..1}.t_order${0..1}
      databaseStrategy:
        inline:
          shardingColumn: user_id
          algorithmExpression: ds${user_id % 2}
      tableStrategy:
        inline:
          shardingColumn: order_id
          algorithmExpression: t_order${order_id % 2}
      keyGenerator:
        type: SNOWFLAKE
        column: order_id
    t_order_item:
      actualDataNodes: ds${0..1}.t_order_item${0..1}
      databaseStrategy:
        inline:
          shardingColumn: user_id
          algorithmExpression: ds${user_id % 2}
      tableStrategy:
        inline:
          shardingColumn: order_id
          algorithmExpression: t_order_item${order_id % 2}
  bindingTables:
    - t_order,t_order_item
  broadcastTables:
    - t_config
```

```
  defaultDataSourceName: ds0
  defaultTableStrategy:
    none:
  defaultKeyGenerator:
    type: SNOWFLAKE
    column: order_id

props:
  sql.show: true
```

Read-Write Split

Configuration Item Explanation

```
dataSources:
  ds_master: !!org.apache.commons.dbcp.BasicDataSource
    driverClassName: com.mysql.jdbc.Driver
    url: jdbc:mysql://localhost:3306/ds_master
    username: root
    password:
  ds_slave0: !!org.apache.commons.dbcp.BasicDataSource
    driverClassName: com.mysql.jdbc.Driver
    url: jdbc:mysql://localhost:3306/ds_slave0
    username: root
    password:
  ds_slave1: !!org.apache.commons.dbcp.BasicDataSource
    driverClassName: com.mysql.jdbc.Driver
    url: jdbc:mysql://localhost:3306/ds_slave1
    username: root
    password:

masterSlaveRule:
  name: ds_ms
  masterDataSourceName: ds_master
  slaveDataSourceNames: [ds_slave0, ds_slave1]

props:
  sql.show: true
```

Create a `DataSource` through the `YamlMasterSlaveDataSourceFactory` factory class:

```
DataSource dataSource = YamlMasterSlaveDataSourceFactory.
createDataSource(yamlFile);
```

Data Masking

Configuration Item Explanation

```yaml
dataSource:  !!org.apache.commons.dbcp2.BasicDataSource
  driverClassName: com.mysql.jdbc.Driver
  url: jdbc:mysql://127.0.0.1:3306/encrypt?serverTimezone=UTC&useSSL=false
  username: root
  password:

encryptRule:
  encryptors:
    encryptor_aes:
      type: aes
      props:
        aes.key.value: 123456abc
    encryptor_md5:
      type: md5
  tables:
    t_encrypt:
      columns:
        user_id:
          plainColumn: user_plain
          cipherColumn: user_cipher
          encryptor: encryptor_aes
        order_id:
          cipherColumn: order_cipher
          encryptor: encryptor_md5
props:
  query.with.cipher.column: true # use ciphertext column query
```

Orchestration

Configuration Item Explanation

```yaml
# Omit data sharding, Read-Write split, and Data masking configuration.

orchestration:
  name: orchestration_ds
  overwrite: true
  registry:
    type: zookeeper
    namespace: orchestration
    serverLists: localhost:2181
```

ShardingSphere-3.x

Data Sharding

Configuration Item Explanation

```yaml
dataSources:
  ds0: !!org.apache.commons.dbcp.BasicDataSource
    driverClassName: com.mysql.jdbc.Driver
    url: jdbc:mysql://localhost:3306/ds0
    username: root
    password:
  ds1: !!org.apache.commons.dbcp.BasicDataSource
    driverClassName: com.mysql.jdbc.Driver
    url: jdbc:mysql://localhost:3306/ds1
    username: root
    password:

shardingRule:
  tables:
    t_order:
      actualDataNodes: ds${0..1}.t_order${0..1}
      databaseStrategy:
        inline:
          shardingColumn: user_id
          algorithmExpression: ds${user_id % 2}
      tableStrategy:
        inline:
          shardingColumn: order_id
          algorithmExpression: t_order${order_id % 2}
      keyGeneratorColumnName: order_id
    t_order_item:
      actualDataNodes: ds${0..1}.t_order_item${0..1}
      databaseStrategy:
        inline:
          shardingColumn: user_id
          algorithmExpression: ds${user_id % 2}
      tableStrategy:
        inline:
          shardingColumn: order_id
          algorithmExpression: t_order_item${order_id % 2}
  bindingTables:
    - t_order,t_order_item
  broadcastTables:
    - t_config

  defaultDataSourceName: ds0
  defaultTableStrategy:
```

```
    none:
  defaultKeyGeneratorClassName: io.shardingsphere.core.keygen.DefaultKeyGenerator

props:
  sql.show: true
```

Read-Write Split

Configuration Item Explanation

```
dataSources:
  ds_master: !!org.apache.commons.dbcp.BasicDataSource
    driverClassName: com.mysql.jdbc.Driver
    url: jdbc:mysql://localhost:3306/ds_master
    username: root
    password:
  ds_slave0: !!org.apache.commons.dbcp.BasicDataSource
    driverClassName: com.mysql.jdbc.Driver
    url: jdbc:mysql://localhost:3306/ds_slave0
    username: root
    password:
  ds_slave1: !!org.apache.commons.dbcp.BasicDataSource
    driverClassName: com.mysql.jdbc.Driver
    url: jdbc:mysql://localhost:3306/ds_slave1
    username: root
    password:

masterSlaveRule:
  name: ds_ms
  masterDataSourceName: ds_master
  slaveDataSourceNames: [ds_slave0, ds_slave1]
  props:
      sql.show: true
  configMap:
      key1: value1
```

Create a `DataSource` through the `YamlMasterSlaveDataSourceFactory` factory class:

```
DataSource dataSource = MasterSlaveDataSourceFactory.createDataSource(yamlFile);
```

Orchestration

Configuration Item Explanation

```
# Omit data sharding, Read-Write split configuration.

orchestration:
  name: orchestration_ds
  overwrite: true
  registry:
    namespace: orchestration
    serverLists: localhost:2181
```

ShardingSphere-2.x

Data Sharding

Configuration Item Explanation

```
dataSources:
  db0: !!org.apache.commons.dbcp.BasicDataSource
    driverClassName: org.h2.Driver
    url: jdbc:h2:mem:db0;DB_CLOSE_DELAY=-1;DATABASE_TO_UPPER=false;MODE=MYSQL
    username: sa
    password:
    maxActive: 100
  db1: !!org.apache.commons.dbcp.BasicDataSource
    driverClassName: org.h2.Driver
    url: jdbc:h2:mem:db1;DB_CLOSE_DELAY=-1;DATABASE_TO_UPPER=false;MODE=MYSQL
    username: sa
    password:
    maxActive: 100

shardingRule:
  tables:
    config:
      actualDataNodes: db${0..1}.t_config
    t_order:
      actualDataNodes: db${0..1}.t_order_${0..1}
      databaseStrategy:
        standard:
          shardingColumn: user_id
          preciseAlgorithmClassName: io.shardingjdbc.core.yaml.fixture.
SingleAlgorithm
      tableStrategy:
        inline:
```

```
          shardingColumn: order_id
          algorithmInlineExpression: t_order_${order_id % 2}
      keyGeneratorColumnName: order_id
      keyGeneratorClass: io.shardingjdbc.core.yaml.fixture.IncrementKeyGenerator
    t_order_item:
      actualDataNodes: db${0..1}.t_order_item_${0..1}
      # The strategy of binding the rest of the tables in the table is the same as
the strategy of the first table
      databaseStrategy:
        standard:
          shardingColumn: user_id
          preciseAlgorithmClassName: io.shardingjdbc.core.yaml.fixture.
SingleAlgorithm
      tableStrategy:
        inline:
          shardingColumn: order_id
          algorithmInlineExpression: t_order_item_${order_id % 2}
  bindingTables:
    - t_order,t_order_item
  # Default database sharding strategy
  defaultDatabaseStrategy:
    none:
  defaultTableStrategy:
    complex:
      shardingColumns: id, order_id
      algorithmClassName: io.shardingjdbc.core.yaml.fixture.MultiAlgorithm
  props:
    sql.show: true
```

Read-Write Split

concept

In order to relieve the pressure on the database, the write and read operations are separated into different data sources. The write library is called the master library, and the read library is called the slave library. One master library can be configured with multiple slave libraries.

Supported

1. Provides a read-write separation configuration with one master and multiple slaves, which can be used independently or with sub-databases and sub-meters.
2. Independent use of read-write separation to support SQL transparent transmission.
3. In the same thread and the same database connection, if there is a write operation, subsequent read operations will be read from the main library to ensure data consistency.

4. Spring namespace.
5. Hint-based mandatory main library routing.

Unsupported

1. Data synchronization between the master library and the slave library.
2. Data inconsistency caused by the data synchronization delay of the master library and the slave library.
3. Double writing or multiple writing in the main library.

rule configuration

```
dataSources:
  db_master: !!org.apache.commons.dbcp.BasicDataSource
    driverClassName: org.h2.Driver
    url: jdbc:h2:mem:db_master;DB_CLOSE_DELAY=-1;DATABASE_TO_UPPER=false;MODE=MYSQL
    username: sa
    password:
    maxActive: 100
  db_slave_0: !!org.apache.commons.dbcp.BasicDataSource
    driverClassName: org.h2.Driver
    url: jdbc:h2:mem:db_slave_0;DB_CLOSE_DELAY=-1;DATABASE_TO_UPPER=false;
MODE=MYSQL
    username: sa
    password:
    maxActive: 100
  db_slave_1: !!org.apache.commons.dbcp.BasicDataSource
      driverClassName: org.h2.Driver
      url: jdbc:h2:mem:db_slave_1;DB_CLOSE_DELAY=-1;DATABASE_TO_UPPER=false;
MODE=MYSQL
      username: sa
      password:
      maxActive: 100

masterSlaveRule:
  name: db_ms
  masterDataSourceName: db_master
  slaveDataSourceNames: [db_slave_0, db_slave_1]
  configMap:
    key1: value1
```

Create a `DataSource` through the `MasterSlaveDataSourceFactory` factory class:

```
DataSource dataSource = MasterSlaveDataSourceFactory.createDataSource(yamlFile);
```

Orchestration

Configuration Item Explanation

Zookeeper sharding table and database Orchestration Configuration Item Explanation

```
dataSources: Data sources configuration

shardingRule: Sharding rule configuration

orchestration: Zookeeper Orchestration Configuration
  name: Orchestration name
  overwrite: Whether to overwrite local configurations with config center
configurations; if it can, each initialization should refer to local configurations
  zookeeper: Registry center Configuration
    namespace: Registry center namespace
    serverLists: The list of servers that connect to governance instance, including
IP and port number, use commas to separate, such as: host1:2181,host2:2181
    baseSleepTimeMilliseconds: The initial millisecond value of the interval to
wait for retry
    maxSleepTimeMilliseconds: The maximum millisecond value of the interval to wait
for retry
    maxRetries: The maximum retry count
    sessionTimeoutMilliseconds: The session timeout milliseconds
    connectionTimeoutMilliseconds: The connecton timeout milliseconds
    digest: Permission token to connect to Zookeeper. default no authorization is
required
```

Etcd sharding table and database Orchestration Configuration Item Explanation

```
dataSources: Data sources configuration

shardingRule: Sharding rule configuration

orchestration: Etcd Orchestration Configuration
  name: Orchestration name
  overwrite: Whether to overwrite local configurations with config center
configurations; if it can, each initialization should refer to local configurations
  etcd: Registry center Configuration
    serverLists: The list of servers that connect to governance instance, including
IP and port number, use commas to separate, such as: http://host1:2379,http://
host2:2379
    timeToLiveSeconds: Time to live seconds for ephemeral nodes
    timeoutMilliseconds: The request timeout milliseconds
    maxRetries: The maximum retry count
    retryIntervalMilliseconds: The retry interval milliseconds
```

Sharding table and database Data source construction method

```
DataSource dataSource = OrchestrationShardingDataSourceFactory.
createDataSource(yamlFile);
```

Read-Write split Data source construction method

```
DataSource dataSource = OrchestrationMasterSlaveDataSourceFactory.
createDataSource(yamlFile);
```

Java API

5.0.0-beta

Sharding

Root Configuration

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

Attributes:

Name	*DataType*	*Description*	*Def ault Va lue*
tables (+)	Collection<ShardingTa bleRuleConfiguration>	Sharding table rules	•
autoTables (+)	Coll ection<ShardingAutoTa bleRuleConfiguration>	Sharding automatic table rules	•
bind ingTableGroups (*)	Collection<String>	Binding table rules	E mpty
b roadcastTables (*)	Collection<String>	Broadcast table rules	E mpty
def aultDatabaseSh ardingStrategy (?)	Sharding StrategyConfiguration	Default database sharding strategy	Not shar ding
defaultTableSh ardingStrategy (?)	Sharding StrategyConfiguration	Default table sharding strategy	Not shar ding
defaultKeyGe nerateStrategy (?)	KeyG eneratorConfiguration	Default key generator	S nowf lake
shar dingAlgorithms (+)	Map<String, ShardingSphereAl gorithmConfiguration>	Sharding algorithm name and configurations	None
keyGenerators (?)	Map<String, ShardingSphereAl gorithmConfiguration>	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 Automatic 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.

Key Generate 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.

Readwrite-splitting

Root Configuration

Class name: ReadwriteSplittingRuleConfiguration

Attributes:

• Name*	*DataType*	*Description*
d ataSo urces (+)	Collectio n<ReadwriteSplittingData SourceRuleConfiguration>	Data sources of write and reads
loa dBala ncers (*)	Map<String, ShardingSpher eAlgorithmConfiguration>	Load balance algorithm name and configurations of replica data sources

Readwrite-splitting Data Source Configuration

Class name: ReadwriteSplittingDataSourceRuleConfiguration

Attributes:

Name	*D ataType*	*Description*	*Default Value*
name	String	Readwrite-splitting data source name	•
writeDat aSource-Name	String	Write sources source name	•
readData Source-Names (+)	Co llection <String>	Read sources source name list	•
loadB alancerName (?)	String	Load balance algo-rithm name of replica sources	Round robin load bal-ance algorithm

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

Encryption

Root Configuration

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

Attributes:

Name	*DataType*	*Description*	*D ef au lt Va lu e*
tables (+)	Collectio n<EncryptTableRu leConfiguration>	Encrypt table rule configurations	
encr yptors (+)	Map<String, Shard ingSphereAlgorit hmCon-figuration>	Encrypt algorithm name and configurations	
que ryWith Cipher Col-umn (?)	boolean	Whether query with cipher column for data encrypt. User you can use plaintext to query if have	tr ue

Encrypt Table Rule Configuration

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

Attributes:

• Name*	*DataType*	*Description*
name	String	Table name
co lumns (+)	Collection <EncryptColumnRuleConfiguration>	Encrypt column rule configurations

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

Encrypt Algorithm Configuration

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

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.

Shadow DB

Root Configuration

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

Attributes:

N am e	*D ataT ype*	*Description*
co lu mn	St ring	Shadow field name in SQL, SQL with a value of true will be routed to the shadow database for execution
s ou rc eD at aS ou rc eN am es	List <Str ing>	Source data source names
s ha do wD at aS ou rc eN am es	List <Str ing>	Shadow data source names

Governance

Configuration Item Explanation

Management

Configuration Entrance

Class name: org.apache.shardingsphere.governance.repository.api.config.GovernanceConfiguration

Attributes:

Name	*Data Type*	*Description*
name	String	Governance instance name
regist ryCenterConfiguration	Regis tryCenterConfiguration	Config of registry-center

The type of registryCenter could be Zookeeper or Etcd.

Governance Instance Configuration

Class name: org.apache.shardingsphere.governance.repository.api.config.ClusterPersistRepositoryConfiguration

Attributes:

• Name*	• DataType*	*Description*
type	String	Governance instance type, such as: Zookeeper, etcd
serverLists	String	The list of servers that connect to governance instance, including IP and port number, use commas to separate, such as: host1:2181,host2:2181
props	Properties	Properties for center instance config, such as options of zookeeper
overwrite	boolean	Local configurations overwrite config center configurations or not; if they overwrite, each start takes reference of local configurations

ZooKeeper Properties Configuration

Name	• Data T ype*	*Description*	*Default Value*
digest (?)	St ring	Connect to authority tokens in registry center	No need for authority
operationTi meoutMilliseconds (?)	int	The operation timeout milliseconds	500 mi lliseconds
maxRetries (?)	int	The maximum retry count	3
retryInt ervalMilliseconds (?)	int	The retry interval milliseconds	500 mi lliseconds
timeToLiveSeconds (?)	int	Time to live seconds for ephemeral nodes	60 seconds

Etcd Properties Configuration

Name	*Data Type*	*Description*	*Default Value*
ti meToLiveSeconds (?)	long	Time to live seconds for data persist	30 seconds

ShardingSphere-4.x

Sharding

ShardingDataSourceFactory

Name	*DataType*	*Explanation*
dataSourceMap	Map<String, DataSource>	Data sources configuration
sha rdingRuleConfig	Shar dingRuleConfiguration	Data sharding configuration rule
props (?)	Properties	Property configurations

ShardingRuleConfiguration

Name	*DataType*	*Explanation*
t ableRuleConfigs	Collection	Sharding rule list
bin dingTableGroups (?)	Collection	Binding table rule list
broadcastTables (?)	Collection	Broadcast table rule list
defaul tDataSourceName (?)	String	Tables not configured with sharding rules will locate according to default data sources
default DatabaseShardin gStrategyConfig (?)	Sharding StrategyConfiguration	Default database sharding strategy
defa ultTableShardin gStrategyConfig (?)	Sharding StrategyConfiguration	Default table sharding strategy
defaultKey GeneratorConfig (?)	KeyG eneratorConfiguration	Default key generator configuration, use user-defined ones or built-in ones, e.g. SNOWFLAKE/UUID. Default key generator is `o rg. apache.shardingsphere. core.keygen.generator.imp l. SnowflakeKeyGenerator`
masterS laveRuleConfigs (?)	Collection	Read-write split rules, default indicates not using read-write split

TableRuleConfiguration

Name	*DataType*	*Explanation*
logicTable	String	Name of logic table
actual-DataNodes (?)	String	Describe data source names and actual tables, delimiter as point, multiple data nodes split by comma, support inline expression. Absent means sharding databases only. Example: ds: math:*{0..7}.tbl*{0..7}
database-ShardingStrate-gyConfig (?)	Sharding Strategy-Configura-tion	Databases sharding strategy, use default databases sharding strategy if absent
table-ShardingStrate-gyConfig (?)	Sharding Strategy-Configura-tion	Tables sharding strategy, use default databases sharding strategy if absent
key Genera-torConfig (?)	KeyG en-eratorCon-figuration	Key generator configuration, use default key generator if absent
encrypt orConfig-uration (?)	E ncryp-torConfig-uration	Encrypt generator configuration

StandardShardingStrategyConfiguration

Subclass of ShardingStrategyConfiguration.

Name	*DataType*	*Explanation*
shardingColumn	String	Sharding column name
preciseSh ardingAlgo-rithm	Pre ciseShardingAlgo-rithm	Precise sharding algorithm used in = and `IN`
rangeSh ardingAlgorithm (?)	R angeShardingAlgo-rithm	Range sharding algorithm used in `BETWEEN`

ComplexShardingStrategyConfiguration

The implementation class of `ShardingStrategyConfiguration`, used in complex sharding situations with multiple sharding keys.

Name	*DataType*	*Explanation*
shardingColumns	String	Sharding column name, separated by commas
shardingAlgorithm	Complex KeysShardingAlgorithm	Complex sharding algorithm

InlineShardingStrategyConfiguration

The implementation class of `ShardingStrategyConfiguration`, used in sharding strategy of inline expression.

Name	*DataType*	*Explanation*
shardingColumns	String	Sharding column name, separated by commas
algo rithmExpression	String	Inline expression of sharding strategies, should conform to groovy syntax; refer to Inline expression for more details

HintShardingStrategyConfiguration

The implementation class of `ShardingStrategyConfiguration`, used to configure hint sharding strategies.

Name	*DataType*	*Description*
shardingAlgorithm	HintShardingAlgorithm	Hint sharding algorithm

NoneShardingStrategyConfiguration

The implementation class of `ShardingStrategyConfiguration`, used to configure none-sharding strategies.

KeyGeneratorConfiguration

Name	*DataType*	*Description*
column	String	Column name of key generator
type	String	Type of key generator, use user-defined ones or built-in ones, e.g. SNOWFLAKE, UUID
props	Properties	The Property configuration of key generators

Properties

Property configuration that can include these properties of these key generators.

SNOWFLAKE

Name	• Data T ype*	*Explanation*
worker.id (?)	long	The unique id for working machine, the default value is `0`
max.tolerate.time.difference.milliseconds (?)	long	The max tolerate time for different server's time difference in milliseconds, the default value is `10`
max.vibration.offset (?)	int	The max upper limit value of vibrate number, range `[0, 4096)`, the default value is `1`. 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`

Readwrite-splitting

MasterSlaveDataSourceFactory

Name	*DataType*	*Explanation*
dataSourceMap	Map<String, DataSource>	Mapping of data source and its name
master SlaveRuleConfig	MasterS laveRuleConfiguration	Master slave rule configuration
props (?)	Properties	Property configurations

MasterSlaveRuleConfiguration

Name	*DataType*	*Explanation*
name	String	Readwrite-splitting data source name
maste rDataSource-Name	String	Master database source name
slave DataSourceNames	Collection	Slave database source name list
loadB alanceAlgorithm (?)	MasterSlav eLoadBalanceAlgo-rithm	Slave database load balance

Properties

Property configuration items, can be of the following properties.

Name	• Data T ype*	*Explanation*
sql.show (?)	boo lean	Print SQL parse and rewrite log or not, default value: false
executor.size (?)	int	Be used in work thread number implemented by SQL; no limits if it is 0. default value: 0
max.connec tions.size.per.query (?)	int	The maximum connec-tion number allocated by each query of each physical database, default value: 1
check.ta ble.metadata.enabled (?)	boo lean	Check meta-data consistency or not in initialization, default value: false

Data Masking

EncryptDataSourceFactory

Name	*DataType*	*Explanation*
dataSource	DataSource	Data source
encryptRuleConfig	EncryptRuleConfiguration	encrypt rule configuration
props (?)	Properties	Property configurations

EncryptRuleConfiguration

Name	*DataType*	*Explanation*
encryp- tors	Map<String, Encryp torRuleConfiguration>	Encryptor names and encryptors
tables	Map<String, EncryptTa bleRuleConfigura- tion>	Encrypt table names and encrypt ta- bles

Properties

Property configuration items, can be of the following properties.

Name	• Data T ype*	*Explanation*
sql.show (?)	boo lean	Print SQL parse and rewrite log or not, default value: false
quer y.with.cipher.column (?)	boo lean	When there is a plainColumn, use cipherColumn or not to query, default value: true

Orchestration

OrchestrationShardingDataSourceFactory

Name	*DataType*	*Explanation*
dataSourceMap	Map<String, DataSource>	Same as `` ShardingDataSourceFactory``
sha rdingRuleConfig	Shar dingRuleConfiguration	Same as `` ShardingDataSourceFactory``
props (?)	Properties	Same as `` ShardingDataSourceFactory``
orch estrationConfig	Orche strationConfiguration	Orchestration rule configurations

OrchestrationMasterSlaveDataSourceFactory

Name	*DataType*	*Explanation*
dataSourceMap	Map<String, DataSource>	Same as `Mas terSlaveDataSourceFactory`
master SlaveRule-Config	MasterS laveRuleConfiguration	Same as `Mas terSlaveDataSourceFactory`
configMap (?)	Map<String, Object>	Same as `Mas terSlaveDataSourceFactory`
props (?)	Properties	Same as `` ShardingDataSourceFactory``
orch estrationConfig	Orche strationConfiguration	Orchestration rule configurations

OrchestrationEncryptDataSourceFactory

Name	*DataType*	*Explanation*
dataSource	DataSource	Same as ` *EncryptDataSourceFactory*`
en cryptRuleConfig	Enc ryptRuleConfiguration	Same as ` *EncryptDataSourceFactory*`
props (?)	Properties	Same as ` *EncryptDataSourceFactory*`
orch estrationConfig	Orche strationConfiguration	Orchestration rule configurations

OrchestrationConfiguration

Name	*DataType*	*Explanation*
instanceC onfiguration-Map	Map<String, CenterConfiguration>	config map of config- center&registry-center, the key is center's name, the value is the co nfig-center/registry-center

CenterConfiguration

Name	*DataType*	*Explanation*
type	String	The type of center instance(z ookeeper/etcd/apollo/nacos)
proper- ties	String	Properties for center instance config, such as options of zookeeper
or ches- tra- tionType	String	The type of orchestration center: config-center or registry-center, if both, use "s etOrchestrationType("regist ry_center,config_center");"
serverLists	String	Connect to server lists in center, including IP address and port number; ad- dresses are separated by commas, such as `host1:2181,host2:2181`
names- pace (?)	String	Namespace of center instance

Properties

Property configuration items, can be of the following properties.

Name	• Data T ype*	*Explanation*
overwrite	boo lean	Local configurations overwrite center configurations or not; if they overwrite, each start takes reference of local configura- tions

If type of center is `zookeeper` with config-center&registry-center, properties could be set with the follow options:

Name	• Data T ype*	*Explanation*
digest (?)	St ring	Connect to authority tokens in registry center; default indicates no need for authority
operatio nTimeoutMilliseconds (?)	int	The operation timeout millisecond number, default to be 500 milliseconds
maxRetries (?)	int	The maximum retry count, default to be 3 times
retry IntervalMilliseconds (?)	int	The retry interval millisecond number, default to be 500 milliseconds
timeToLiveSeconds (?)	int	The living time for temporary nodes, default to be 60 seconds

If type of center is `etcd` with config-center&registry-center, properties could be set with the follow options:

Name	• Data T ype*	*Explanation*
timeToLiveSeconds (?)	long	The etcd TTL in seconds, default to be 30 seconds

If type of center is `apollo` with config-center&registry-center, properties could be set with the follow options:

Name	• Data T ype*	*Explanation*
appId (?)	St ring	Apollo appId, default to be "APOLLO_SHARDINGSPHERE"
env (?)	St ring	Apollo env, default to be "DEV"
clusterName (?)	St ring	Apollo clusterName, default to be "default"
administrator (?)	St ring	Apollo administrator, default to be ""
token (?)	St ring	Apollo token, default to be ""
portalUrl (?)	St ring	Apollo portalUrl, default to be ""
connectTimeout (?)	int	Apollo connectTimeout, default to be 1000 milliseconds
readTimeout (?)	int	Apollo readTimeout, default to be 5000 milliseconds

If type of center is nacos with config-center&registry-center, properties could be set with the follow options:

Name	*Data Type*	*Explanation*
group (?)	String	Nacos group, "SHARDING_SPHERE_DEFAULT_GROUP" in default
timeout (?)	long	Nacos timeout, default to be 3000 milliseconds

ShardingSphere-3.x

Sharding

ShardingDataSourceFactory

Name	*DataType*	*Explanation*
dataSourceMap	Map<String, DataSource>	Data sources configuration
sha rdingRuleConfig	Shar dingRuleConfiguration	Data sharding configuration rule
configMap (?)	Map<String, Object>	Config map
props (?)	Properties	Property configurations

ShardingRuleConfiguration

Name	*DataType*	*Explanation*
t ableRuleConfigs	Collection	Table rule configuration
bin dingTableGroups (?)	Collection	Binding table groups
broadcastTables (?)	Collection	Broadcast table groups
defaul tDataSource-Name (?)	String	Tables not configured with sharding rules will locate according to default data sources
default Database-Shardin gStrategyConfig (?)	Sharding StrategyConfiguration	Default database sharding strategy
defa ultTableShardin gStrategyConfig (?)	Sharding StrategyConfiguration	Default table sharding strategy
defaultKey Generator-Config (?)	KeyGenerator	Default key generator, default value is io.shardingsphere.core.keygen.DefaultKeyGenerator
masterS laveRuleConfigs (?)	Collection	Read-write splitting rule configuration

TableRuleConfiguration

Name	*DataType*	*Explanation*
logicTable	String	Name of logic table
actual-DataNodes (?)	String	Describe data source names and actual tables, delimiter as point, multiple data nodes split by comma, support inline expression. Absent means sharding databases only. Example: ds: math:*{0..7}.tbl*{0..7}
database-ShardingStrate-gyConfig (?)	Sharding Strategy-Configura-tion	Databases sharding strategy, use default databases sharding strategy if absent
table-ShardingStrate-gyConfig (?)	Sharding Strategy-Configura-tion	Tables sharding strategy, use default databases sharding strategy if absent
logicIndex (?)	String	Name if logic index. If use DROP INDEX XXX SQL in Oracle/PostgreSQL, This property needs to be set for finding the actual tables
key Genera-torConfig (?)	String	Key generator column name, do not use Key generator if absent
keyGenera-tor (?)	KeyGener-ator	Key generator, use default key generator if absent

StandardShardingStrategyConfiguration

Subclass of ShardingStrategyConfiguration.

Name	*DataType*	*Explanation*
shardingColumn	String	Sharding column name
preciseSh ardingAlgo-rithm	Pre ciseShardingAlgo-rithm	Precise sharding algorithm used in `=` and `IN`
rangeSh ardingAlgorithm (?)	R angeShardingAlgo-rithm	Range sharding algorithm used in `BETWEEN`

ComplexShardingStrategyConfiguration

Subclass of ShardingStrategyConfiguration.

Name	*DataType*	*Explanation*
shardingColumns	String	Sharding column name, separated by commas
shardingAlgorithm	Complex KeysShardingAlgorithm	Complex sharding algorithm

InlineShardingStrategyConfiguration

Subclass of ShardingStrategyConfiguration.

Name	*DataType*	*Explanation*
shardingColumns	String	Sharding column name, separated by commas
algo rithmExpression	String	Inline expression of sharding strategies, should conform to groovy syntax; refer to Inline expression for more details

HintShardingStrategyConfiguration

Subclass of ShardingStrategyConfiguration.

Name	*DataType*	*Description*
shardingAlgorithm	HintShardingAlgorithm	Hint sharding algorithm

NoneShardingStrategyConfiguration

Subclass of ShardingStrategyConfiguration.

Properties

Enumeration of properties.

Name	• Data T ype*	*Explanation*
sql.show (?)	boo lean	Print SQL parse and rewrite log, default value: false
executor.size (?)	int	The number of SQL execution threads, zero means no limit. default value: 0
max.connec tions.size.per.query (?)	int	Max connection size for every query to every actual database. default value: 1
check.ta ble.metadata.enabled (?)	boo lean	Check the metadata consistency of all the tables, default value : false

configMap

User-defined arguments.

Readwrite-splitting

MasterSlaveDataSourceFactory

Name	*DataType*	*Description*
dataSourceMap	Map<String, DataSource>	Map of data sources and their names
master SlaveRuleConfig	MasterS laveRuleConfiguration	Master slave rule configuration
configMap (?)	Map<String, Object>	Config map
props (?)	Properties	Properties

MasterSlaveRuleConfiguration

Name	*DataType*	*Description*
name	String	Name of master slave data source
mast erDataSourceName	String	Name of master data source
slav eDataSourceNames	Collection	Names of Slave data sources
load BalanceAlgorithm (?)	MasterSla veLoadBalanceAlgorithm	Load balance algorithm

configMap

User-defined arguments.

PropertiesConstant

Enumeration of properties.

Name	*Dat aTy pe**	*Description*
sql.show (?)	b ool ean	To show SQLS or not, default value: false
executor.size (?)	int	The number of working threads, default value: CPU count
max.connec tions.size.per.query (?)	int	Max connection size for every query to every actual database. default value: 1
check.ta ble.metadata.enabled (?)	b ool ean	Check the metadata consistency of all the tables, default value : false

Orchestration

OrchestrationShardingDataSourceFactory

Name	*DataType*	*Explanation*
dataSourceMap	Map<String, DataSource>	Same as `` ShardingDataSourceFactory``
sha rdingRuleConfig	Shar dingRuleConfiguration	Same as `` ShardingDataSourceFactory``
configMap (?)	Map<String, Object>	Same with `` ShardingDataSourceFactory``
props (?)	Properties	Same as `` ShardingDataSourceFactory``
orch estrationConfig	Orche strationConfiguration	Orchestration rule configurations

OrchestrationMasterSlaveDataSourceFactory

Name	*DataType*	*Explanation*
dataSourceMap	Map<String, DataSource>	Same as `Mas terSlaveDataSourceFac-tory`
master SlaveRule-Config	MasterS laveRuleConfigura-tion	Same as `Mas terSlaveDataSourceFac-tory`
configMap (?)	Map<String, Object>	Same as `Mas terSlaveDataSourceFac-tory`
props (?)	Properties	Same as `` ShardingDataSourceFactory``
orch estrationConfig	Orche strationConfiguration	Orchestration configurations

OrchestrationConfiguration

Name	*DataType*	*Explanation*
name	String	Name of orchestration instance
overwrite	boolean	Use local configuration to overwrite registry center or not
regCenterCon-fig	Regist ryCenterConfigura-tion	Registry center configuration

RegistryCenterConfiguration

Name	*DataType*	*Explanation*
serverLists	String	Registry servers list, multiple split as comma. Example: host1:2181,host2:2181
namespace (?)	String	Namespace of registry
digest (?)	String	Digest for registry. Default is not need digest.
operationTime outMil-liseconds (?)	int	Operation timeout time in milliseconds. Default value is 500 milliseconds.
maxRetries (?)	int	Max number of times to retry. Default value is 3
retryInter valMilliseconds (?)	int	Time interval in milliseconds on each retry. Default value is 500 milliseconds.
ti meToLiveSeconds (?)	int	Time to live in seconds of ephemeral keys. Default value is 60 seconds.

ShardingSphere-2.x

Readwrite-splitting

concept

In order to relieve the pressure on the database, the write and read operations are separated into different data sources. The write library is called the master library, and the read library is called the slave library. One master library can be configured with multiple slave libraries.

Supported

1. Provides a readwrite-splitting configuration with one master and multiple slaves, which can be used independently or with sub-databases and sub-meters.
2. Independent use of readwrite-splitting to support SQL transparent transmission.
3. In the same thread and the same database connection, if there is a write operation, subsequent read operations will be read from the main library to ensure data consistency.
4. Spring namespace.
5. Hint-based mandatory main library routing.

Unsupported

1. Data synchronization between the master library and the slave library.
2. Data inconsistency caused by the data synchronization delay of the master library and the slave library.
3. Double writing or multiple writing in the main library.

Code development example

only readwrite-splitting

```
// Constructing a readwrite-splitting data source, the readwrite-splitting data
source implements the DataSource interface, which can be directly processed as a
data source. masterDataSource, slaveDataSource0, slaveDataSource1, etc. are real
data sources configured using connection pools such as DBCP
Map<String, DataSource> dataSourceMap = new HashMap<>();
dataSourceMap.put("masterDataSource", masterDataSource);
dataSourceMap.put("slaveDataSource0", slaveDataSource0);
dataSourceMap.put("slaveDataSource1", slaveDataSource1);

// Constructing readwrite-splitting configuration
MasterSlaveRuleConfiguration masterSlaveRuleConfig = new
MasterSlaveRuleConfiguration();
```

```
masterSlaveRuleConfig.setName("ms_ds");
masterSlaveRuleConfig.setMasterDataSourceName("masterDataSource");
masterSlaveRuleConfig.getSlaveDataSourceNames().add("slaveDataSource0");
masterSlaveRuleConfig.getSlaveDataSourceNames().add("slaveDataSource1");

DataSource dataSource = MasterSlaveDataSourceFactory.
createDataSource(dataSourceMap, masterSlaveRuleConfig);
```

sharding table and database + readwrite-splitting

```
// Constructing a readwrite-splitting data source, the readwrite-splitting data
source implements the DataSource interface, which can be directly processed as a
data source. masterDataSource, slaveDataSource0, slaveDataSource1, etc. are real
data sources configured using connection pools such as DBCP
Map<String, DataSource> dataSourceMap = new HashMap<>();
dataSourceMap.put("masterDataSource0", masterDataSource0);
dataSourceMap.put("slaveDataSource00", slaveDataSource00);
dataSourceMap.put("slaveDataSource01", slaveDataSource01);

dataSourceMap.put("masterDataSource1", masterDataSource1);
dataSourceMap.put("slaveDataSource10", slaveDataSource10);
dataSourceMap.put("slaveDataSource11", slaveDataSource11);

// Constructing readwrite-splitting configuration
MasterSlaveRuleConfiguration masterSlaveRuleConfig0 = new
MasterSlaveRuleConfiguration();
masterSlaveRuleConfig0.setName("ds_0");
masterSlaveRuleConfig0.setMasterDataSourceName("masterDataSource0");
masterSlaveRuleConfig0.getSlaveDataSourceNames().add("slaveDataSource00");
masterSlaveRuleConfig0.getSlaveDataSourceNames().add("slaveDataSource01");

MasterSlaveRuleConfiguration masterSlaveRuleConfig1 = new
MasterSlaveRuleConfiguration();
masterSlaveRuleConfig1.setName("ds_1");
masterSlaveRuleConfig1.setMasterDataSourceName("masterDataSource1");
masterSlaveRuleConfig1.getSlaveDataSourceNames().add("slaveDataSource10");
masterSlaveRuleConfig1.getSlaveDataSourceNames().add("slaveDataSource11");

// Continue to create ShardingDataSource through ShardingSlaveDataSourceFactory
ShardingRuleConfiguration shardingRuleConfig = new ShardingRuleConfiguration();
shardingRuleConfig.getMasterSlaveRuleConfigs().add(masterSlaveRuleConfig0);
shardingRuleConfig.getMasterSlaveRuleConfigs().add(masterSlaveRuleConfig1);

DataSource dataSource = ShardingDataSourceFactory.createDataSource(dataSourceMap,
shardingRuleConfig);
```

ShardingSphere-1.x

Readwrite-splitting

concept

In order to relieve the pressure on the database, the write and read operations are separated into different data sources. The write library is called the master library, and the read library is called the slave library. One master library can be configured with multiple slave libraries.

Supported

1. Provides a readwrite-splitting configuration with one master and multiple slaves, which can be used independently or with sub-databases and sub-meters.
2. In the same thread and the same database connection, if there is a write operation, subsequent read operations will be read from the main library to ensure data consistency.
3. Spring namespace.
4. Hint-based mandatory main library routing.

Unsupported

1. Data synchronization between the master library and the slave library.
2. Data inconsistency caused by the data synchronization delay of the master library and the slave library.
3. Double writing or multiple writing in the main library.

Code development example

```
// Constructing a readwrite-splitting data source, the readwrite-splitting data
source implements the DataSource interface, which can be directly processed as a
data source. masterDataSource, slaveDataSource0, slaveDataSource1, etc. are real
data sources configured using connection pools such as DBCP
Map<String, DataSource> slaveDataSourceMap0 = new HashMap<>();
slaveDataSourceMap0.put("slaveDataSource00", slaveDataSource00);
slaveDataSourceMap0.put("slaveDataSource01", slaveDataSource01);
// You can choose the master-slave library load balancing strategy, the default is
ROUND_ROBIN, and there is RANDOM to choose from, or customize the load strategy
DataSource masterSlaveDs0 = MasterSlaveDataSourceFactory.createDataSource("ms_0",
"masterDataSource0", masterDataSource0, slaveDataSourceMap0,
MasterSlaveLoadBalanceStrategyType.ROUND_ROBIN);

Map<String, DataSource> slaveDataSourceMap1 = new HashMap<>();
```

```
slaveDataSourceMap1.put("slaveDataSource10", slaveDataSource10);
slaveDataSourceMap1.put("slaveDataSource11", slaveDataSource11);
DataSource masterSlaveDs1 = MasterSlaveDataSourceFactory.createDataSource("ms_1",
"masterDataSource1", masterDataSource1, slaveDataSourceMap1,
MasterSlaveLoadBalanceStrategyType.ROUND_ROBIN);

// Constructing readwrite-splitting configuration
Map<String, DataSource> dataSourceMap = new HashMap<>();
dataSourceMap.put("ms_0", masterSlaveDs0);
dataSourceMap.put("ms_1", masterSlaveDs1);
```

Spring namespace configuration change history

ShardingSphere-5.0.0-beta

Sharding

Configuration Item Explanation

Namespace: http://shardingsphere.apache.org/schema/shardingsphere/sharding/sharding-5.0.0.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.

Readwrite-Splitting

Configuration Item Explanation

Namespace: http://shardingsphere.apache.org/schema/shardingsphere/readwrite-splitting/readwrite-splitting-5.0.0.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	*Ty pe*	*Description*
id	Att rib ute	Readwrite-splitting data source rule name
write-da ta-source-name	Att rib ute	Write data source name
read-dat a-source-names	Att rib ute	Read data source names, multiple data source names separated with comma
load-balance - algorithm-ref	Att rib ute	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 Use Norms for more details about query consistent routing.

Encryption

Configuration Item Explanation

Namespace: http://shardingsphere.apache.org/schema/shardingsphere/encrypt/encrypt-5.0.0.xsd

<encrypt:rule />

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

<encrypt:table />

Name	Type	Description
name	Attribute	Encrypt table name
column (+)	Tag	Encrypt column configuration

<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.

Shadow-DB

Configuration Item Explanation

Namespace: http://shardingsphere.apache.org/schema/shardingsphere/shadow/shadow-5.0.0.xsd

<shadow:rule />

Name	*Type*	*Description*
id	Attribute	Spring Bean Id
column	Attribute	Shadow column name
mappings(?)	Tag	Mapping relationship between production database and shadow database

<shadow:mapping />

Name	*Type*	*Description*
product-data-source-name	Attribute	Production database name
shadow-data-source-name	Attribute	Shadow database name

4.x

Sharding

Configuration Item Explanation

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

<sharding:data-source />

Name	*Type*	*Description*
id	Attribute	Spring Bean Id
sharding-rule	Tag	Sharding rule configuration
props (?)	Tag	Properties

<sharding:sharding-rule />

Name	*T ype*	*Description*
data-source-names	A ttri bute	Data source Bean list with comma separating multiple Beans
table-rules	Tag	Configuration objects of table sharding rules
binding-table-rules (?)	Tag	Binding table rule list
broadcast-table-rules (?)	Tag	Broadcast table rule list
default-data-source-name (?)	A ttri bute	Tables without sharding rules will be located through default data source
def ault-database-strategy-ref (?)	A ttri bute	Default database sharding strategy, which corresponds to id of ; default means the database is not split
default-table-strategy-ref (?)	A ttri bute	Default table sharding strategy,which corresponds to id of ; default means the database is not split
default-key-generator (?)	A ttri bute	Default key generator configuration, use user-defined ones or built-in ones, e.g. SNOWFLAKE/UUID. Default key generator is `org.apache.sha rding-sphere.core.keygen.generat or.impl.SnowflakeKeyGenerator`
encrypt-rule (?)	Tag	Encrypt rule

<sharding:table-rules />

Name	*Type*	*Description*
table-rule (+)	Tag	Configuration objects of table sharding rules

<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 sharding databases only.
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
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	A ttribute	Standard sharding strategy name
sharding-column	A ttribute	Sharding column name
preci se-algorithm-ref (?)	A ttribute	Precise algorithm reference, applied in `=` and `IN`; the class needs to implement `PreciseShardingAlgorithm` interface
ran ge-algorithm-ref (?)	A ttribute	Range algorithm reference, applied in `BETWEEN`; the class needs to implement `RangeShardingAlgorithm` interface

<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	Complex sharding algorithm reference; the class needs to implement `ComplexKeysShardingAlgorithm` interface

<sharding:inline-strategy />

Name	*T ype*	*Description*
id	A ttri bute	Spring Bean Id
shardi ng-columns	A ttri bute	Sharding column names, multiple columns separated with comma
alg orithm-ref	A ttri bute	Sharding algorithm inline expression, which needs to conform to groovy statements

<sharding:hint-database-strategy />

Name	*Type*	*Description*
id	Attribute	Hint sharding strategy name
algorithm-ref	Attribute	Hint sharding algorithm; the class needs to implement `HintShardingAlgorithm` interface

<sharding:none-strategy />

Name	*Type*	*Description*
id	Attribute	Spring Bean Id

<sharding:key-generator />

Name	*Type*	*Description*
column	Attribute	Auto-increment column name
type	Attribute	Auto-increment key generator `Type`; self-defined generator or internal Type generator (SNOWFLAKE/UUID) can both be selected
props-ref	Attribute	The Property configuration reference of key generators

Properties Property configuration that can include these properties of these key generators.

SNOWFLAKE | *Name* | *Data Type* | *Explanation* | | ———- | ——| ——————————| | worker.id (?) | long | The unique id for working machine, the default value is `0` | | max.tolerate.time.difference.milliseconds (?) | long | The max tolerate time for different server' s time difference in milliseconds, the default value is `10` | | max.vibration.offset (?) | int | The max upper limit value of vibrate number, range `[0, 4096)`, the default value is `1`. 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` |

<sharding:encrypt-rules />

Name	*Type*	*Description*
encryptor-rule (+)	Tag	Encryptor rule

<sharding:encrypt-rule />

Name	*Type*	*Description*
encrypt:encrypt-rule(?)	Tag	Encrypt rule

<sharding:props />

Name	*Type*	*Description*
sql.show (?)	Attribute	Show SQL or not; default value: false
exec utor.size (?)	Attribute	Executing thread number; default value: CPU core number
max .connecti ons.size. per.query (?)	Attribute	The maximum connection number that each physical database allocates to each query; default value: 1
c heck.tabl e.metadat a.enabled (?)	Attribute	Whether to check meta-data consistency of sharding table when it initializes; default value: false
query. with.ciph er.column (?)	Attribute	When there is a plainColumn, use cipherColumn or not to query, default value: true

Readwrite-Splitting

Configuration Item Explanation

Namespace: http://shardingsphere.apache.org/schema/shardingsphere/masterslave/master-slave.xsd

<master-slave:data-source />

Name	*Type*	*Explanation*
id	Attribute	Spring Bean id
master-data-source-name	Attribute	Bean id of data source in master database
slave-data-source-names	Attribute	Bean id list of data source in slave database; multiple Beans are separated by commas
strategy-ref (?)	Attribute	Slave database load balance algorithm reference; the class needs to implement `MasterSlaveLoadBalanceAlgorithm` interface
strategy-type (?)	Attribute	Load balance algorithm type of slave database; optional value: ROUND_ROBIN and RANDOM; if there is `load-balance-algorithm-class-name`, the configuration can be omitted
config-map (?)	Tag	Users' self-defined configurations
props (?)	Tag	Attribute configurations

<master-slave:props />

Name	*Type*	*Explanation*
sql.show (?)	Attribute	Show SQL or not; default value: false
executor.size (?)	Attribute	Executing thread number; default value: CPU core number
max.connections.size.per.query (?)	Attribute	The maximum connection number that each physical database allocates to each query; default value: 1
check.table.metadata.enabled (?)	Attribute	Whether to check meta-data consistency of sharding table when it initializes; default value: false

<master-slave:load-balance-algorithm />

4.0.0-RC2 version added

Name	*T ype*	*Explanation*
id	A ttri bute	Spring Bean Id
type	A ttri bute	Type of load balance algorithm, 'RANDOM' 或' ROUND_ROBIN', support custom extension
p rops-ref (?)	A ttri bute	Properties of load balance algorithm

Data Masking

Configuration Item Explanation

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

<encrypt:data-source />

Name	*Type*	*Type*
id	Attribute	Spring Bean Id
data-source-name	Attribute	Encrypt data source Bean Id
props (?)	Tag	Attribute configurations

<encrypt:encryptors />

Name	*Type*	*Type*
encryptor(+)	Tag	Encryptor configuration

<encrypt:encryptor />

Name	*Type*	*Type*
id	Attribute	Names of Encryptor
type	Attribute	Types of Encryptor, including MD5/AES or customize type
props-re	Attribute	Attribute configurations

<encrypt:tables />

Name	*Type*	*Type*
table(+)	Tag	Encrypt table configuration

<encrypt:table />

Name	*Type*	*Type*
column(+)	Tag	Encrypt column configuration

<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:props />

Name	*Type*	*Description*
sql.show (?)	Attribute	Show SQL or not; default value: false
query.with.cipher.column (?)	Attribute	When there is a plainColumn, use cipherColumn or not to query, default value: true

Orchestration

Data Sharding + Orchestration

Namespace: http://shardingsphere.apache.org/schema/shardingsphere/orchestration/orchestration.xsd

<orchestration:master-slave-data-source />

Name	*Type*	*Description*
id	Attribute	Id
data-source-ref (?)	Attribute	Orchestrated database Id
registry-center-ref	Attribute	Registry center Id
overwrite	Attribute	Whether to overwrite local configurations with registry center configurations; if it can, each initialization should refer to local configurations; default means not to overwrite

Read-Write Split + Orchestration

Namespace: http://shardingsphere.apache.org/schema/shardingsphere/orchestration/orchestration.xsd

<orchestration:sharding-data-source />

Name	*Type*	*Description*
id	Attribute	Id
dat a-source-ref (?)	Attribute	Orchestrated database Id
registr y-center-ref	Attribute	Registry center Id
overwrite	Attribute	Use local configuration to overwrite registry center or not

Data Masking + Orchestration

Namespace: http://shardingsphere.apache.org/schema/shardingsphere/orchestration/orchestration.xsd

<orchestration:encrypt-data-source />

Name	*Type*	*Description*
id	Attribute	Id
dat a-source-ref (?)	Attribute	Orchestrated database Id
registr y-center-ref	Attribute	Registry center Id
overwrite	Attribute	Use local configuration to overwrite registry center or not

Orchestration registry center

Namespace: http://shardingsphere.apache.org/schema/shardingsphere/orchestration/orchestration.xsd

<orchestration:registry-center />

Name	*Type*	*Description*
id	Attribute	Spring Bean Id of registry center
type	Attribute	Registry center type. Example:zookeeper
server-lists	Attribute	Registry servers list, multiple split as comma. Example: host1:2181,host2:2181
namespace (?)	Attribute	Namespace of registry
digest (?)	Attribute	Digest for registry. Default is not need digest
operat ion-timeout- milliseconds (?)	Attribute	Operation timeout time in milliseconds, default value is 500 seconds
max-retries (?)	Attribute	Max number of times to retry, default value is 3
ret ry-interval- milliseconds (?)	Attribute	Time interval in milliseconds on each retry, default value is 500 milliseconds
time-to- live-seconds (?)	Attribute	Living time of temporary nodes; default value: 60 seconds
props-ref (?)	Attribute	Other customize properties of registry center

3.x

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

Sharding

Configuration Item Explanation

```
<?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:p="http://www.springframework.org/schema/p"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:tx="http://www.springframework.org/schema/tx"
       xmlns:sharding="http://shardingsphere.io/schema/shardingsphere/sharding"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
                        http://www.springframework.org/schema/beans/spring-beans.
xsd
                        http://shardingsphere.io/schema/shardingsphere/sharding
                        http://shardingsphere.io/schema/shardingsphere/sharding/
sharding.xsd
```

```
                        http://www.springframework.org/schema/context
                        http://www.springframework.org/schema/context/spring-
context.xsd
                        http://www.springframework.org/schema/tx
                        http://www.springframework.org/schema/tx/spring-tx.xsd">
    <context:annotation-config />
    <context:component-scan base-package="io.shardingsphere.example.spring.
namespace.jpa" />

    <bean id="entityManagerFactory" class="org.springframework.orm.jpa.
LocalContainerEntityManagerFactoryBean">
        <property name="dataSource" ref="shardingDataSource" />
        <property name="jpaVendorAdapter">
            <bean class="org.springframework.orm.jpa.vendor.
HibernateJpaVendorAdapter" p:database="MYSQL" />
        </property>
        <property name="packagesToScan" value="io.shardingsphere.example.spring.
namespace.jpa.entity" />
        <property name="jpaProperties">
            <props>
                <prop key="hibernate.dialect">org.hibernate.dialect.MySQLDialect</
prop>
                <prop key="hibernate.hbm2ddl.auto">create</prop>
                <prop key="hibernate.show_sql">true</prop>
            </props>
        </property>
    </bean>
    <bean id="transactionManager" class="org.springframework.orm.jpa.
JpaTransactionManager" p:entityManagerFactory-ref="entityManagerFactory" />
    <tx:annotation-driven />

    <bean id="ds0" class="org.apache.commons.dbcp.BasicDataSource" destroy-method=
"close">
        <property name="driverClassName" value="com.mysql.jdbc.Driver" />
        <property name="url" value="jdbc:mysql://localhost:3306/ds0" />
        <property name="username" value="root" />
        <property name="password" value="" />
    </bean>

    <bean id="ds1" class="org.apache.commons.dbcp.BasicDataSource" destroy-method=
"close">
        <property name="driverClassName" value="com.mysql.jdbc.Driver" />
        <property name="url" value="jdbc:mysql://localhost:3306/ds1" />
        <property name="username" value="root" />
        <property name="password" value="" />
    </bean>

    <bean id="preciseModuloDatabaseShardingAlgorithm" class="io.shardingsphere.
example.spring.namespace.jpa.algorithm.PreciseModuloDatabaseShardingAlgorithm" />
```

```
    <bean id="preciseModuloTableShardingAlgorithm" class="io.shardingsphere.
example.spring.namespace.jpa.algorithm.PreciseModuloTableShardingAlgorithm" />

    <sharding:standard-strategy id="databaseShardingStrategy" sharding-column=
"user_id" precise-algorithm-ref="preciseModuloDatabaseShardingAlgorithm" />
    <sharding:standard-strategy id="tableShardingStrategy" sharding-column="order_
id" precise-algorithm-ref="preciseModuloTableShardingAlgorithm" />

    <sharding:data-source id="shardingDataSource">
        <sharding:sharding-rule data-source-names="ds0,ds1">
            <sharding:table-rules>
                <sharding:table-rule logic-table="t_order" actual-data-nodes="ds$->
{0..1}.t_order$->{0..1}" database-strategy-ref="databaseShardingStrategy" table-
strategy-ref="tableShardingStrategy" generate-key-column-name="order_id" />
                <sharding:table-rule logic-table="t_order_item" actual-data-nodes=
"ds$->{0..1}.t_order_item$->{0..1}" database-strategy-ref="databaseShardingStrategy
" table-strategy-ref="tableShardingStrategy" generate-key-column-name="order_item_
id" />
            </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_config" />
            </sharding:broadcast-table-rules>
        </sharding:sharding-rule>
    </sharding:data-source>
</beans>
```

Readwrite-splitting

Configuration Item Explanation

```
<?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:context="http://www.springframework.org/schema/context"
       xmlns:p="http://www.springframework.org/schema/p"
       xmlns:tx="http://www.springframework.org/schema/tx"
       xmlns:master-slave="http://shardingsphere.io/schema/shardingsphere/
masterslave"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
                        http://www.springframework.org/schema/beans/spring-beans.
xsd
                        http://www.springframework.org/schema/context
```

```
                        http://www.springframework.org/schema/context/spring-
context.xsd
                        http://www.springframework.org/schema/tx
                        http://www.springframework.org/schema/tx/spring-tx.xsd
                        http://shardingsphere.io/schema/shardingsphere/masterslave
                        http://shardingsphere.io/schema/shardingsphere/masterslave/
master-slave.xsd">
    <context:annotation-config />
    <context:component-scan base-package="io.shardingsphere.example.spring.
namespace.jpa" />

    <bean id="entityManagerFactory" class="org.springframework.orm.jpa.
LocalContainerEntityManagerFactoryBean">
        <property name="dataSource" ref="masterSlaveDataSource" />
        <property name="jpaVendorAdapter">
            <bean class="org.springframework.orm.jpa.vendor.
HibernateJpaVendorAdapter" p:database="MYSQL" />
        </property>
        <property name="packagesToScan" value="io.shardingsphere.example.spring.
namespace.jpa.entity" />
        <property name="jpaProperties">
            <props>
                <prop key="hibernate.dialect">org.hibernate.dialect.MySQLDialect</
prop>
                <prop key="hibernate.hbm2ddl.auto">create</prop>
                <prop key="hibernate.show_sql">true</prop>
            </props>
        </property>
    </bean>
    <bean id="transactionManager" class="org.springframework.orm.jpa.
JpaTransactionManager" p:entityManagerFactory-ref="entityManagerFactory" />
    <tx:annotation-driven />

    <bean id="ds_master" class="org.apache.commons.dbcp.BasicDataSource" destroy-
method="close">
        <property name="driverClassName" value="com.mysql.jdbc.Driver" />
        <property name="url" value="jdbc:mysql://localhost:3306/ds_master" />
        <property name="username" value="root" />
        <property name="password" value="" />
    </bean>

    <bean id="ds_slave0" class="org.apache.commons.dbcp.BasicDataSource" destroy-
method="close">
        <property name="driverClassName" value="com.mysql.jdbc.Driver" />
        <property name="url" value="jdbc:mysql://localhost:3306/ds_slave0" />
        <property name="username" value="root" />
        <property name="password" value="" />
    </bean>
```

```
    <bean id="ds_slave1" class="org.apache.commons.dbcp.BasicDataSource" destroy-
method="close">
        <property name="driverClassName" value="com.mysql.jdbc.Driver" />
        <property name="url" value="jdbc:mysql://localhost:3306/ds_slave1" />
        <property name="username" value="root" />
        <property name="password" value="" />
    </bean>

    <bean id="randomStrategy" class="io.shardingsphere.api.algorithm.masterslave.
RandomMasterSlaveLoadBalanceAlgorithm" />
    <master-slave:data-source id="masterSlaveDataSource" master-data-source-name=
"ds_master" slave-data-source-names="ds_slave0, ds_slave1" strategy-ref=
"randomStrategy">
            <master-slave:props>
                <prop key="sql.show">${sql_show}</prop>
                <prop key="executor.size">10</prop>
                <prop key="foo">bar</prop>
            </master-slave:props>
    </master-slave:data-source>
</beans>
```

Orchestration

Configuration Item Explanation

```
<?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:sharding="http://shardingsphere.io/schema/shardingsphere/
orchestration/sharding"
       xmlns:master-slave="http://shardingsphere.io/schema/shardingsphere/
orchestration/masterslave"
       xmlns:reg="http://shardingsphere.io/schema/shardingsphere/orchestration/reg"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
                           http://www.springframework.org/schema/beans/spring-
beans.xsd
                           http://shardingsphere.io/schema/shardingsphere/
orchestration/reg
                           http://shardingsphere.io/schema/shardingsphere/
orchestration/reg/reg.xsd
                           http://shardingsphere.io/schema/shardingsphere/
orchestration/sharding
                           http://shardingsphere.io/schema/shardingsphere/
orchestration/sharding/sharding.xsd
                           http://shardingsphere.io/schema/shardingsphere/
orchestration/masterslave
```

```
                            http://shardingsphere.io/schema/shardingsphere/
orchestration/masterslave/master-slave.xsd">

    <reg:registry-center id="regCenter" server-lists="localhost:2181" namespace=
"orchestration-spring-namespace-demo" overwtite="false" />
    <sharding:data-source id="shardingMasterSlaveDataSource" registry-center-ref=
"regCenter" />
    <master-slave:data-source id="masterSlaveDataSource" registry-center-ref=
"regCenter" />
</beans>
```

2.x

Readwrite-splitting

The configuration example for Spring namespace

```
<?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:context="http://www.springframework.org/schema/context"
    xmlns:sharding="http://shardingsphere.io/schema/shardingjdbc/sharding"
    xmlns:masterslave="http://shardingsphere.io/schema/shardingjdbc/masterslave"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
                        http://www.springframework.org/schema/beans/spring-beans.
xsd
                        http://www.springframework.org/schema/context
                        http://www.springframework.org/schema/context/spring-
context.xsd
                        http://shardingsphere.io/schema/shardingjdbc/sharding
                        http://shardingsphere.io/schema/shardingjdbc/sharding/
sharding.xsd
                        http://shardingsphere.io/schema/shardingjdbc/masterslave
                        http://shardingsphere.io/schema/shardingjdbc/masterslave/
master-slave.xsd
                        ">
    <!-- Actual source data Configuration -->
    <bean id="dbtbl_0_master" class="org.apache.commons.dbcp.BasicDataSource"
destroy-method="close">
        <property name="driverClassName" value="com.mysql.jdbc.Driver"/>
        <property name="url" value="jdbc:mysql://localhost:3306/dbtbl_0_master"/>
        <property name="username" value="root"/>
        <property name="password" value=""/>
    </bean>

    <bean id="dbtbl_0_slave_0" class="org.apache.commons.dbcp.BasicDataSource"
destroy-method="close">
```

```
        <property name="driverClassName" value="com.mysql.jdbc.Driver"/>
        <property name="url" value="jdbc:mysql://localhost:3306/dbtbl_0_slave_0"/>
        <property name="username" value="root"/>
        <property name="password" value=""/>
    </bean>

    <bean id="dbtbl_0_slave_1" class="org.apache.commons.dbcp.BasicDataSource"
destroy-method="close">
        <property name="driverClassName" value="com.mysql.jdbc.Driver"/>
        <property name="url" value="jdbc:mysql://localhost:3306/dbtbl_0_slave_1"/>
        <property name="username" value="root"/>
        <property name="password" value=""/>
    </bean>

    <bean id="dbtbl_1_master" class="org.apache.commons.dbcp.BasicDataSource"
destroy-method="close">
        <property name="driverClassName" value="com.mysql.jdbc.Driver"/>
        <property name="url" value="jdbc:mysql://localhost:3306/dbtbl_1_master"/>
        <property name="username" value="root"/>
        <property name="password" value=""/>
    </bean>

    <bean id="dbtbl_1_slave_0" class="org.apache.commons.dbcp.BasicDataSource"
destroy-method="close">
        <property name="driverClassName" value="com.mysql.jdbc.Driver"/>
        <property name="url" value="jdbc:mysql://localhost:3306/dbtbl_1_slave_0"/>
        <property name="username" value="root"/>
        <property name="password" value=""/>
    </bean>

    <bean id="dbtbl_1_slave_1" class="org.apache.commons.dbcp.BasicDataSource"
destroy-method="close">
        <property name="driverClassName" value="com.mysql.jdbc.Driver"/>
        <property name="url" value="jdbc:mysql://localhost:3306/dbtbl_1_slave_1"/>
        <property name="username" value="root"/>
        <property name="password" value=""/>
    </bean>

    <!-- Readwrite-splitting DataSource Configuration -->
    <master-slave:data-source id="dbtbl_0" master-data-source-name="dbtbl_0_master"
slave-data-source-names="dbtbl_0_slave_0, dbtbl_0_slave_1" strategy-type="ROUND_
ROBIN" />
    <master-slave:data-source id="dbtbl_1" master-data-source-name="dbtbl_1_master"
slave-data-source-names="dbtbl_1_slave_0, dbtbl_1_slave_1" strategy-type="ROUND_
ROBIN" />

    <sharding:inline-strategy id="databaseStrategy" sharding-column="user_id"
algorithm-expression="dbtbl_${user_id % 2}" />
```

```
    <sharding:inline-strategy id="orderTableStrategy" sharding-column="order_id"
algorithm-expression="t_order_${order_id % 4}" />

    <sharding:data-source id="shardingDataSource">
        <sharding:sharding-rule data-source-names="dbtbl_0, dbtbl_1">
            <sharding:table-rules>
                <sharding:table-rule logic-table="t_order" actual-data-nodes=
"dbtbl_${0..1}.t_order_${0..3}" database-strategy-ref="databaseStrategy" table-
strategy-ref="orderTableStrategy"/>
            </sharding:table-rules>
        </sharding:sharding-rule>
    </sharding:data-source>
</beans>
```

Spring Boot Starter Configuration

5.0.0-beta

Sharding

Configuration Item Explanation

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

# For multiple sharding columns scenario
spring.shardingsphere.rules.sharding.tables.<table-name>.database-strategy.complex.
<sharding-algorithm-name>.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 algorithm name
```

```
# Sharding by hint
spring.shardingsphere.rules.sharding.tables.<table-name>.database-strategy.hint.
<sharding-algorithm-name>.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= # 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.sharding.default-database-strategy.xxx= # Default strategy
for database sharding
spring.shardingsphere.sharding.default-table-strategy.xxx= # Default strategy for
table sharding
spring.shardingsphere.sharding.default-key-generate-strategy.xxx= # Default Key
generator strategy

# 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 keygen Algorithm List。

Readwrite-splitting

Configuration Item Explanation

```
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>.primary-data-source-name= # Write data source name
spring.shardingsphere.rules.readwrite-splitting.data-sources.<readwrite-splitting-
data-source-name>.replica-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
```

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

Encryption

Configuration Item Explanation

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

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
```

Shadow DB

Configuration Item Explanation

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

spring.shardingsphere.rules.shadow.column= # Shadow column name
spring.shardingsphere.rules.shadow.shadow-mappings.<product-data-source-name>= #
Shadow data source name
```

Governance

Configuration Item Explanation

Management

```
spring.shardingsphere.governance.name= # Governance name
spring.shardingsphere.governance.registry-center.type= # Governance instance type.
Example:Zookeeper, etcd, Apollo, Nacos
spring.shardingsphere.governance.registry-center.server-lists= # The list of
servers that connect to governance instance, including IP and port number; use
commas to separate
spring.shardingsphere.governance.registry-center.props= # Other properties
spring.shardingsphere.governance.overwrite= # Whether to overwrite local
configurations with config center configurations; if it can, each initialization
should refer to local configurations
```

Mixed Rules

Configuration Item Explanation

```
# data source configuration
spring.shardingsphere.datasource.names= write-ds0,write-ds1,write-ds0-read0,write-
ds1-read0

spring.shardingsphere.datasource.write-ds0.url= # Database URL connection
spring.shardingsphere.datasource.write-ds0.type=  # Database connection pool type
name
```

```
spring.shardingsphere.datasource.write-ds0.driver-class-name= # Database driver
class name
spring.shardingsphere.datasource.write-ds0.username= # Database username
spring.shardingsphere.datasource.write-ds0.password= # Database password
spring.shardingsphere.datasource.write-ds0.xxx=  # Other properties of database
connection pool

spring.shardingsphere.datasource.write-ds1.url= # Database URL connection
# ...Omit specific configuration.

spring.shardingsphere.datasource.write-ds0-read0.url= # Database URL connection
# ...Omit specific configuration.

spring.shardingsphere.datasource.write-ds1-read0.url= # Database URL connection
# ...Omit specific configuration.

# Sharding rules configuration
# Databases sharding strategy
spring.shardingsphere.rules.sharding.default-database-strategy.standard.sharding-
column=user_id
spring.shardingsphere.rules.sharding.default-database-strategy.standard.sharding-
algorithm-name=default-database-strategy-inline
# Binding table rules configuration ,and multiple groups of binding-tables
configured with arrays
spring.shardingsphere.rules.sharding.binding-tables[0]=t_user,t_user_detail
spring.shardingsphere.rules.sharding.binding-tables[1]= # Binding table names,
multiple table name are separated by commas
spring.shardingsphere.rules.sharding.binding-tables[x]= # Binding table names,
multiple table name are separated by commas
# Broadcast table rules configuration
spring.shardingsphere.rules.sharding.broadcast-tables= # Broadcast table names,
multiple table name are separated by commas

# Table sharding strategy
# The enumeration value of `ds_$->{0..1}` is the name of the logical data source
configured with readwrite-splitting
spring.shardingsphere.rules.sharding.tables.t_user.actual-data-nodes=ds_$->{0..1}.
t_user_$->{0..1}
spring.shardingsphere.rules.sharding.tables.t_user.table-strategy.standard.
sharding-column=user_id
spring.shardingsphere.rules.sharding.tables.t_user.table-strategy.standard.
sharding-algorithm-name=user-table-strategy-inline

# Data encrypt configuration
# Table `t_user` is the name of the logical table that uses for data sharding
configuration.
spring.shardingsphere.rules.encrypt.tables.t_user.columns.user_name.cipher-
column=user_name
```

```
spring.shardingsphere.rules.encrypt.tables.t_user.columns.user_name.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

# Data encrypt algorithm configuration
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

# Key generate strategy configuration
spring.shardingsphere.rules.sharding.tables.t_user.key-generate-strategy.
column=user_id
spring.shardingsphere.rules.sharding.tables.t_user.key-generate-strategy.key-
generator-name=snowflake

# Sharding algorithm configuration
spring.shardingsphere.rules.sharding.sharding-algorithms.default-database-strategy-
inline.type=INLINE
# The enumeration value of `ds_$->{user_id % 2}` is the name of the logical data
source configured with readwrite-splitting
spring.shardingsphere.rules.sharding.sharding-algorithms.default-database-strategy-
inline.algorithm-expression=ds$->{user_id % 2}
spring.shardingsphere.rules.sharding.sharding-algorithms.user-table-strategy-
inline.type=INLINE
spring.shardingsphere.rules.sharding.sharding-algorithms.user-table-strategy-
inline.algorithm-expression=t_user_$->{user_id % 2}

# Key generate algorithm configuration
spring.shardingsphere.rules.sharding.key-generators.snowflake.type=SNOWFLAKE
spring.shardingsphere.rules.sharding.key-generators.snowflake.props.worker-id=123

# read query configuration
# ds_0,ds_1 is the logical data source name of the readwrite-splitting
spring.shardingsphere.rules.readwrite-splitting.data-sources.ds_0.write-data-
source-name=write-ds0
spring.shardingsphere.rules.readwrite-splitting.data-sources.ds_0.read-data-source-
names=write-ds0-read0
spring.shardingsphere.rules.readwrite-splitting.data-sources.ds_0.load-balancer-
name=read-random
spring.shardingsphere.rules.readwrite-splitting.data-sources.ds_1.write-data-
source-name=write-ds1
spring.shardingsphere.rules.readwrite-splitting.data-sources.ds_1.read-data-source-
names=write-ds1-read0
```

```
spring.shardingsphere.rules.readwrite-splitting.data-sources.ds_1.load-balancer-
name=read-random

# Load balance algorithm configuration
spring.shardingsphere.rules.readwrite-splitting.load-balancers.read-random.
type=RANDOM
```

Shardingsphere-4.x

Data Sharding

Configuration Item Explanation

```
spring.shardingsphere.datasource.names= #Data source name; multiple data sources
are separated by commas

spring.shardingsphere.datasource.<data-source-name>.type= #Database connection pool
type name
spring.shardingsphere.datasource.<data-source-name>.driver-class-name= #Database
driver class name
spring.shardingsphere.datasource.<data-source-name>.url= #Database url connection
spring.shardingsphere.datasource.<data-source-name>.username= #Database username
spring.shardingsphere.datasource.<data-source-name>.password= #Database password
spring.shardingsphere.datasource.<data-source-name>.xxx= #Other properties of
database connection pool

spring.shardingsphere.sharding.tables.<logic-table-name>.actual-data-nodes= #It is
consisted of data source name + table name, separated by decimal points; multiple
tables are separated by commas and support inline expressions; default means using
existing data sources and logic table names to generate data nodes; it can be
applied in broadcast tables (each database needs a same table for relevance query,
dictionary table mostly) or the situation with sharding database but without
sharding table (table structures of all the databases are consistent)

#Database sharding strategy; default means using default database sharding
strategy; it can only choose one of the following sharding strategies

#It is applied in standard sharding situation of single-sharding key
spring.shardingsphere.sharding.tables.<logic-table-name>.database-strategy.
standard.sharding-column= #Sharding column name
spring.shardingsphere.sharding.tables.<logic-table-name>.database-strategy.
standard.precise-algorithm-class-name= #Precise algorithm class name, applied in =
and IN; the class needs to implement PreciseShardingAlgorithm interface and provide
parameter-free constructor
spring.shardingsphere.sharding.tables.<logic-table-name>.database-strategy.
standard.range-algorithm-class-name= #Range sharding algorithm class name, applied
in BETWEEN, optional; the class should implement RangeShardingAlgorithm interface
and provide parameter-free constructor
```

```
#It is applied in complex sharding situations with multiple sharding keys
spring.shardingsphere.sharding.tables.<logic-table-name>.database-strategy.complex.
sharding-columns= #Sharding column name, with multiple columns separated by commas
spring.shardingsphere.sharding.tables.<logic-table-name>.database-strategy.complex.
algorithm-class-name= #Complex sharding algorithm class name; the class needs to
implement ComplexKeysShardingAlgorithm interface and provide parameter-free
constructor

#Inline expression sharding strategy
spring.shardingsphere.sharding.tables.<logic-table-name>.database-strategy.inline.
sharding-column= #Sharding column name
spring.shardingsphere.sharding.tables.<logic-table-name>.database-strategy.inline.
algorithm-expression= #Inline expression of sharding algorithm, which needs to
conform to groovy statements

#Hint Sharding Strategy
spring.shardingsphere.sharding.tables.<logic-table-name>.database-strategy.hint.
algorithm-class-name= #Hint algorithm class name;  the class needs to implement
HintShardingAlgorithm interface and provide parameter-free constructor

#Table sharding strategy, same as database sharding strategy
spring.shardingsphere.sharding.tables.<logic-table-name>.table-strategy.xxx=
#Omitted

spring.shardingsphere.sharding.tables.<logic-table-name>.key-generator.column=
#Auto-increment column name; default means not using auto-increment key generator
spring.shardingsphere.sharding.tables.<logic-table-name>.key-generator.type= #Auto-
increament key generator type; default means using default auto-increament key
generator; user defined generator or internal generator (SNOWFLAKE, UUID) can both
be selected
spring.shardingsphere.sharding.tables.<logic-table-name>.key-generator.props.
<property-name>= #Properties, Notice: when use SNOWFLAKE, `worker.id` and `max.
tolerate.time.difference.milliseconds` for `SNOWFLAKE` need to be set. To use the
generated value of this algorithm as sharding value, it is recommended to configure
`max.vibration.offset`

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

spring.shardingsphere.sharding.broadcast-tables[0]= #Broadcast table rule list
spring.shardingsphere.sharding.broadcast-tables[1]= #Broadcast table rule list
spring.shardingsphere.sharding.broadcast-tables[x]= #Broadcast table rule list

spring.shardingsphere.sharding.default-data-source-name= #Tables without sharding
rules will be located through default data source
spring.shardingsphere.sharding.default-database-strategy.xxx= #Default database
sharding strategy
```

```
spring.shardingsphere.sharding.default-table-strategy.xxx= #Default table sharding strategy
spring.shardingsphere.sharding.default-key-generator.type= #Default auto-increament key generator of type; it will use org.apache.shardingsphere.core.keygen.generator.impl.SnowflakeKeyGenerator in default; user defined generator or internal generator (SNOWFLAKE or UUID) can both be used
spring.shardingsphere.sharding.default-key-generator.props.<property-name>= #Auto-increament key generator property configuration, such as worker.id and max.tolerate.time.difference.milliseconds of SNOWFLAKE algorithm

spring.shardingsphere.sharding.master-slave-rules.<master-slave-data-source-name>.master-data-source-name= #Refer to readwrite-splitting part for more details
spring.shardingsphere.sharding.master-slave-rules.<master-slave-data-source-name>.slave-data-source-names[0]= #Refer to readwrite-splitting part for more details
spring.shardingsphere.sharding.master-slave-rules.<master-slave-data-source-name>.slave-data-source-names[1]= #Refer to readwrite-splitting part for more details
spring.shardingsphere.sharding.master-slave-rules.<master-slave-data-source-name>.slave-data-source-names[x]= #Refer to readwrite-splitting part for more details
spring.shardingsphere.sharding.master-slave-rules.<master-slave-data-source-name>.load-balance-algorithm-class-name= #Refer to readwrite-splitting part for more details
spring.shardingsphere.sharding.master-slave-rules.<master-slave-data-source-name>.load-balance-algorithm-type= #Refer to readwrite-splitting part for more details

spring.shardingsphere.props.sql.show= #Show SQL or not; default value: false
spring.shardingsphere.props.executor.size= #Executing thread number; default value: CPU core number
```

Readwrite Split

Configuration Item Explanation

```
#Omit data source configurations; keep it consistent with data sharding

spring.shardingsphere.sharding.master-slave-rules.<master-slave-data-source-name>.master-data-source-name= #Data source name of master database
spring.shardingsphere.sharding.master-slave-rules.<master-slave-data-source-name>.slave-data-source-names[0]= #Data source name list of slave database
spring.shardingsphere.sharding.master-slave-rules.<master-slave-data-source-name>.slave-data-source-names[1]= #Data source name list of slave database
spring.shardingsphere.sharding.master-slave-rules.<master-slave-data-source-name>.slave-data-source-names[x]= #Data source name list of slave database
spring.shardingsphere.sharding.master-slave-rules.<master-slave-data-source-name>.load-balance-algorithm-class-name= #Load balance algorithm class name; the class needs to implement MasterSlaveLoadBalanceAlgorithm interface and provide parameter-free constructor
```

```
spring.shardingsphere.sharding.master-slave-rules.<master-slave-data-source-name>.
load-balance-algorithm-type= #Load balance algorithm class of slave database;
optional value: ROUND_ROBIN and RANDOM; if there is load-balance-algorithm-class-
name, the configuration can be omitted

spring.shardingsphere.props.sql.show= #Show SQL or not; default value: false
spring.shardingsphere.props.executor.size= #Executing thread number; default value:
CPU core number
spring.shardingsphere.props.check.table.metadata.enabled= #Whether to check meta-
data consistency of sharding table when it initializes; default value: false
```

Data Masking

Configuration Item Explanation

```
#Omit data source configurations; keep it consistent with data sharding

spring.shardingsphere.encrypt.encryptors.<encryptor-name>.type= #Type of encryptor,
use user-defined ones or built-in ones, e.g. MD5/AES
spring.shardingsphere.encrypt.encryptors.<encryptor-name>.props.<property-name>=
#Properties, Notice: when use AES encryptor, `aes.key.value` for AES encryptor need
to be set
spring.shardingsphere.encrypt.tables.<table-name>.columns.<logic-column-name>.
plainColumn= #Plain column name
spring.shardingsphere.encrypt.tables.<table-name>.columns.<logic-column-name>.
cipherColumn= #Cipher column name
spring.shardingsphere.encrypt.tables.<table-name>.columns.<logic-column-name>.
assistedQueryColumn= #AssistedColumns for query, when use
ShardingQueryAssistedEncryptor, it can help query encrypted data
spring.shardingsphere.encrypt.tables.<table-name>.columns.<logic-column-name>.
encryptor= #Encryptor name
```

Orchestration

Configuration Item Explanation

```
#Omit data source, data sharding, readwrite split and data masking configurations

spring.shardingsphere.orchestration.name= #Orchestration instance name
spring.shardingsphere.orchestration.overwrite= #Whether to overwrite local
configurations with registry center configurations; if it can, each initialization
should refer to local configurations
spring.shardingsphere.orchestration.registry.type= #Registry center type.
Example:zookeeper
spring.shardingsphere.orchestration.registry.server-lists= #The list of servers
that connect to registry center, including IP and port number; use commas to
separate
```

```
spring.shardingsphere.orchestration.registry.namespace= #Registry center namespace
spring.shardingsphere.orchestration.registry.digest= #The token that connects to
the registry center; default means there is no need for authentication
spring.shardingsphere.orchestration.registry.operation-timeout-milliseconds= #The
millisecond number for operation timeout; default value: 500 milliseconds
spring.shardingsphere.orchestration.registry.max-retries= #Maximum retry time after
failing; default value: 3 times
spring.shardingsphere.orchestration.registry.retry-interval-milliseconds= #Interval
time to retry; default value: 500 milliseconds
spring.shardingsphere.orchestration.registry.time-to-live-seconds= #Living time of
temporary nodes; default value: 60 seconds
spring.shardingsphere.orchestration.registry.props= #Customize registry center
props.
```

shardingsphere-3.x

Sharding

Configuration Item Explanation

```
sharding.jdbc.datasource.names= #Names of data sources. Multiple data sources
separated with comma

sharding.jdbc.datasource.<data-source-name>.type= #Class name of data source pool
sharding.jdbc.datasource.<data-source-name>.driver-class-name= #Class name of
database driver
sharding.jdbc.datasource.<data-source-name>.url= #Database URL
sharding.jdbc.datasource.<data-source-name>.username= #Database username
sharding.jdbc.datasource.<data-source-name>.password= #Database password
sharding.jdbc.datasource.<data-source-name>.xxx= #Other properties for data source
pool

sharding.jdbc.config.sharding.tables.<logic-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. Example: ds${0..7}.tbl${0..7}

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

#Standard sharding scenario for single sharding column
sharding.jdbc.config.sharding.tables.<logic-table-name>.database-strategy.standard.
sharding-column= #Name of sharding column
sharding.jdbc.config.sharding.tables.<logic-table-name>.database-strategy.standard.
precise-algorithm-class-name= #Precise algorithm class name used for `=` and `IN`.
This class need to implements PreciseShardingAlgorithm, and require a no argument
constructor
```

```
sharding.jdbc.config.sharding.tables.<logic-table-name>.database-strategy.standard.
range-algorithm-class-name= #Range algorithm class name used for `BETWEEN`. This
class need to implements RangeShardingAlgorithm, and require a no argument
constructor

#Complex sharding scenario for multiple sharding columns
sharding.jdbc.config.sharding.tables.<logic-table-name>.database-strategy.complex.
sharding-columns= #Names of sharding columns. Multiple columns separated with comma
sharding.jdbc.config.sharding.tables.<logic-table-name>.database-strategy.complex.
algorithm-class-name= #Complex sharding algorithm class name. This class need to
implements ComplexKeysShardingAlgorithm, and require a no argument constructor

#Inline expression sharding scenario for si-gle s-arding column
sharding.jdbc.config.sharding.tables.<logic-table-name>.database-strategy.inline.
sharding-column= #Name of sharding column
sharding.jdbc.config.sharding.tables.<logic-table-name>.database-strategy.inline.
algorithm-expression= #Inline expression for sharding algorithm

#Hint sharding strategy
sharding.jdbc.config.sharding.tables.<logic-table-name>.database-strategy.hint.
algorithm-class-name= #Hint sharding algorithm class name. This class need to
implements HintShardingAlgorithm, and require a no argument constructor

#Tables sharding strategy, Same as database- shar-ing strategy
sharding.jdbc.config.sharding.tables.<logic-table-name>.table-strategy.xxx= #Ignore

sharding.jdbc.config.sharding.tables.<logic-table-name>.key-generator-column-name=
#Column name of key generator, do not use Key generator if absent
sharding.jdbc.config.sharding.tables.<logic-table-name>.key-generator-class-name=
#Key generator, use default key generator if absent. This class need to implements
KeyGenerator, and require a no argument constructor

sharding.jdbc.config.sharding.tables.<logic-table-name>.logic-index= #Name if logic
index. If use `DROP INDEX XXX` SQL in Oracle/PostgreSQL, This property needs to be
set for finding the actual tables

sharding.jdbc.config.sharding.binding-tables[0]= #Binding table rule configurations
sharding.jdbc.config.sharding.binding-tables[1]= #Binding table rule configurations
sharding.jdbc.config.sharding.binding-tables[x]= #Binding table rule configurations

sharding.jdbc.config.sharding.broadcast-tables[0]= #Broadcast table rule
configurations
sharding.jdbc.config.sharding.broadcast-tables[1]= #Broadcast table rule
configurations
sharding.jdbc.config.sharding.broadcast-tables[x]= #Broadcast table rule
configurations

sharding.jdbc.config.sharding.default-data-source-name= #If table not configure at
table rule, will route to defaultDataSourceName
```

```
sharding.jdbc.config.sharding.default-database-strategy.xxx= #Default strategy for
sharding databases, same as databases sharding strategy
sharding.jdbc.config.sharding.default-table-strategy.xxx= #Default strategy for
sharding tables, same as tables sharding strategy
sharding.jdbc.config.sharding.default-key-generator-class-name= #Default key
generator class name, default value is `io.shardingsphere.core.keygen.
DefaultKeyGenerator`. This class need to implements KeyGenerator, and require a no
argument constructor

sharding.jdbc.config.sharding.master-slave-rules.<master-slave-data-source-name>.
master-data-source-name= #more details can reference readwrite-splitting part
sharding.jdbc.config.sharding.master-slave-rules.<master-slave-data-source-name>.
slave-data-source-names[0]= #more details can reference readwrite-splitting part
sharding.jdbc.config.sharding.master-slave-rules.<master-slave-data-source-name>.
slave-data-source-names[1]= #more details can reference readwrite-splitting part
sharding.jdbc.config.sharding.master-slave-rules.<master-slave-data-source-name>.
slave-data-source-names[x]= #more details can reference readwrite-splitting part
sharding.jdbc.config.sharding.master-slave-rules.<master-slave-data-source-name>.
load-balance-algorithm-class-name= #more details can reference readwrite-splitting
part
sharding.jdbc.config.sharding.master-slave-rules.<master-slave-data-source-name>.
load-balance-algorithm-type= #more details can reference readwrite-splitting part
sharding.jdbc.config.config.map.key1= #more details can reference Readwrite-
splitting part
sharding.jdbc.config.config.map.key2= #more details can reference Readwrite-
splitting part
sharding.jdbc.config.config.map.keyx= #more details can reference Readwrite-
splitting part

sharding.jdbc.config.props.sql.show= #To show SQLS or not, default value: false
sharding.jdbc.config.props.executor.size= #The number of working threads, default
value: CPU count

sharding.jdbc.config.config.map.key1= #User-defined arguments
sharding.jdbc.config.config.map.key2= #User-defined arguments
sharding.jdbc.config.config.map.keyx= #User-defined arguments
```

Readwrite-splitting

Configuration Item Explanation

```
#Ignore data sources configuration, same as sharding

sharding.jdbc.config.sharding.master-slave-rules.<master-slave-data-source-name>.
master-data-source-name= #Name of master data source
sharding.jdbc.config.sharding.master-slave-rules.<master-slave-data-source-name>.
slave-data-source-names[0]=  #Name of master data source
```

```
sharding.jdbc.config.sharding.master-slave-rules.<master-slave-data-source-name>.
slave-data-source-names[1]= #Names of Slave data sources
sharding.jdbc.config.sharding.master-slave-rules.<master-slave-data-source-name>.
slave-data-source-names[x]= #Names of Slave data sources
sharding.jdbc.config.sharding.master-slave-rules.<master-slave-data-source-name>.
load-balance-algorithm-class-name= #Load balance algorithm class name. This class
need to implements MasterSlaveLoadBalanceAlgorithm, and require a no argument
constructor
sharding.jdbc.config.sharding.master-slave-rules.<master-slave-data-source-name>.
load-balance-algorithm-type= #Load balance algorithm type, values should be:
`ROUND_ROBIN` or `RANDOM`. Ignore if `load-balance-algorithm-class-name` is present

sharding.jdbc.config.config.map.key1= #User-defined arguments
sharding.jdbc.config.config.map.key2= #User-defined arguments
sharding.jdbc.config.config.map.keyx= #User-defined arguments

sharding.jdbc.config.props.sql.show= #To show SQLS or not, default value: false
sharding.jdbc.config.props.executor.size= #The number of working threads, default
value: CPU count
sharding.jdbc.config.props.check.table.metadata.enabled= #Check the metadata
consistency of all the tables, default value: false
```

Orchestration

Configuration Item Explanation

```
#Ignore data sources, sharding and readwrite splitting configuration

sharding.jdbc.config.sharding.orchestration.name= #Name of orchestration instance
sharding.jdbc.config.sharding.orchestration.overwrite= #Use local configuration to
overwrite registry center or not
sharding.jdbc.config.sharding.orchestration.registry.server-lists= #Rgistry servers
list, multiple split as comma. Example: host1:2181,host2:2181
sharding.jdbc.config.sharding.orchestration.registry.namespace= #Namespace of
registry
sharding.jdbc.config.sharding.orchestration.registry.digest= #Digest for registry.
Default is not need digest.
sharding.jdbc.config.sharding.orchestration.registry.operation-timeout-
milliseconds= #Operation timeout time in milliseconds, default value is 500
milliseconds
sharding.jdbc.config.sharding.orchestration.registry.max-retries= #Max number of
times to retry, default value is 3
sharding.jdbc.config.sharding.orchestration.registry.retry-interval-milliseconds=
#Time interval in milliseconds on each retry, default value is 500 milliseconds
sharding.jdbc.config.sharding.orchestration.registry.time-to-live-seconds= #Time to
live in seconds of ephemeral keys, default value is 60 seconds
```

Shardingsphere-2.x

Sharding

Configuration Item Explanation

```
# Ignore data sources configuration
sharding.jdbc.config.sharding.default-data-source-name= #Tables without sharding
rules will be located through default data source
sharding.jdbc.config.sharding.default-database-strategy.inline.sharding-column=
#Name of database sharding column
sharding.jdbc.config.sharding.default-database-strategy.inline.algorithm-
expression= #Inline expression for database sharding algorithm
sharding.jdbc.config.sharding.tables.t_order.actualDataNodes= #Describe data source
names and actual tables, delimiter as point, multiple data nodes separated with
comma, support inline expression. Absent means sharding databases only. Example: ds
${0..7}.tbl${0..7}
sharding.jdbc.config.sharding.tables.t_order.tableStrategy.inline.shardingColumn=
#Name of table sharding column
sharding.jdbc.config.sharding.tables.t_order.tableStrategy.inline.
algorithmInlineExpression= #Inline expression for table sharding algorithm
sharding.jdbc.config.sharding.tables.t_order.keyGeneratorColumnName= #Column name
of key generator, do not use Key generator if absent


sharding.jdbc.config.sharding.tables.<logic-table-name>.key-generator-column-name=
#Column name of key generator, do not use Key generator if absent
sharding.jdbc.config.sharding.tables.<logic-table-name>.key-generator-class-name=
#Key generator, use default key generator if absent. This class need to implements
KeyGenerator, and require a no argument constructor
```

Readwrite-splitting

Configuration Item Explanation

```
# Ignore data sources configuration

sharding.jdbc.config.masterslave.load-balance-algorithm-type= #Load balance
algorithm class of slave database; optional value: ROUND_ROBIN and RANDOM; if there
is load-balance-algorithm-class-name, the configuration can be omitted
sharding.jdbc.config.masterslave.name= # master name
sharding.jdbc.config.masterslave.master-data-source-name= #Name of master data
source
sharding.jdbc.config.masterslave.slave-data-source-names= #Name of master data
source
```

Orchestration

Configuration Item Explanation

```
# Ignore data sources configuration
sharding.jdbc.config.orchestration.name= #Name of orchestration instance
sharding.jdbc.config.orchestration.overwrite= #Use local configuration to overwrite
registry center or not


sharding.jdbc.config.sharding.orchestration.name= #Name of orchestration instance
sharding.jdbc.config.sharding.orchestration.overwrite= #Use local configuration to
overwrite registry center or not
sharding.jdbc.config.sharding.orchestration.registry.server-lists= #Rgistry servers
list, multiple split as comma. Example: host1:2181,host2:2181
sharding.jdbc.config.sharding.orchestration.registry.namespace= #Namespace of
registry
sharding.jdbc.config.sharding.orchestration.registry.digest= #Digest for registry.
Default is not need digest.
sharding.jdbc.config.sharding.orchestration.registry.operation-timeout-
milliseconds= #Operation timeout time in milliseconds, default value is 500
milliseconds
sharding.jdbc.config.sharding.orchestration.registry.max-retries= #Max number of
times to retry, default value is 3
sharding.jdbc.config.sharding.orchestration.registry.retry-interval-milliseconds=
#Time interval in milliseconds on each retry, default value is 500 milliseconds
sharding.jdbc.config.sharding.orchestration.registry.time-to-live-seconds= #Time to
live in seconds of ephemeral keys, default value is 60 seconds

# The configuration in Zookeeper
sharding.jdbc.config.orchestration.zookeeper.namespace= #Namespace of zookeeper
registry
sharding.jdbc.config.orchestration.zookeeper.server-lists= #Zookeeper Rgistry
servers list, multiple split as comma. Example: host1:2181,host2:2181

# The configuration in Etcd
sharding.jdbc.config.orchestration.etcd.server-lists= #Etcd Rgistry servers list,
multiple split as comma. Example: host1:2181,host2:2181
```

9.9.2 ShardingSphere-Proxy

5.0.0-alpha

Data Source Configuration Item Explanation

```
schemaName: # Logic schema name.

dataSourceCommon:
  username: # Database username.
  password: # Database password.
  connectionTimeoutMilliseconds: # Connection timeout milliseconds.
  idleTimeoutMilliseconds: # Idle timeout milliseconds.
  maxLifetimeMilliseconds: # Maximum life milliseconds.
  maxPoolSize: 50 # Maximum connection count in the pool.
  minPoolSize: 1  # Minimum connection count in the pool.

dataSources: # Data sources configuration, multiple <data-source-name> available.
  <data-source-name>: # Different from ShardingSphere-JDBC configuration, it does
not need to be configured with database connection pool.
    url: # Database URL.
rules: # Keep consist with ShardingSphere-JDBC configuration.
# ...
```

Override dataSourceCommon Configuration

If you want to override the 'dataSourceCommon' property, configure it separately for each data source.

```
dataSources: # Data sources configuration, multiple <data-source-name> available.
  <data-source-name>: # Different from ShardingSphere-JDBC configuration, it does
not need to be configured with database connection pool.
    url: # Database URL.
    username: # Database username, Override dataSourceCommon username property.
    password: # Database password, Override dataSourceCommon password property.
    connectionTimeoutMilliseconds: # Connection timeout milliseconds, Override
dataSourceCommon connectionTimeoutMilliseconds property.
    idleTimeoutMilliseconds: # Idle timeout milliseconds, Override dataSourceCommon
idleTimeoutMilliseconds property.
    maxLifetimeMilliseconds: # Maximum life milliseconds, Override dataSourceCommon
maxLifetimeMilliseconds property.
    maxPoolSize: 50 # Maximum connection count in the pool, Override
dataSourceCommon maxPoolSize property.
    minPoolSize: 1  # Minimum connection count in the pool, Override
dataSourceCommon minPoolSize property.
```

Authentication

It is used to verify the authentication to log in ShardingSphere-Proxy, which must use correct user name and password after the configuration of them.

```
authentication:
  users:
    root: # Self-defined username.
      password: root # Self-defined password.
    sharding: # Self-defined username.
      password: sharding # Self-defined password.
      authorizedSchemas: sharding_db, replica_query_db # Schemas authorized to this user, please use commas to connect multiple schemas. Default authorized schemas is all of the schemas.
```

Proxy Properties

```
props:
  sql-show: # 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.
  sql-simple: # Whether show SQL details in simple style.
  acceptor-size: # The max thread size of accepter group to accept TCP connections.
  executor-size: # 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.
  max-connections-size-per-query: # Max opened connection size for each query.
  check-table-metadata-enabled: # Whether validate table meta data consistency when application startup or updated.
  query-with-cipher-column: # Whether query with cipher column for data encrypt. User you can use plaintext to query if have.
  proxy-frontend-flush-threshold: # Flush threshold for every records from databases for ShardingSphere-Proxy.
  proxy-transaction-type: # Default transaction type of ShardingSphere-Proxy. Include: LOCAL, XA and BASE.
  proxy-opentracing-enabled: # Whether enable opentracing for ShardingSphere-Proxy.
  proxy-hint-enabled: # Whether enable hint for ShardingSphere-Proxy. Using Hint will switch proxy thread mode from IO multiplexing to per connection per thread, which will reduce system throughput.
```

ShardingSphere-4.x

Data Source and Sharding Configuration Item Explanation

Data Sharding

```
schemaName: # Logic data schema name.

dataSources: # Data source configuration, which can be multiple data_source_name.
  <data_source_name>: # Different from Sharding-JDBC configuration, it does not need to be configured with database connection pool.
    url: # Database url connection.
    username: # Database username.
    password: # Database password.
    connectionTimeoutMilliseconds: 30000 # Connection timeout.
    idleTimeoutMilliseconds: 60000 # Idle timeout setting.
    maxLifetimeMilliseconds: 1800000 # Maximum lifetime.
    maxPoolSize: 65 # Maximum connection number in the pool.

shardingRule: #Omit data sharding configuration and be consistent with Sharding-JDBC configuration.
```

Read-write splitting

```
schemaName: # Logic data schema name.

dataSources: # Omit data source configurations; keep it consistent with data sharding.

masterSlaveRule: # Omit data source configurations; keep it consistent with Sharding-JDBC.
```

Data Masking

```
dataSource: # Ignore data sources configuration.

encryptRule:
  encryptors:
    <encryptor-name>:
      type: # encryptor type.
      props: # Properties, e.g. `aes.key.value` for AES encryptor.
        aes.key.value:
  tables:
    <table-name>:
      columns:
```

```
        <logic-column-name>:
          plainColumn: # plaintext column name.
          cipherColumn: # ciphertext column name.
          assistedQueryColumn: # AssistedColumns for query, when use
ShardingQueryAssistedEncryptor, it can help query encrypted data.
          encryptor: # encrypt name.
props:
  query.with.cipher.column: true #Whether use cipherColumn to query or not
```

Overall Configuration Explanation

Orchestration

It is the same with Sharding-JDBC configuration.

Proxy Properties

```
# Omit configurations that are the same with Sharding-JDBC.

props:
  acceptor.size: # The thread number of accept connection; default to be 2 times of
cpu core.
  proxy.transaction.type: # Support LOCAL, XA, BASE; Default is LOCAL transaction,
for BASE type you should copy ShardingTransactionManager associated jar to lib
directory.
  proxy.opentracing.enabled: # Whether to enable opentracing, default not to
enable; refer to [APM](https://shardingsphere.apache.org/document/current/en/
features/orchestration/apm/) for more details.
  check.table.metadata.enabled: # Whether to check metadata consistency of sharding
table when it initializes; default value: false.
```

Authentication

It is used to verify the authentication to log in Sharding-Proxy, which must use correct user name and password after the configuration of them.

```
authentication:
  users:
    root: # self-defined username.
      password: root # self-defined password.
    sharding: # self-defined username.
      password: sharding # self-defined password.
      authorizedSchemas: sharding_db, masterslave_db # schemas authorized to this
user, please use commas to connect multiple schemas. Default authorizedSchemas is
all of the schemas.
```

ShardingSphere-3.x

Data sources and sharding rule configuration reference

Data Sharding

```
schemaName: # Logic database schema name.

dataSources: # Data sources configuration, multiple `data_source_name` available.
  <data_source_name>: # Different with Sharding-JDBC, do not need configure data
source pool here.
    url: # Database URL.
    username: # Database username.
    password: # Database password.
    autoCommit: true # The default config of hikari connection pool.
    connectionTimeout: 30000 # The default config of hikari connection pool.
    idleTimeout: 60000 # The default config of hikari connection pool.
    maxLifetime: 1800000 # The default config of hikari connection pool.
    maximumPoolSize: 65 # The default config of hikari connection pool.

shardingRule: # Ignore sharding rule configuration, same as Sharding-JDBC.
```

Read-write splitting

```
schemaName: # Logic database schema name.

dataSources: # Ignore data source configuration, same as sharding.

masterSlaveRule: # Ignore read-write splitting rule configuration, same as
Sharding-JDBC.
```

Global configuration reference

Orchestration

Same as configuration of Sharding-JDBC.

Proxy Properties

```
# Ignore configuration which same as Sharding-JDBC.

props:
  acceptor.size: # Max thread count to handle client's requests, default value is
CPU*2.
  proxy.transaction.enabled: # Enable transaction, only support XA now, default
value is false.
  proxy.opentracing.enabled: # Enable open tracing, default value is false. More
details please reference[APM](https://shardingsphere.apache.org/document/current/
en/features/orchestration/apm/).
  check.table.metadata.enabled: # To check the metadata consistency of all the
tables or not, default value : false.
```

Authorization

To perform Authorization for Sharding Proxy when login in. After configuring the username and password, you must use the correct username and password to login into the Proxy.

```
authentication:
  username: root
  password:
```

10 Downloads

10.1 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.

10.1.1 Apache ShardingSphere - Version: 5.0.0 (Release Date: Nov 10th, 2021)

- Source Codes: [SRC] [ASC] [SHA512]
- ShardingSphere-JDBC Binary Distribution: [TAR] [ASC] [SHA512]
- ShardingSphere-Proxy Binary Distribution: [TAR] [ASC] [SHA512]

10.2 All Releases

Find all releases in the Archive repository. Find all incubator releases in the Archive incubator repository.

10.3 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
```