# Apache ShardingSphere document

***v5.0.0-beta***

**Apache ShardingSphere**

**Jun 19, 2021**

# Contents

# 1 Overview

**Stargazers over time**

**Contributor over time**

Apache ShardingSphere is an open-source ecosystem consisted of a set of distributed database solutions, 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.

Apache ShardingSphere aiming at reasonably making full use of the computation and storage capacity of existed database in distributed system, rather than a totally new database. As the cornerstone of enterprises, relational database still takes a huge market share. Therefore, we prefer to focus on its increment instead of a total overturn.

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, replica query, 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.

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

Welcome communicate with community via mail list.

## 1.1 Introduction

### 1.1.1 ShardingSphere-JDBC

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, Druid, HikariCP.
- Support any kind of JDBC standard database: MySQL, Oracle, SQLServer, PostgreSQL and any SQL92 followed databases.

### 1.1.2 ShardingSphere-Proxy

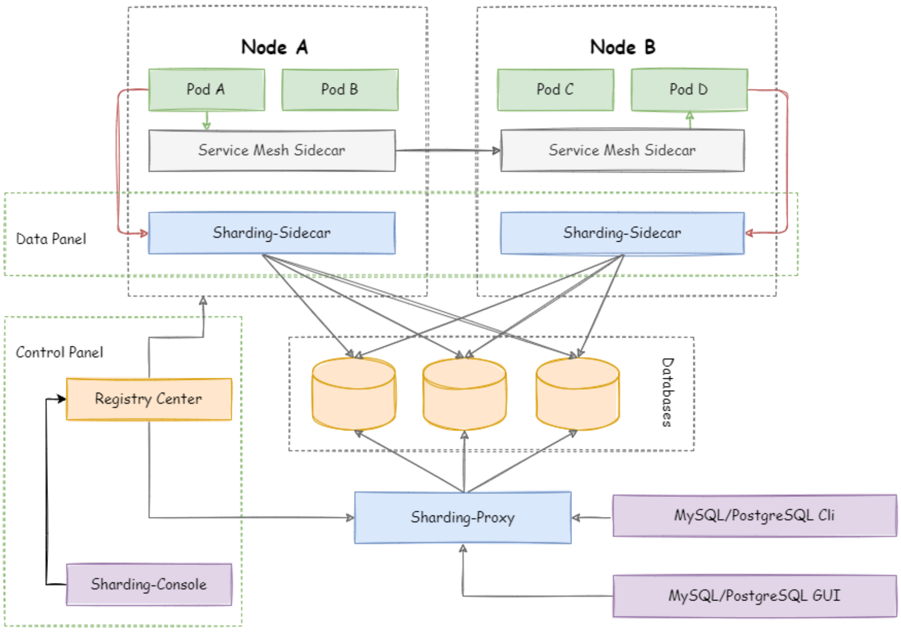
ShardingSphere-Proxy defines itself as a transparent database proxy, providing a database server that encapsulates database binary protocol to support heterogeneous languages. Friendlier to DBA, the MySQL version provided now can use any kind of terminal (such as MySQL Command Client, MySQL Workbench, etc.) that is compatible of MySQL protocol to operate data.

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

### 1.1.3 ShardingSphere-Sidecar(TODO)

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`.

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




| | *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 |

### 1.1.4 Hybrid Architecture

ShardingSphere-JDBC adopts 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 ecosphere consists of multiple endpoints together. Through a mixed use of ShardingSphere-JDBC and ShardingSphere-Proxy and unified sharding strategy by the same registry center, ShardingSphere can build an application system applicable to all kinds of scenarios. Architects can adjust the system architecture to the most applicable one to current business more freely.

## 1.2 Features

### 1.2.1 Data Sharding

- Database sharding & Table sharding
- Replica query
- Sharding strategy customization
- Centre-less Distributed primary key

### 1.2.2 Distributed Transaction

- Unified Transaction API
- XA transaction
- BASE transaction

### 1.2.3 Database Governance

- Distributed Governance
- Data migration & Scale out
- Tracing & Observability Supported
- Data Encryption

# 2 Quick Start

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

## 2.1 ShardingSphere-JDBC

### 2.1.1 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.

### 2.1.2 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 Configuration Manual for more details.

### 2.1.3 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, MyBatis and other ORM frameworks.

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

## 2.2 ShardingSphere-Proxy

### 2.2.1 1. Rule Configuration

Edit `%SHARDINGSPHERE_PROXY_HOME%/conf/config-xxx.yaml`. Please refer to Configuration Manual for more details.

Edit `%SHARDINGSPHERE_PROXY_HOME%/conf/server.yaml`. Please refer to Configuration Manual for more details.

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

### 2.2.2 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 and put it into `%SHARDINGSPHERE_PROXY_HOME%/lib` directory.

### 2.2.3 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}
```

### 2.2.4 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}
```

## 2.3 ShardingSphere-Scaling(Alpha)

### 2.3.1 1. Rule Configuration

Edit `%SHARDINGSPHERE_SCALING_HOME%/conf/server.yaml`. Please refer to Configuration Manual for more details.

> %SHARDINGSPHERE_PROXY_HOME% is the shardingsphere scaling extract path. for example: /Users/ss/shardingsphere-scaling-bin/

### 2.3.2 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 and put it into `%SHARDINGSPHERE_SCALING_HOME%/lib` directory.

### 2.3.3 3. Start Server

```
sh %SHARDINGSPHERE_SCALING_HOME%/bin/start.sh
```

### 2.3.4 4. Create Migration Job

Use HTTP interface to manage the migration jobs.

Please refer to Configuration Manual for more details.

# 3 Concepts & Features

This chapter describes concepts and features about Apache ShardingSphere. Please refer to User manual for more details.

## 3.1 Sharding

### 3.1.1 Background

The traditional solution that stores all the data in one concentrated node has hardly satisfied the requirement of massive Internet 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 replica query 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.

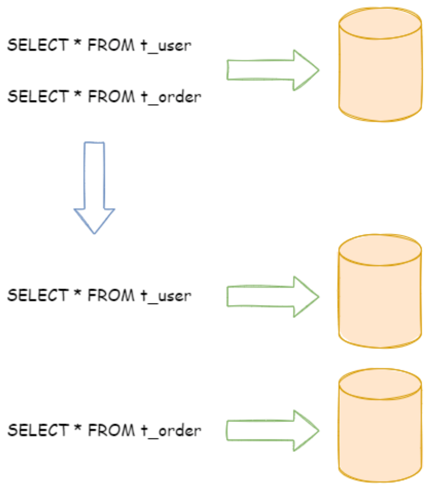
Under the circumstance that traditional relational databases cannot satisfy the requirement of the Internet, there are more and more attempts to store the data in native distributed NoSQL. But its incompatibility with SQL and imperfection in ecosystem block it from defeating the relational database in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

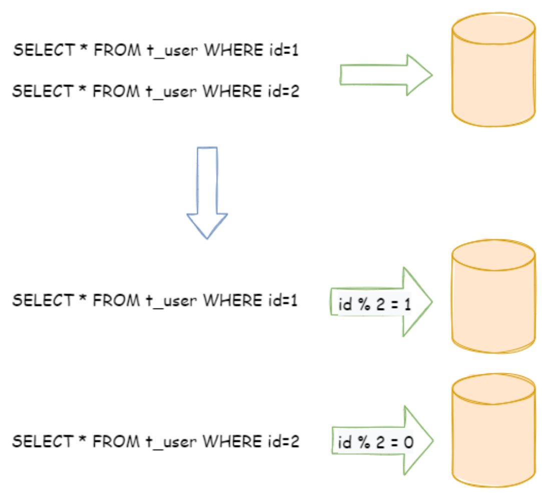
According to business sharding method, it is called vertical sharding, or longitudinal sharding, the core concept of which is to specialize databases for different uses. Before sharding, a database consists of many tables corresponding to different businesses. But after sharding, tables are categorized 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.

### 3.1.2 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.

### 3.1.3 Goal

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

### 3.1.4 Core Concept

#### Overview

This chapter is to introduce core concepts of data sharding, including:

- Core concepts of SQL
- Core concepts of sharding
- Core concepts of configuration
- Inline expression
- Distributed primary key
- Hint Sharding route

#### SQL

#### Logic Table

It refers collectively to horizontal sharding databases (tables) with the same logic and data structure. For instance, the order data 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 sharding database, i.e., `t_order_0` to `t_order_9` in the instance above.

#### Data Node

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

#### 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-table correlating query, so the query efficiency will increase greatly. Take this one for example, if SQL is:

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

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:

```sql
SELECT i.* FROM t_order_0 o JOIN t_order_item_0 i ON o.order_id=i.order_id WHERE o.
order_id in (10, 11);
SELECT i.* FROM t_order_0 o JOIN t_order_item_1 i ON o.order_id=i.order_id WHERE o.
order_id in (10, 11);
SELECT i.* FROM t_order_1 o JOIN t_order_item_0 i ON o.order_id=i.order_id WHERE o.
order_id in (10, 11);
SELECT i.* FROM t_order_1 o JOIN t_order_item_1 i ON o.order_id=i.order_id WHERE o.
order_id in (10, 11);
```

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

```sql
SELECT i.* FROM t_order_0 o JOIN t_order_item_0 i ON o.order_id=i.order_id WHERE o.
order_id in (10, 11);
SELECT i.* FROM t_order_1 o JOIN t_order_item_1 i ON o.order_id=i.order_id WHERE o.
order_id in (10, 11);
```

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. Their structures and data are the same in each database. It can be applied to the small data volume scenario that needs to correlate with big data volume 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 scenarios where the amount of data is not large and does not require any sharding operations.

#### Sharding

#### Sharding Key

The database field used in sharding refers to the key field in horizontal sharding of the database (table). 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 field 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`. They need to be implemented by developers themselves and can be highly flexible.

Currently, 3 kinds of sharding algorithms are available. Since the sharding algorithm and business achievement are closely related, it extracts all kinds of scenarios by sharding strategies, instead of providing built-in sharding algorithms. Therefore, it can provide higher abstraction and the interface for developers to implement sharding algorithm by themselves.

- Standard Sharding Algorithm

`StandardShardingAlgorithm` is to process the sharding case in which single sharding keys `=`, `IN`, `BETWEEN AND`, `>`, `<`, `>=`, `<=` are used; `StandardShardingStrategy` needs to be used together.

- Complex Keys Sharding Algorithm

`ComplexKeysShardingAlgorithm` is to process the sharding case in which multiple sharding keys are used; `ComplexShardingStrategy` needs to be used together. It has a relatively complex logic that requires developers to deal by themselves.

- Hint Sharding Algorithm

`HintShardingAlgorithm` is to process the sharding case in which Hint is used; `HintShardingStrategy`needs to be used together.

#### 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, i.e., the sharding strategy, can be used in sharding operation. For now, 4 kinds of sharding strategies are available.

- Standard Sharding Strategy

`StandardShardingStrategy` provides support for the sharding operation of =, >, <, >=, <=, `IN` and `BETWEEN AND` in SQL. `StandardShardingStrategy` only supports single sharding keys and provides two sharding algorithms of `PreciseShardingAlgorithm` and `RangeShardingAlgorithm`. `PreciseShardingAlgorithm` is compulsory and used to operate the sharding of = and `IN`. `RangeShardingAlgorithm` is optional and used to operate the sharding of `BETWEEN AND`, >, <, >=, <=. `BETWEEN AND` in SQL will operate by way of all data node route without the configuration of `RangeShardingAlgorithm`.

- Complex Sharding Strategy

`ComplexShardingStrategy` provides support for the sharding operation of =, >, <, >=, <=, `IN` and `BETWEEN AND` in SQL. `ComplexShardingStrategy` supports multiple sharding keys, but since their relationships are so complex that there is not too much encapsulation, the combination of sharding keys and sharding operators are in the algorithm interface and achieved by developers with the most flexibility.

- Hint Sharding Strategy

`HintShardingStrategy` refers to the sharding strategy which get sharding values by hint rather than extracted from SQL.

- None sharding strategy

`NoneShardingStrategy` refers to the strategy with no sharding.

#### SQL Hint

In the case that the `ShardingColumn` is not decided by SQL but other external conditions, SQL hint can be used flexibly to inject `ShardingColumn`. For example, in the internal system, databases are divided according to the staff's ID, but this column does not exist in the database. SQL Hint can be used by two ways, Java API and SQL comment (to do). Please refer to Hint for more details.

#### Configuration

#### Sharding Rule

The main entrance for Sharding rules includes the configurations of data source, tables, binding tables and replica query.

#### Data Sources Configuration

Real data sources list.

#### Tables Configuration

Configurations of logic table names, data node and table sharding rules.

#### Data Node Configuration

It is used in the configurations of the mapping relationship between logic tables and actual tables and can be divided into two kinds: uniform distribution and user-defined distribution.

- Uniform distribution

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

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

So the data node configurations will be as follows:

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

- User-defined distribution

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

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

So the data node configurations will be as follows:

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

### Sharding Strategy Configuration

There are two dimensions of sharding strategies, database sharding and table sharding.

- Database sharding strategy

`DatabaseShardingStrategy` is used to configure data in the targeted database.

- Table sharding strategy

`TableShardingStrategy` is used to configure data in the targeted table that exists in the database. So the table sharding strategy relies on the result of the database sharding strategy.

API of those two kinds of strategies are totally same.

### Auto-increment Key Generation Strategy

Replacing the original database auto-increment key with that generated in the server can make distributed key not repeat.

### 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 stored 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
```

#### Data Node Configuration

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 Configuration

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.

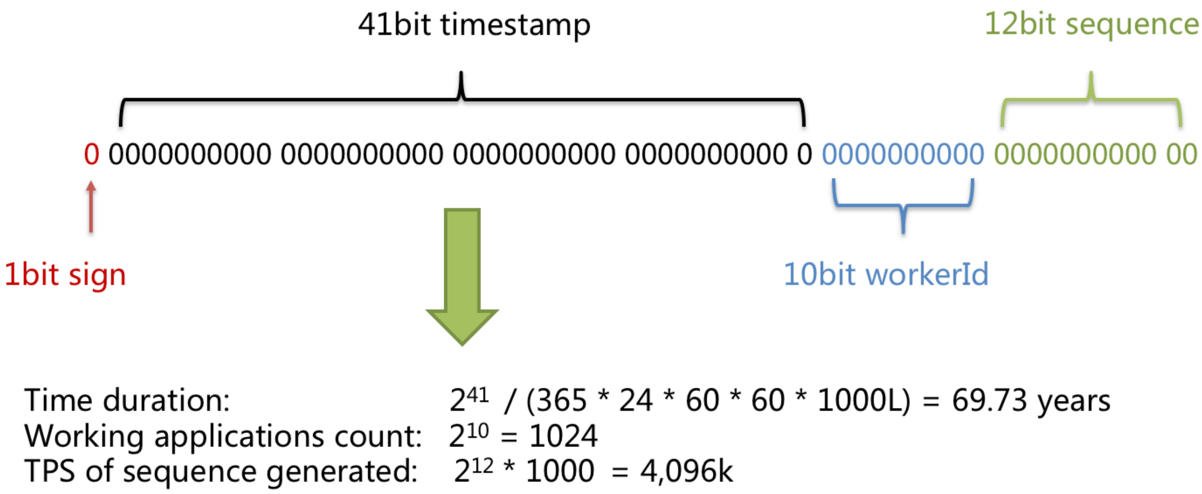
- work ID bit (10bit)

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.




#### 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 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

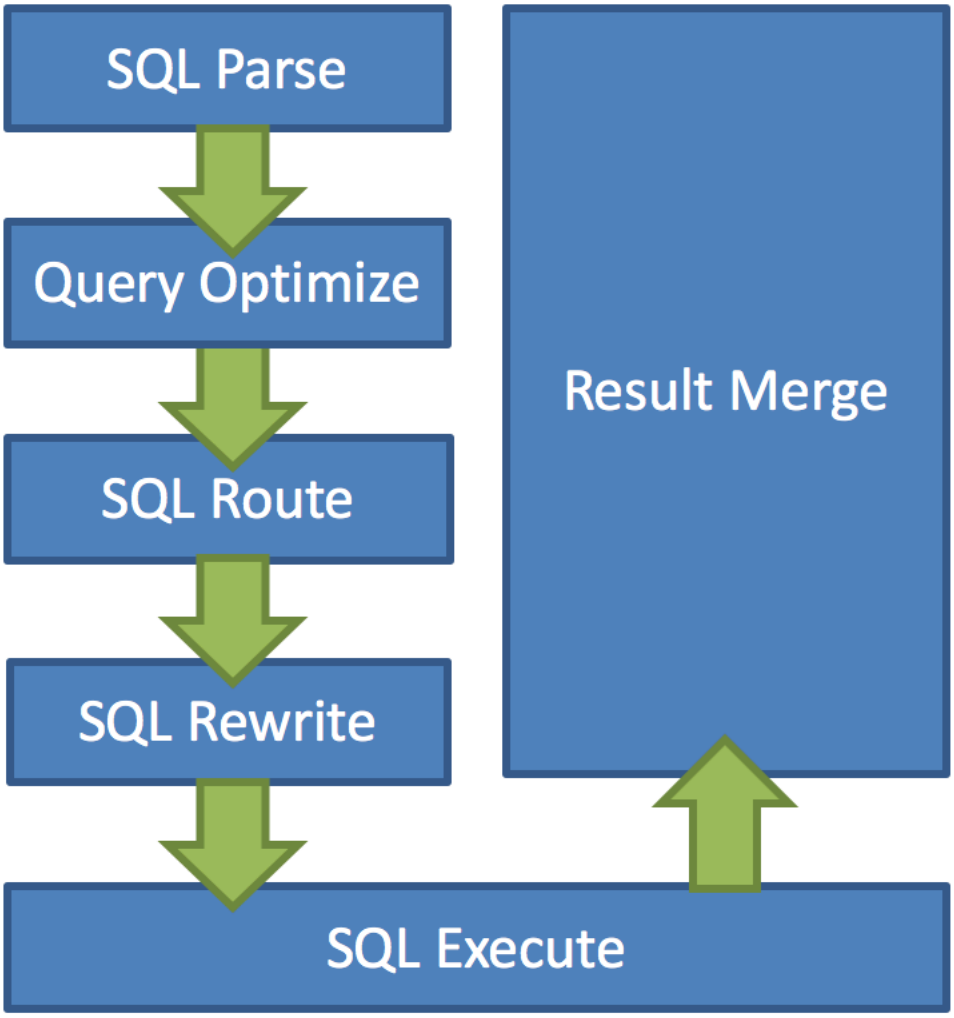
Apache ShardingSphere uses `ThreadLocal` to manage sharding key values. Users can program to add sharding conditions to `HintManager`, but the condition is only effective within the current thread.

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.

### 3.1.5 Guide to Kernel

The major sharding processes of all the three ShardingSphere products are identical. The core consists of `SQL parsing => query optimization => SQL route => SQL rewrite => SQL execution => result merger`.



#### 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.

#### Query Optimization

It merges and optimizes sharding conditions, such as OR.

#### 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.

#### 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.

#### SQL Execution

Through multi-thread executor, it executes asynchronously.

#### 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.

#### 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.).

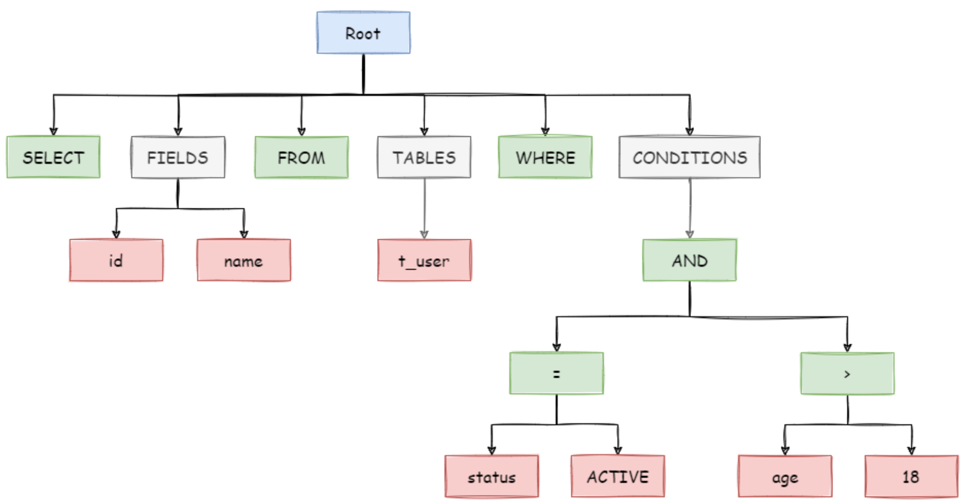
#### Abstract Syntax Tree

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:

```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.

### 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
 * sql type:String SQL to be parsed
 * useCache type:boolean whether use cache
 * @return parse tree
 */
ParseTree tree = new SQLParserEngine(databaseType).parse(sql, useCache)
```

- GET SQLStatement

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

- SQL Format

```
/**
 * databaseType type:String values MySQL
 * useCache type:boolean whether use cache
 * @return String
 */
ParseTree tree = new SQLParserEngine(databaseType).parse(sql, useCache);
```

```
SQLVisitorEngine sqlVisitorEngine = new SQLVisitorEngine(databaseType, "FORMAT");
String formatedSql = sqlVisitorEngine.visit(tree);
```

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; |

#### 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 databases`t_order_0` and `t_order_1`, the SQL will be executed on `t_order_1` after routing. The following is a sample code using the API.

```
String sql = "SELECT * FROM t_order";
try (
        HintManager hintManager = HintManager.getInstance();
        Connection conn = dataSource.getConnection();
        PreparedStatement pstmt = conn.prepareStatement(sql)) {
    hintManager.setDatabaseShardingValue(3);
    try (ResultSet rs = pstmt.executeQuery()) {
        while (rs.next()) {
            //...
        }
```

```
    }
}
```

### Standard Route

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:

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

The route result will be:

```sql
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:

```sql
SELECT * FROM t_order o JOIN t_order_item i ON o.order_id=i.order_id  WHERE order_
id IN (1, 2);
```

Then, the route result will be:

```sql
SELECT * FROM t_order_0 o JOIN t_order_item_0 i ON o.order_id=i.order_id  WHERE
order_id IN (1, 2);
SELECT * FROM t_order_1 o JOIN t_order_item_1 i ON o.order_id=i.order_id  WHERE
order_id IN (1, 2);
```

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:

```sql
SELECT * FROM t_order_0 o JOIN t_order_item_0 i ON o.order_id=i.order_id  WHERE
order_id IN (1, 2);
SELECT * FROM t_order_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.

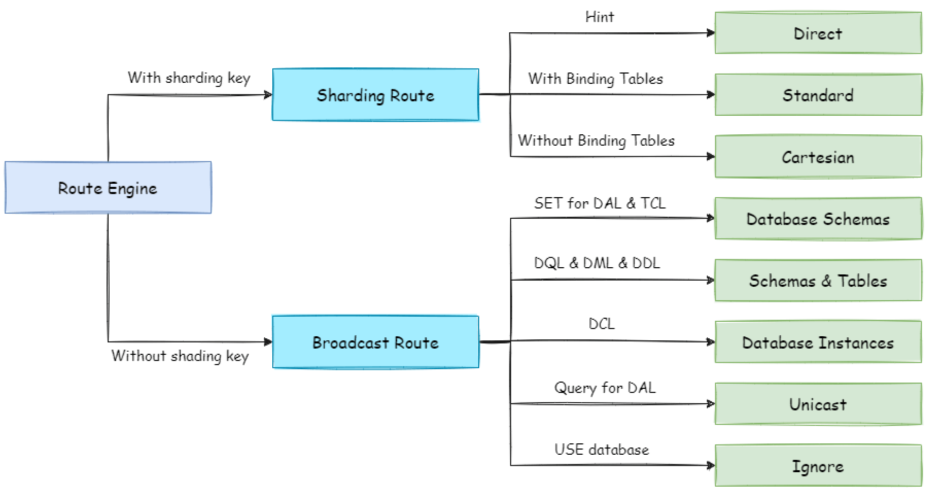
#### Ignore Route

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:




### 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.

### 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:

```sql
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:

```sql
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:

```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:

```sql
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:

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

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:

```sql
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);
```

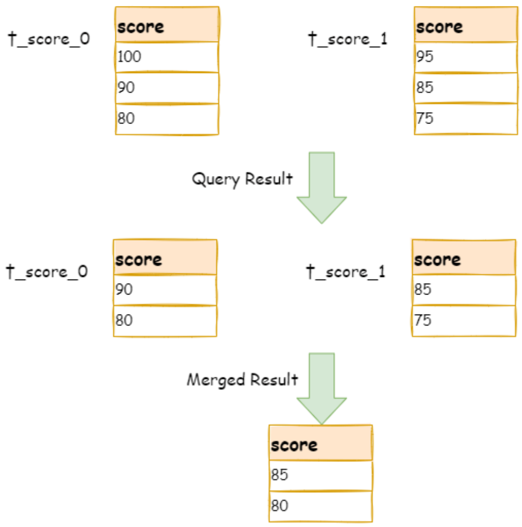
When auto-increment key derives column, if the user writes SQL with placeholder, he only needs to rewrite parameter list but not SQL itself.

### 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:

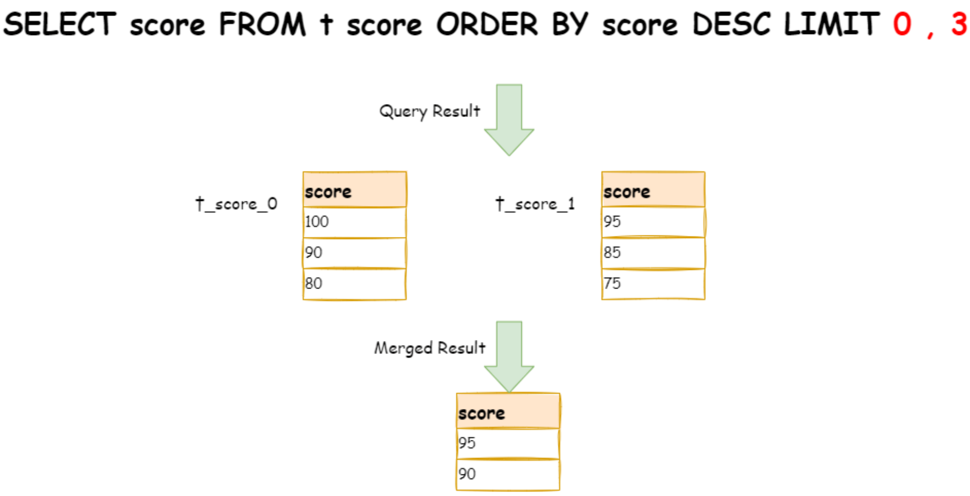
```
SELECT score FROM t_score ORDER BY score DESC LIMIT 1, 2;
```

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 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.




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:

```
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:

```sql
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:

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

Is rewritten as:

```sql
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:

```sql
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.

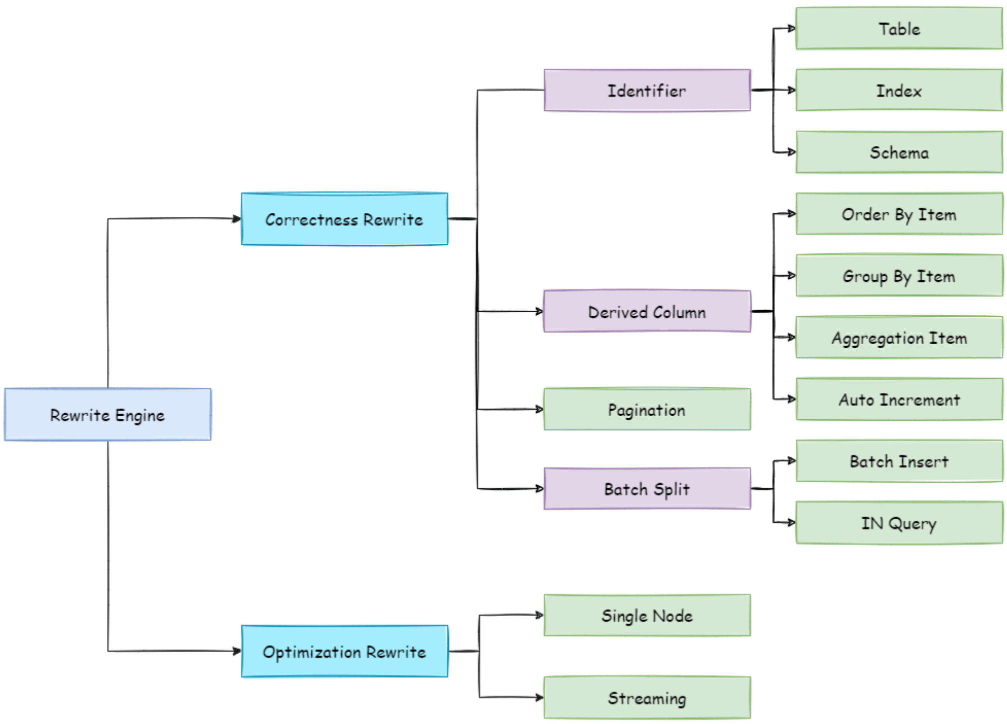
#### Single Node Optimization

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.




### 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`.

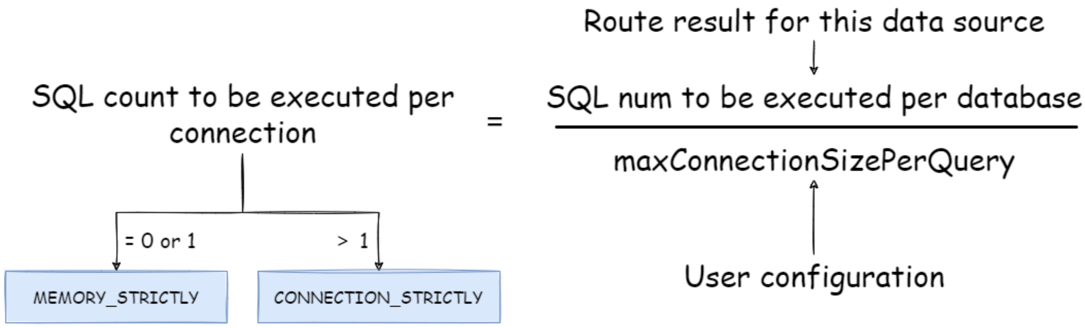
#### Preparation Phrase

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.

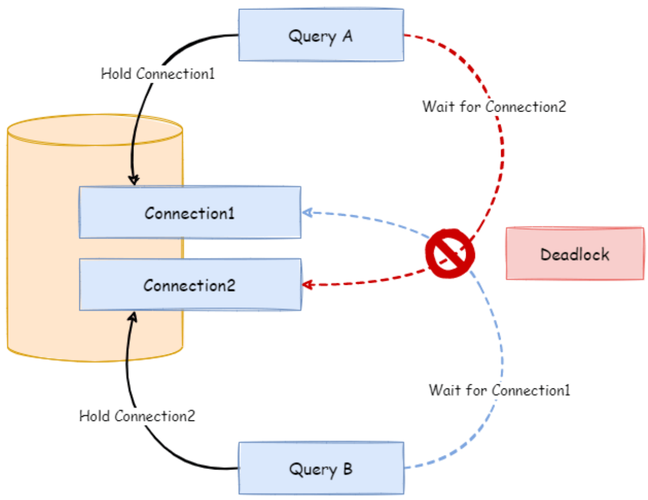



Within the range that maxConnectionSizePerQuery permits, when the request number that one connection needs to execute is more than 1, meaning current database connection cannot hold the corresponding data result set, it must uses memory merger. On the contrary, when it equals to 1, meaning current database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




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:

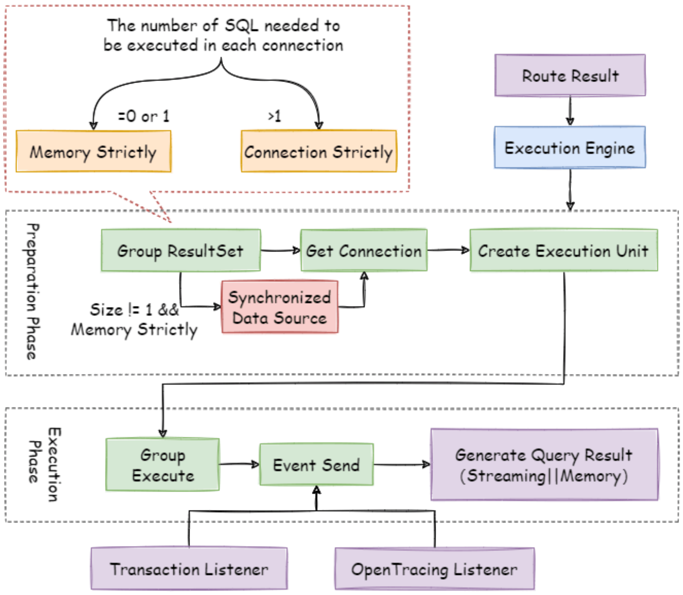
1. Avoid the setting that locking only takes one database connection each time. Because under this kind of circumstance, two requests waiting for each other will not happen, so there is no need for locking. Most OLTP operations use sharding keys to route to the only data node, which will make the system in a totally unlocked state, thereby improve the concurrency efficiency further. In addition to routing to a single shard, replica query 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:




#### 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

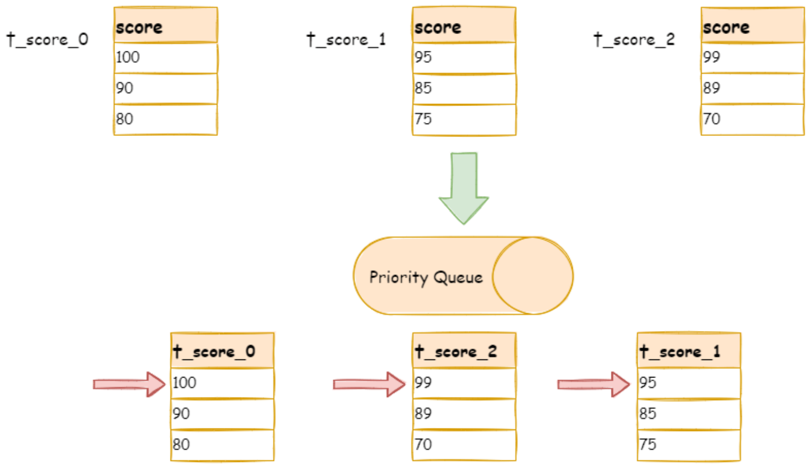
As the simplest merger method, iteration merger only requires the combination of multiple data result sets into a single-direction chain table. After iterating current data result sets in the chain table, it only needs to move the element of chain table to the next position and iterate the next data 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.

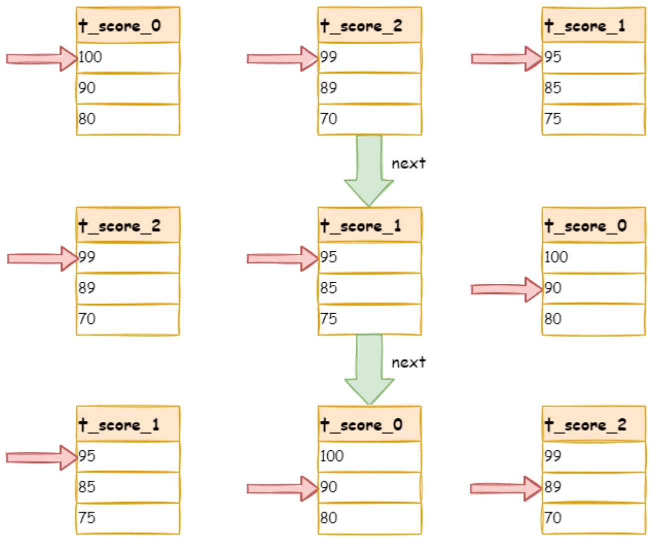
Here is an instance to explain ShardingSphere' s order-by merger. The following picture is an illustration of ordering by the score. Data result sets returned by 3 tables are shown in the example and each one of them has already been ordered according to the score, but there is no order between 3 data 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.




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 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.

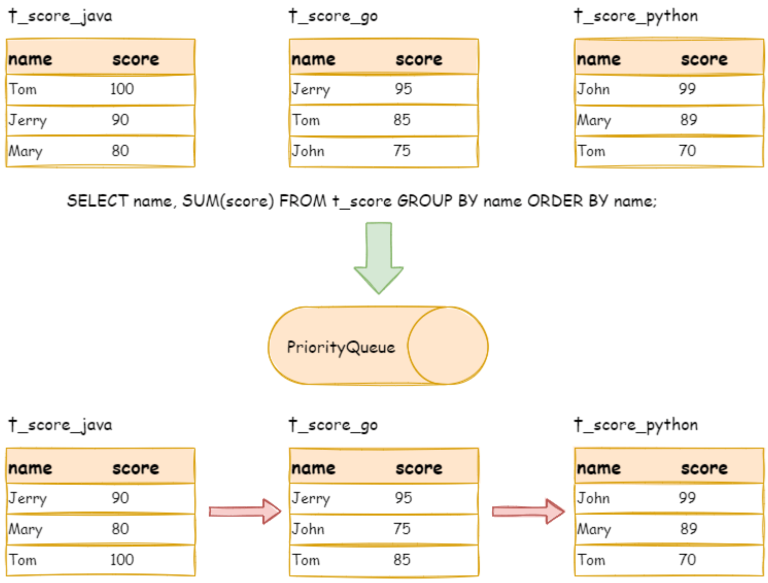
#### Group-by Merger

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.

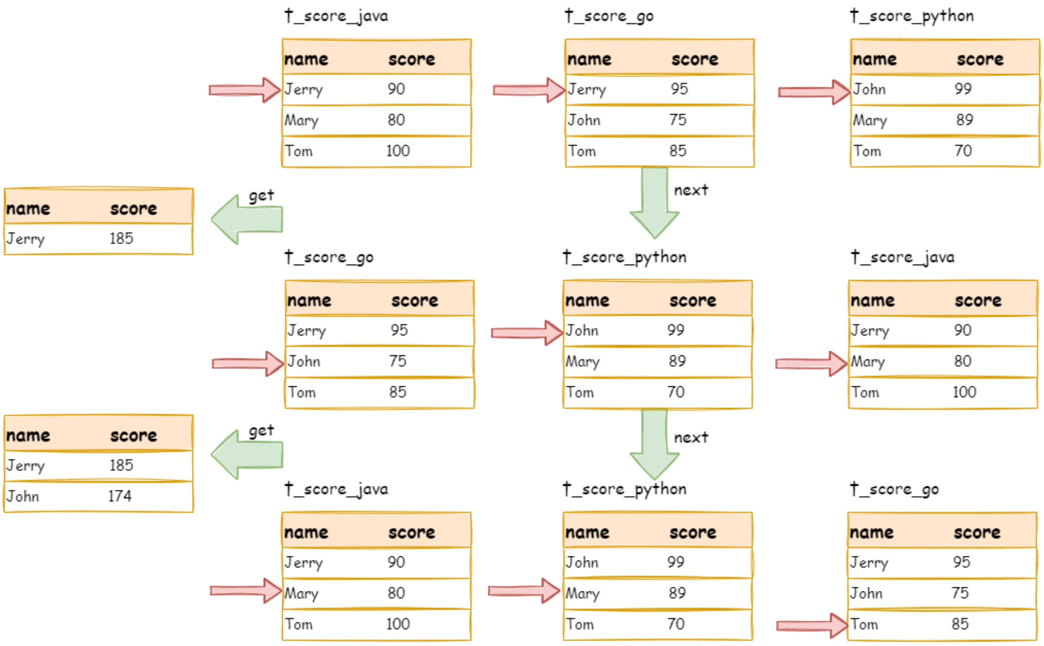
For instance, if it is sharded by subject, table structure contains examinees' name (to simplify, name repetition is not taken into consideration) and score. The SQL used to acquire each examinee's total score is as follow:

```sql
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:




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 "Jetty", will be popped out of the queue. After acquiring all the students' scores with the name of "Jetty", the accumulation operation will be proceeded. Hence, after the first next invocation is finished, the result set acquired is the sum of Jetty's scores. In the same time, all the cursors in data result sets will be moved down to a different data value next to "Jetty" 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.

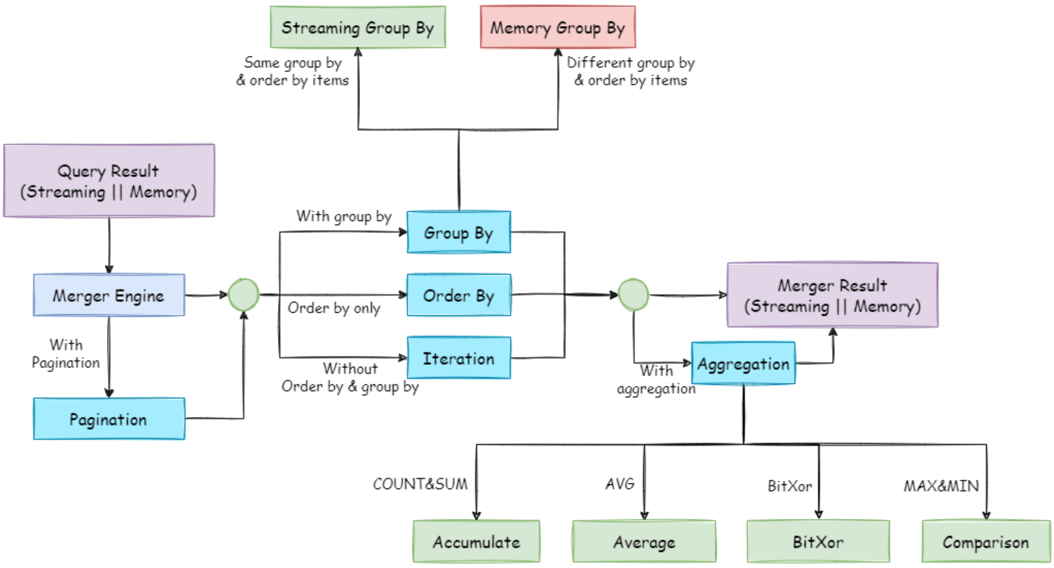
What' s to be noticed, pagination with `LIMIT` is not the best practice actually, because a large amount of data still needs to be transmitted to ShardingSphere' s memory space for ordering. `LIMIT` cannot search for data by index, so paginating with ID is a better solution on the premise that the ID 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:




### 3.1.6 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

Since the SQL syntax is flexible and complex and distributed databases and stand-alone databases do not have identical query scenarios, SQLs incompatible with stand-alone databases are hard to avoid.

This document has listed identified supported SQL types and unsupported SQL types, trying to avoid traps for users.

It is inevitably to have some unlisted SQLs, welcome to supplement for that. We will also try to support those unavailable SQLs in future versions.

### Supported SQL

#### Route to single data node

- 100% compatible (MySQL only, we are completing other databases).

#### Route to multiple data nodes

Fully support DML, DDL, DCL, TCL and some DAL. Support pagination, DISTINCT, ORDER BY, GROUP BY, aggregation and JOIN. Here is an example of a most complex kind of DML:

- Main SELECT

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

- select_expr

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

- table_reference

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

### Unsupported SQL

#### Route to multiple data nodes

Partially support CASE WHEN * `CASE WHEN` containing sub-query is not supported * `CASE WHEN` containing logical-table is not supported(please use alias of table)

Do not support HAVING and UNION (ALL)

Partly available sub-query * If subquery and outer query specify sharding key at the same time, the value of sharding key must be consistent.

Support not only pagination sub-query (see pagination for more details), but also sub-query with the same mode. No matter how many layers are nested, ShardingSphere can parse to the first sub-query that contains data table. Once it finds another sub-query of this kind in the sub-level nested, it will directly throw a parsing exception.

For example, the following sub-query is available:

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

The following sub-query is unavailable:

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

To be simple, through sub-query, non-functional requirements are available in most cases, such as pagination, sum count and so on; but functional requirements are unavailable for now.

Due to the restriction of merger, sub-query that contains aggregation function is unavailable for now.

Do not support SQL that contains schema, for the concept of ShardingSphere is to use multiple data source as one data source, so all the SQL visits are based on one logic schema.

#### Operation for shardingColumn

ShardingColumn in expressions and functions will lead to full routing.

The following SQL is unavailable to single sharding, if `create_time` is shardingColumn:

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

ShardingSphere extract the value of ShardingColumn through `literal` of SQL, so ShardingSphere can not calculate the sharding value from the SQL because the data inside the expression is in database.

When shardingColumn in expressions and functions, ShardingSphere will use full routing to get results.

#### Example

#### Supported SQL

| 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) FROM tbl_name | |
| SELECT subquery_alias.col1 FROM (select tbl_name.col1 from tbl_name where tbl_name.col2=?) subquery_alias | |
| (SELECT * FROM tbl_name) | |
| 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 = ? | The table inserted and the table selected must be the same or bind tables |
| REPLACE INTO tbl_name (col1, col2, …) SELECT col1, col2, …FROM tbl_name WHERE col3 = ? | The table replaced and the table selected must be the same or bind 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 | |

#### Unsupported SQL

| SQL | Reason |
| --- | --- |
| INSERT INTO tbl_name (col1, col2, …) SELECT * FROM tbl_name WHERE col3 = ? | SELECT clause does not support *-shorthand and built-in key generators |
| REPLACE INTO tbl_name (col1, col2, …) SELECT * FROM tbl_name WHERE col3 = ? | SELECT clause does not support *-shorthand and built-in key generators |
| SELECT * FROM tbl_name1 UNION SELECT * FROM tbl_name2 | UNION |
| SELECT * FROM tbl_name1 UNION ALL SELECT * FROM tbl_name2 | UNION ALL |
| SELECT SUM(DISTINCT col1), SUM(col1) FROM tbl_name | See DISTINCT availability detail |
| SELECT * FROM tbl_name WHERE to_date(create_time, 'yyyy-mm-dd' ) = ? | Lead to full routing |
| SELECT MAX(tbl_name.col1) FROM tbl_name | The select function item contains TableName. Otherwise, If this query table had an alias, then TableAlias could work well in select function items. |

#### DISTINCT Availability Explanation

#### Supported SQL

| SQL |
| --- |
| SELECT DISTINCT * FROM tbl_name WHERE col1 = ? |
| SELECT DISTINCT col1 FROM tbl_name |
| SELECT DISTINCT col1, col2, col3 FROM tbl_name |
| SELECT DISTINCT col1 FROM tbl_name ORDER BY col1 |
| SELECT DISTINCT col1 FROM tbl_name ORDER BY col2 |
| SELECT DISTINCT(col1) FROM tbl_name |
| SELECT AVG(DISTINCT col1) FROM tbl_name |
| SELECT SUM(DISTINCT col1) FROM tbl_name |
| SELECT COUNT(DISTINCT col1) FROM tbl_name |
| SELECT COUNT(DISTINCT col1) FROM tbl_name GROUP BY col1 |
| SELECT COUNT(DISTINCT col1 + col2) FROM tbl_name |
| SELECT COUNT(DISTINCT col1), SUM(DISTINCT col1) FROM tbl_name |
| SELECT COUNT(DISTINCT col1), col1 FROM tbl_name GROUP BY col1 |
| SELECT col1, COUNT(DISTINCT col1) FROM tbl_name GROUP BY col1 |

#### Unsupported SQL

| SQL | Reason |
|---|---|
| SELECT SUM(DISTINCT tbl_name.col1), SUM(tbl_name.col1) FROM tbl_name | The select function item contains TableName. Otherwise, If this query table had an alias, then TableAlias could work well in select function items. |

### 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 `1000010 * 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(n)`, 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:

```
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:

```
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:

```
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 ?
```

## 3.2 Distributed Transaction

### 3.2.1 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 |

### 3.2.2 Challenge

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

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

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

### 3.2.3 Goal

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

### 3.2.4 Core Concept

#### Navigation

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

- 2PC transaction with XA
- BASE transaction with Seata

#### XA Transaction

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.

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.

#### Seata BASE transaction

Seata is a distributed transaction framework developed by Alibaba Group and Ant Finance. The goal of AT transaction is to provide incremental transaction ACID semantics under the micro-service architecture, so that developers can use distributed transactions as they use local transactions. The core idea of AT transaction is the same as Apache ShardingSphere.

Seata AT transaction model includes TM (Transaction Manager), RM (Resource Manager) and TC (Transaction Coordinator). TC is an independent service that needs to be deployed separately. TM and RM are deployed together with user applications in the form of jar packages. They establish long connections with TC and keep RPC throughout the transaction life cycle. The initiator of global transaction is TM, which is in charge of begin and commit/rollback of global transaction. The participant of global transaction is RM, which is in charge of reporting the execution results of branch transaction, and commit/rollback is executed through TC coordination.

A typical lifecycle of Seata managed distributed transaction:

1. TM asks TC to begin a new global transaction. TC generates a XID representing the global transaction.
2. XID is propagated through micro-services' invoke chain.
3. RM register local transaction as a branch of the corresponding global transaction of XID to TC.
4. TM asks TC for commit or rollback the corresponding global transaction of XID.
5. TC drives all branch transactions under the corresponding global transaction of XID to finish branch commit or rollback.

### 3.2.5 Principle

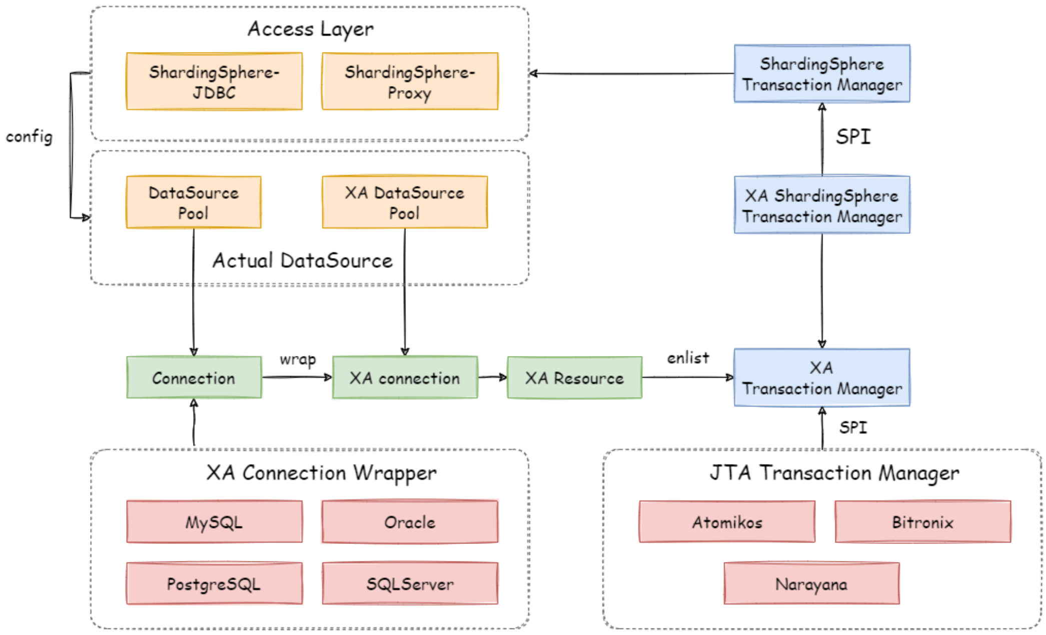
#### Navigation

This chapter mainly introduces the principles of the distributed transactions:

- 2PC transaction with XA
- BASE transaction with Seata

### XA Transaction

`XAShardingTransactionManager` is XA transaction manager of Apache ShardingSphere. Its main responsibly 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, `XAShardingTransactionManager` will use XA transaction managers to start overall XA transactions, which is marked by XID.

### Execute actual sharding SQL

After `XAShardingTransactionManager` 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 `XAShardingTransactionManager` receives the commit command in the access, it will delegate it to the actual XA manager. It will collect all the registered XAResource in the thread, before sending `XAResource.end` to mark the boundary for the XA transaction. Then it will send prepare command one by one to collect votes from XAResource. If all the XAResource feedback is OK, it will send commit command to finally finish it; If there is any No XAResource feedback, it will send rollback command to roll back. After sending the commit command, all XAResource exceptions will be submitted again according to the recovery log to ensure the atomicity and high consistency.

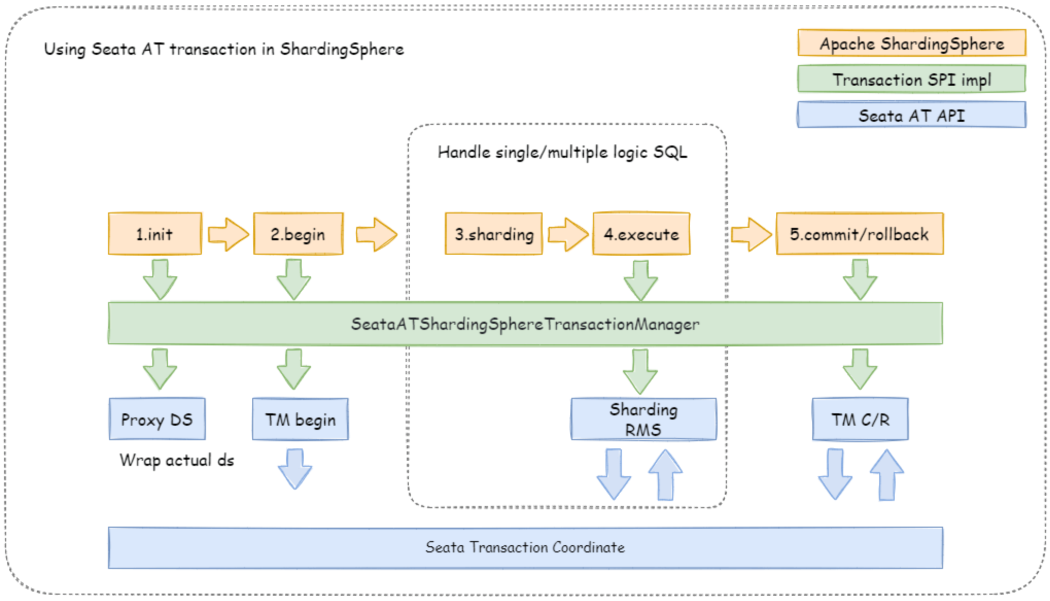
For example:

```
XAResource1.prepare           ## ack: yes
XAResource2.prepare           ## ack: yes
XAResource1.commit
XAResource2.commit

XAResource1.prepare           ## ack: yes
XAResource2.prepare           ## ack: no
XAResource1.rollback
XAResource2.rollback
```

#### Seata BASE transaction

When integrating Seata AT transaction, we need to integrate TM, RM and TC component into Apache Sharding 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 in to Apache ShardingSphere sharding ecosystem.




### Init Seata Engine

When an application containing `ShardingTransactionBaseSeataAT` 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.

### 3.2.6 Use Norms

#### Background

Though Apache ShardingSphere intends to be compatible with all distributed scenario and best performance, under CAP theorem guidance, distributed transaction need to consider about more things.

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

#### Local Transaction

#### Supported Items

- Fully support none-cross-database transactions, for example, sharding table or sharding database with its route result in one database;
- Fully support cross-database transactions caused by logic exceptions, for example, the update of two databases in one transaction, after which, databases will throw null cursor and the content in both databases can be rolled back.

#### Unsupported Items

- Do not support the cross-database transactions caused by network or hardware exceptions. For example, after the update of two databases in one transaction, if one database is down before submitted, then only the data of the other database can be submitted.

#### XA transaction

#### Supported Items

- 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 Items

- Recover committing and rolling back in other machines after the service is down.

#### Seata BASE transaction

#### Supported Items

- 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 Items

- Do not support other isolation level except RC.

#### To Be Optimized Items

- SQL will be parsed twice by Apache ShardingSphere and Seata.

## 3.3 Replica query

### 3.3.1 Background

With increasing system TPS, database capacity has faced great bottleneck effect. For the application system with massive concurrence read operations but less write operations in the same time, we can divide the database into a primary database and a replica database. The primary database is responsible for the addition, deletion and modification of transactions, while the replica database is responsible for queries. It can significantly improve the query performance of the whole system by effectively avoiding line locks caused by data renewal.

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

Different from the horizontal sharding that separates data to all nodes according to sharding keys, replica query routes read operations and write operations separately to the primary database and the replica database according to SQL meaning analysis.

Data in replica query split nodes is consistent, whereas that in horizontal shards is not. The combined use of horizontal sharding and replica query will effectively enhance the system performance.

### 3.3.2 Challenges

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

### 3.3.3 Goal

**The main design goal of the replica query modular of ShardingSphere is to try to reduce the influence of replica query, in order to let users use replica query database group like one database.**

### 3.3.4 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 replica query asynchronization, there may be short-time data inconsistency between them.

#### Load Balance Strategy

Through this strategy, queries are separated to different replica databases.

### 3.3.5 Use Norms

#### Supported Items

- Provide the replica query configuration of one primary database with multiple replica databases, which can be used alone or with sharding table and database;
- Support SQL pass-through in independent use of replica query;

- If there is write operation in the same thread and database connection, all the following read operations are from the primary database to ensure data consistency;
- Forcible primary database route based on SQL Hint;

#### Unsupported Items

- 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 replica query model, the primary nodes need to be used for both reading and writing in the transaction.

## 3.4 Governance

### 3.4.1 Background

As the scale of data continues to expand, a distributed approach using multi-node clusters has gradually become a trend. In this case, how to efficiently and automatically manage cluster nodes, realize the collaborative work of different nodes, configuration consistency, state consistency, high availability, observability, etc., has become a challenge.

This section includes three modules: governance, observability and cluster management(in plan).

### 3.4.2 Challenges

The challenges of distributed governance mainly lie in the complexity of cluster management and how to connect various third-party integrated components in a unified and standard manner.

The complexity of integrated management is reflected in that on the one hand, we need to manage the status of all nodes in a unified manner and can detect the latest changes in real time, whether it is the underlying database node, middleware or business system node, to further provide the basis for the control and scheduling of the cluster. In this regard, we use the cluster topology state diagram to manage the cluster state and the heartbeat detection mechanism to achieve state detection and update.

On the other hand, the unified coordination and the synchronization of policies and rules between different nodes also require us to design a set of global event notification mechanisms and distributed coordination lock mechanisms for exclusive operations in distributed situations. In this regard, we use Zookeeper/Etcd to achieve configuration synchronization, notification of state changes and distributed locks to control exclusive operations.

At the same time, since the governance function itself can use appropriate third-party components as basic services, we need to abstract a unified interface, unify the standard calling APIs of various components and dock to the governance function module.

Finally, for the requirements of manageability and observability, we need to improve the functions of querying, operating and controlling the system through the UI, further improving the support for tracing and APM.

### 3.4.3 Goal

For the governance function, the goals are as follows:

- support Zookeeper/etcd, manage the configuration of data sources, rules and policies, manage the status of each Proxy instances.

For observability, the goals are as follows:

- Support OpenTracing/Skywalking integration and realize call chain tracking.

### 3.4.4 Management

#### Navigation

This chapter mainly introduces the features of the distributed governance:

- Registry center
- Third-party components dependency

#### Registry Center

#### Motivation

- Centralized configuration: more and more running examples have made it hard to manage separate configurations and asynchronized configurations can cause serious problems. Concentrating them in the configuration center can make the management more effective.
- Dynamic configuration: distribution after configuration modification is another important capability of configuration center. It can support dynamic switch between data sources and rule configurations.
- Hold all ephemeral status data dynamically generated in runtime(such as available proxy instances, disabled datasource instances etc).
- Disable the access to replica database and the access of application. Governance still has many functions(such as flow control) to be developed.

### Data Structure in Registry Center

Under defined namespace, `rules`, `props` and `metadata` nodes persist in YAML, modifying nodes can dynamically refresh configurations. `states` 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
   ├──states
   ├     ├──proxynodes
   ├     ├      ├──${your_instance_ip_a}@${your_instance_pid_x}@${UUID}
   ├     ├      ├──${your_instance_ip_b}@${your_instance_pid_y}@${UUID}
   ├     ├      ├──....
   ├     ├──datanodes
   ├     ├      ├──${schema_1}
   ├     ├      ├      ├──${ds_0}
   ├     ├      ├      ├──${ds_1}
   ├     ├      ├──${schema_2}
   ├     ├      ├      ├──${ds_0}
   ├     ├      ├      ├──${ds_1}
   ├     ├      ├──....
```

### /rules

global rule configurations, including configure the username and password for ShardingSphere-Proxy.

```yaml
- !AUTHORITY
users:
  - root@%:root
  - sharding@127.0.0.1:sharding
provider:
  type: NATIVE
```

#### /props

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

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

#### /metadata/${schemeName}/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
    maintenanceIntervalMilliseconds: 30000
    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
    maintenanceIntervalMilliseconds: 30000
    connectionTimeoutMilliseconds: 30000
    idleTimeoutMilliseconds: 60000
    minPoolSize: 1
    username: root
    maxLifetimeMilliseconds: 1800000
```

#### /metadata/${schemeName}/rules

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

```
- !SHARDING
  xxx

- !READWRITE_SPLITTING
```

```
  xxx

- !ENCRYPT
  xxx
```

#### /metadata/${schemeName}/schema

Dynamic modification of metadata content is not supported currently.

```yaml
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
```

#### /states/proxynodes

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

#### /states/datanodes

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

#### Dynamic Effectiveness

Modification, deletion and insertion of relevant configurations in the config center will immediately take effect in the producing environment.

#### Operation Guide

#### Circuit Breaker

Write `DISABLED` (case insensitive) to `IP@PID@UUID` to disable that instance; delete `DISABLED` to enable the instance.

Zookeeper command is as follows:

```
[zk: localhost:2181(CONNECTED) 0] set /${your_zk_namespace}/states/proxynodes/$
{your_instance_ip_a}@${your_instance_pid_x}@${UUID} DISABLED
```

#### Disable Replica Database

Under replica query scenarios, users can write `DISABLED` (case insensitive) to sub-nodes of data source name to disable replica database sources. Delete `DISABLED` or the node to enable it.

Zookeeper command is as follows:

```
[zk: localhost:2181(CONNECTED) 0] set /${your_zk_namespace}/states/datanodes/$
{your_schema_name}/${your_replica_datasource_name} DISABLED
```

#### Third-party Components

Apache ShardingSphere uses SPI to load data to the config center and registry center and disable instances and databases. Currently, Apache ShardingSphere supports frequently used registry centers, Zookeeper, Etcd, Apollo and Nacos. In addition, by injecting them to ShardingSphere with SPI, users can use other third-party config and registry centers to enable databases governance.

| | *Driver* | *Version* | *Config Center* | *Registry Center* |
|---|---|---|---|---|
| ` Zookeeper < https://zookeeper.apache.org/>`__ | Apache Curator r | 3.6.x | Support | Support |
| `**Etcd** <https://etcd.io/>`__ | jetcd | v3 | Support | Support |

#### Change History

##### 5.0.0-alpha

###### Config Center

###### Structure in Configuration Center

```
namespace
├──users                                      # Users configuration
├──props                                      # Properties configuration
├──schemas                                    # Schema configuration
├      ├──${schema_1}                         # Schema name 1
├      ├      ├──datasource                   # Datasource configuration
├      ├      ├──rule                         # Rule configuration
├      ├      ├──table                        # Table configuration
├      ├──${schema_2}                         # Schema name 2
├      ├      ├──datasource                   # Datasource configuration
├      ├      ├──rule                         # Rule configuration
├      ├      ├──table                        # Table configuration
```

###### Registry Center

###### Data Structure in Registry Center

```
namespace
   ├──states
   ├    ├──proxynodes
   ├    ├     ├──${your_instance_ip_a}@${your_instance_pid_x}@${UUID}
   ├    ├     ├──${your_instance_ip_b}@${your_instance_pid_y}@${UUID}
   ├    ├     ├──....
   ├    ├──datanodes
   ├    ├     ├──${schema_1}
   ├    ├     ├      ├──${ds_0}
```

```
  ├    ├     ├      ├──${ds_1}
  ├    ├     ├──${schema_2}
  ├    ├     ├      ├──${ds_0}
  ├    ├     ├      ├──${ds_1}
  ├    ├     ├──....
```

### 3.4.5 Observability

#### Navigation

This chapter mainly introduces the features of the observability:

- APM Integration

#### APM Integration

#### Background

APM is the abbreviation for application performance monitoring. Currently, main APM functions lie in the 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 two ways.

The first way is to send performance tracing data by 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.

The second way is to 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`.

### 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.

### 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

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

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

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

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

#### Exception

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.

### Agent Integration

#### Background

ShardingSphere-Agent is an independent and independently designed project based on ByteBuddy bytecode increase. Based on plugin design, it can integrate seamlessly with ShardingSphere. There are currently Log, metrics, APM and other observability capabilities available.

#### Usage

#### Local build

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

#### Remote download（No release）

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

Add startup arguments

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

#### Agent Configuration

It is found under the local package directory and unzip directory : `agent.yaml`

```yaml applicationName: shardingsphere-agent # application name ignoredPluginNames: # A collection of ignored plugins, indicating that the plugins in the collection are not active - Opentracing - Jaeger - Zipkin - Prometheus - Logging

plugins: Prometheus: host: "localhost" #prometheus host port: 9090 #prometheus port props: JVM_INFORMATION_COLLECTOR_ENABLED : "true" Jaeger: host: "localhost" #jaeger host port: 5775 #jaeger prot 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" #zipkin host port: 9411 #zipkin prot props: SERVICE_NAME: "shardingsphere-agent" URL_VERSION: "/api/v2/spans" #zipkin uri Logging: props: LEVEL: "INFO" #log level

``

When ignoredPluginNames is configured, plugins in the collection are ignored!

## 3.5 Scaling

### 3.5.1 Background

Apache ShardingSphere provides data sharding capability, which can split data to different databases.

For applications that have been running with stand-alone database, there is a problem how to migrate data to sharding data nodes safely and simply; For some applications which have used Apache ShardingSphere, the rapid growth of data may also cause a single data node or even the entire data nodes to reach a bottleneck. How to expand their data nodes for Apache ShardingSphere cluster also became a problem.

### 3.5.2 Introduction

ShardingSphere-Scaling is a common solution for migrating or scaling data in Apache ShardingSphere since **4.1.0**.

### 3.5.3 Challenges

Apache ShardingSphere provides users with great freedom in sharding strategies and algorithms, but it gives a great challenge to scaling. So it's the first challenge that how to find a way can support kinds of sharding strategies and 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.

### 3.5.4 Goal

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

### 3.5.5 Status

current is in **alpha** development.

### 3.5.6 Core Concept

#### Scaling Job

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

#### Data Node

Same as the Data Node in sharding/SQL.

#### 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.

### 3.5.7 Principle

#### 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.

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.

### 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.

#### 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.

### 3.5.8 User Norms

#### Supported Items

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

#### Unsupported Items

- Do not support to scale tables without primary key.

## 3.6 Encryption

### 3.6.1 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 clean data.
- 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

### 3.6.2 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.

### 3.6.3 Goal

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

### 3.6.4 Core Concept

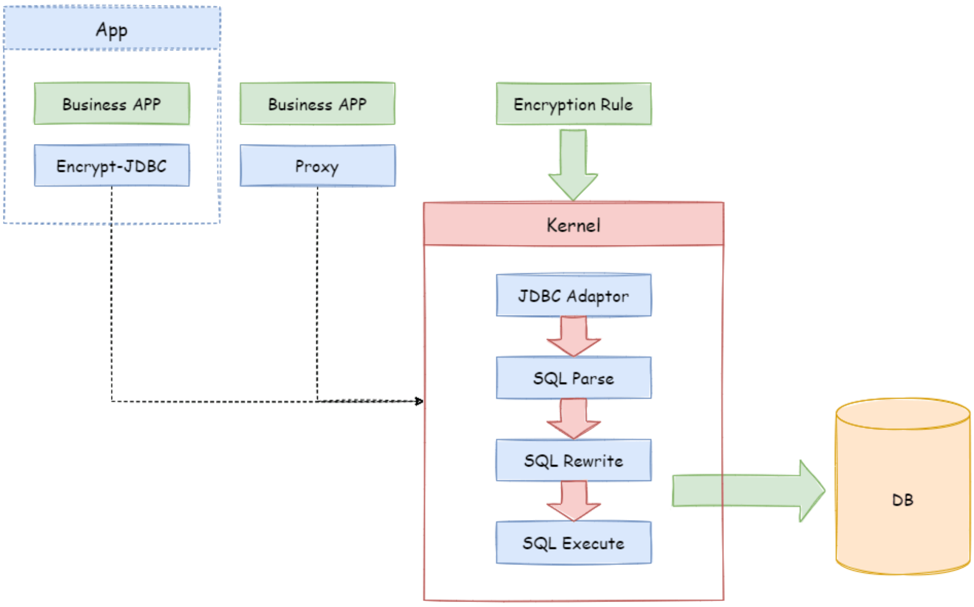
TODO

### 3.6.5 Principle

#### 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.

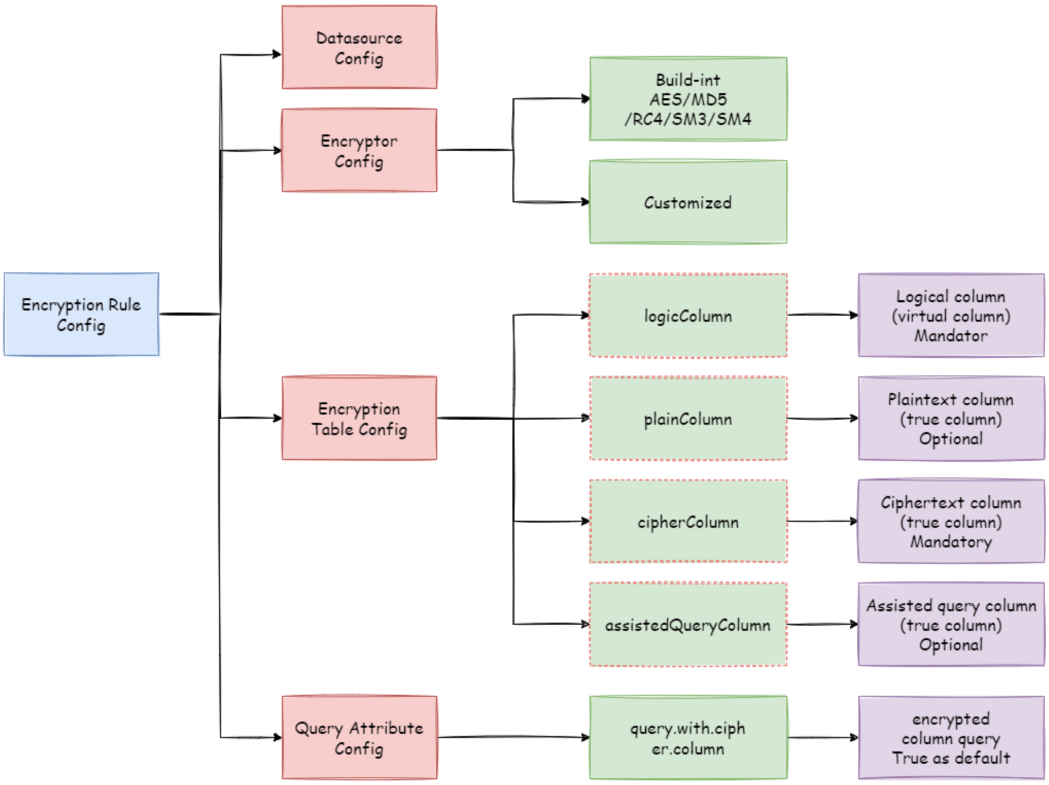
### Overall Architecture




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:




**Datasource Configuration**: The configuration of DataSource.

**Encrypt Algorithm Configuration**: What kind of encryption strategy to use for encryption and decryption. Currently ShardingSphere has two built-in encryption/decryption strategies: AES / MD5. 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)

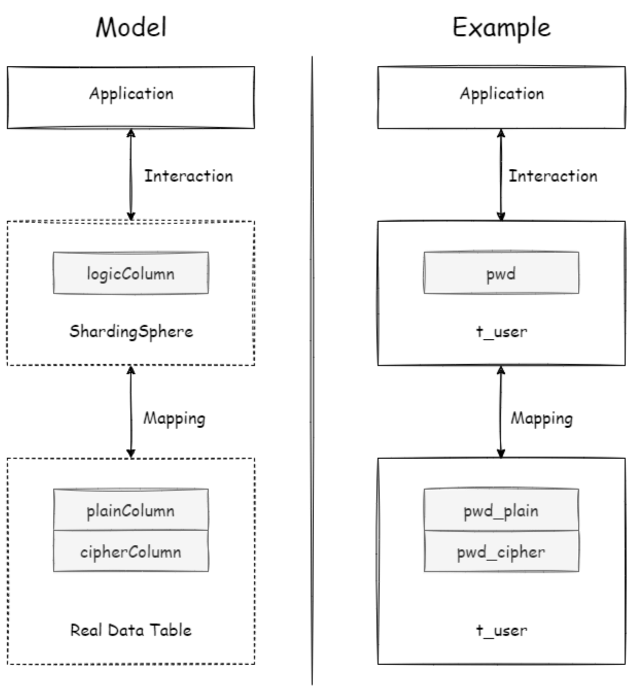
> How to understand `Which column do users want to use to write SQL (logicColumn)`?
>
> We can understand according to the meaning of Apache ShardingSphere. The ultimate goal of Apache ShardingSphere is to shield the encryption of the underlying data, that is, we do not want users to know how the data 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.

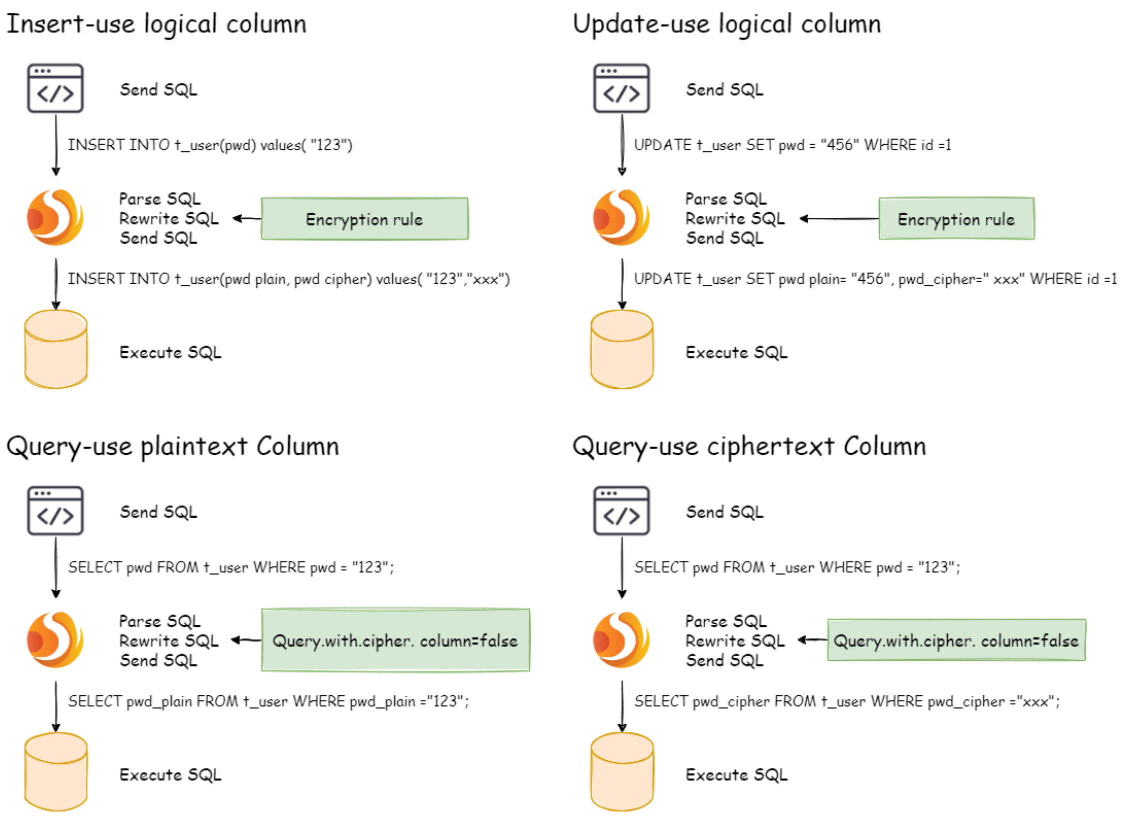
#### Encryption Process

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.




### 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?

### New Business

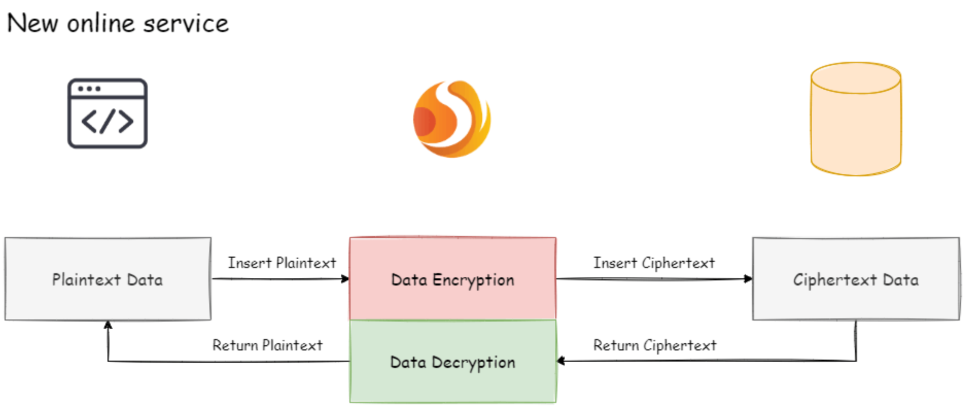
Business scenario analysis: The newly launched business is relatively simple because everything starts from scratch and there is no historical data cleaning problem.

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:




### 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.

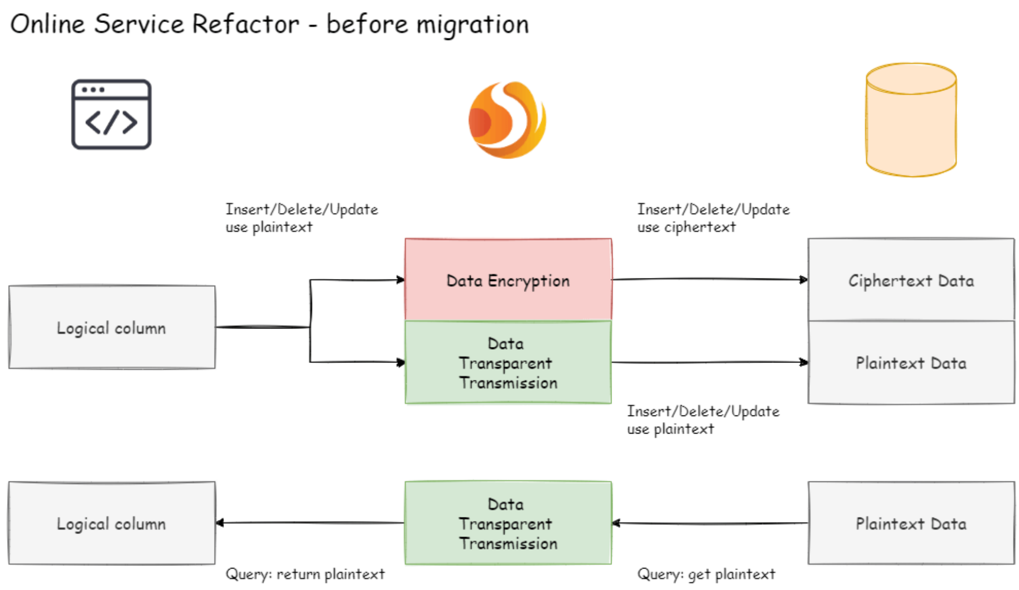
Business developers must hope: reduce the burden of capital costs, do not modify the business code, and be able to safely and smoothly migrate the system. So, the encryption function module of ShardingSphere was born. It can be divided into three steps:

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:

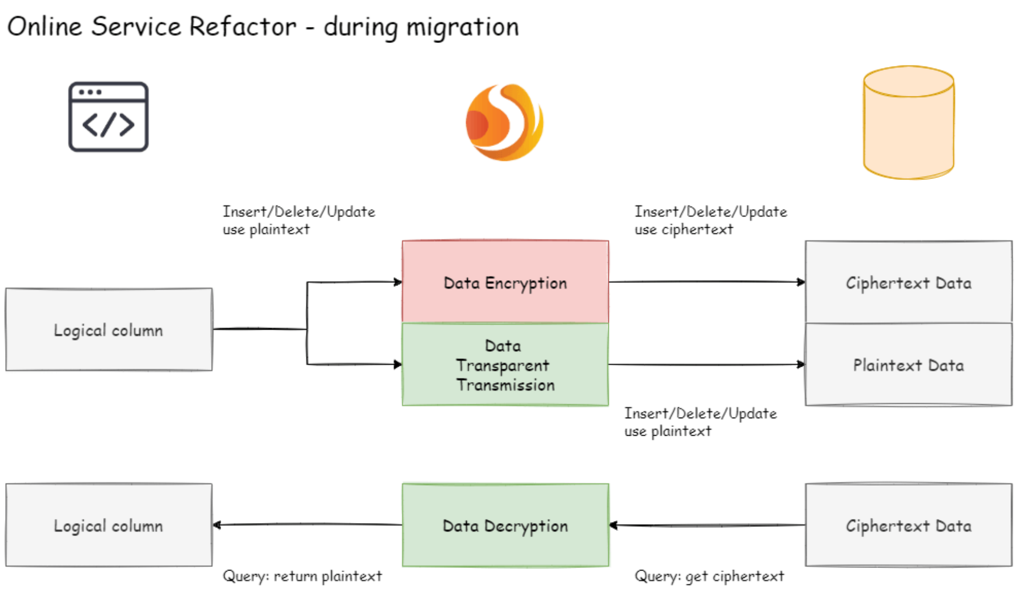



When the newly added data is inserted, it is encrypted as ciphertext data through Apache Sharding-Sphere and stored in the cipherColumn. Now it is necessary to process historical plaintext inventory data. **As Apache ShardingSphere currently does not provide the corresponding migration and washing tools, 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 `queryWithCipherColumn` 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

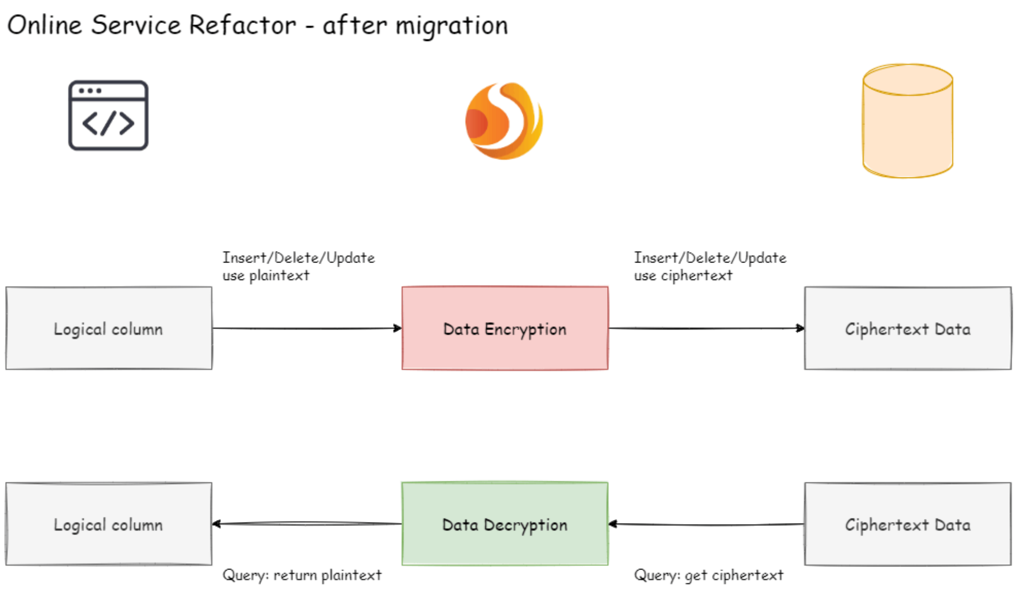
Due to the requirements of the security audit department, it is generally impossible for the business system to keep the plaintext and ciphertext columns of the database permanently synchronized. We need to delete the plaintext data after the system is stable. That is, we need to delete plainColumn (ie 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.

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 and pwd_cipher transformation mapping
          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 fulfil business requirements. The solution has been continuously launched on JD Digits, providing internal basic service support.

### 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.

### 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 two types of implementations for this kind of encrypt solution, MD5 (irreversible) and AES (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.

### 3.6.6 Use Norms

#### Supported Items

- The back-end databases are MySQL, Oracle, PostgreSQL, and SQLServer;
- The user needs to encrypt one or more columns in the database table (data encryption & decryption);
- Compatible with all commonly used SQL.

#### Unsupported Items

- Users need to deal with the original inventory data and wash numbers in the database;
- Use encryption function + sub-database sub-table function, some special SQL is not supported, please refer to SQL specification;
- Encryption fields cannot support comparison operations, such as: greater than less than, ORDER BY, BETWEEN, LIKE, etc;
- Encryption fields cannot support calculation operations, such as AVG, SUM, and calculation expressions.

## 3.7 Shadow DB

### 3.7.1 Navigation

TODO

### 3.7.2 Core Concept

TODO

### 3.7.3 Principle

TODO

## 3.8 Dist SQL

### 3.8.1 Background

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

### 3.8.2 Challenges

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

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

DistSQL is divided into RDL, RQL and SCTL.

- 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;
- SCTL(ShardingSphere Control Language)responsible for the added-on feature of hint, transaction type switch, sharding execute planning and so on.

### 3.8.3 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.**

### 3.8.4 Notice

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

### 3.8.5 Syntax

#### TODO

#### RDL Syntax

#### TODO

#### Data Source

#### TODO

#### Sharding

#### TODO

#### Readwrite-Splitting

#### TODO

#### Encrypt

#### TODO

#### DB Discovery

#### TODO

#### RQL Syntax

##### TODO

#### Data Source

##### TODO

#### Sharding

##### TODO

#### Readwrite-Splitting

##### TODO

#### Encrypt

##### TODO

#### DB Discovery

##### TODO

#### SCTL Syntax

##### TODO

#### SCTL

##### Definition

SCTL（ShardingSphere Control Language）responsible for the added-on feature of hint, transaction type switch, sharding execute planning and so on.

### Usage

| Statement | Function | Example |
|---|---|---|
| sctl:set transact ion_type=XX | Modify transaction_type of the current connection, supports LOCAL, XA, BASE | sctl:set transa ction_type=XA |
| sctl:show trans action_type | Query the transaction type of the current connection | sctl:show tra nsaction_type |
| sctl:show **cached_** connections | Query the number of cached physical database connections in the current connection | sctl:show cache d_connections |
| s ctl:explain SQL | View the execution plan for logical SQL. | sctl:explain select * from t_order |
| sctl:hint set PRIMAR Y_ONLY=true | For current connection, set database operation force route to primary database only or not | sctl:hint set PRIM ARY_ONLY=true |
| sctl:hint set Da tabaseShard ing-Value=yy | For current connection, set sharding value for database sharding only, yy: sharding value | sctl:hint set Database-Shar dingValue=100 |
| sctl:hint ad dDatabaseSh ardingValue xx=yy | For current connection, add sharding value for database, xx: logic table, yy: sharding value | sctl:hint addDatabase ShardingValue t_order=100 |
| sctl:hint addTa-bleSh ardingValue xx=yy | For current connection, add sharding value for table, xx: logic table, yy: sharding value | sctl:hint addTable ShardingValue t_order=100 |
| sctl:hint clear | For current connection, clear all hint settings | sctl:hint clear |
| sctl:hint show status | For current connection, query hint status, primary_only:true/false, sharding _type:databases_only/databases_tables | sctl:hint show status |
| sctl:hint show table status | For current connection, query sharding values of logic tables | sctl:hint show table status |

### Notice

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

## 3.8.6 Usage

### TODO

### Sharding

#### TODO

## 3.9 Pluggable Architecture

### 3.9.1 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.

### 3.9.2 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, replica query, 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.

### 3.9.3 Goal

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

## 3.10 Test Engine

Apache ShardingSphere provided a full functionality test engine. They define SQLs in xml files, each engine is suit for MySQL, PostgreSQL, SQLServer and Oracle.

All test engines designed to modify the configuration files to execute all assertions without any **Java code** modification.

### 3.10.1 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 replica query 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.

### 3.10.2 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.

### 3.10.3 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:
```

```yaml
## 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.

## 3.10.4 Performance Test

### 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, replica query & encrypt & sharding, full route). While another set of associated operation, INSERT & SELECT & DELETE, is used to evaluate performance for replica query. 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.

#### Replica Query

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.

#### Replica Query & 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

```
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:
```

```yaml
    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
    idleTimeoutMilliseconds: 60000
    maxLifetimeMilliseconds: 1800000
    maxPoolSize: 200
rules:
- !READWRITE_SPLITTING
  dataSources:
    pr_ds:
      writeDataSourceName: primary_ds
      readDataSourceNames:
        - replica_ds_0
```

#### Replica Query & Encrypt & Sharding Configuration

```
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
    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
```

```yaml
    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
        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:
```

```yaml
      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:
      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.

### 3.10.5 Performance Test(sysbench)

#### Environment

#### Recommended Hardware

```
CPU: 32 Cores
RAM: 128 GB
NIC: 10Gb Ethernet
```

At least 5 machines are required:

```
Jenkins * 1: ${host-jenkins}
Sysbench * 1: ${host-sysbench}
ShardingSphere-Proxy * 1: ${host-proxy}
MySQL Server * 2: ${host-mysql-1}, ${host-mysql-2}
```

The hardware standards of Jenkins and Sysbench machines can appropriately lower.

#### Software Environment

```
Jenins: The latest version
Sysbench: 1.0.20
ShardingSphere-Proxy: package from master branch
MySQL Server: 5.7.28
```

#### Test Program

According to the above hardware environment, the configuration parameters are as follows, and the parameters should be adjusted according to the changes in the hardware environment.

#### ShardingSphere-Proxy Configuration

```
Proxy runs on ${host-proxy}
Version includes: Master branch, 4.1.1, 3.0.0
Scenarios: config-sharding, config-replica-query, config-sharding-replica-query,
config-encrypt
Configurations: Refer to Appendix 1
```

#### MySQL Server Configuration

Two MySQL instances runs on `${host-mysql-1}` and `${host-mysql-2}` machines respectively.

```
Need to create the 'sbtest' database on both instances in advance.
Set parameter: max_prepared_stmt_count = 500000
Set parameter: max_connections = 2000
```

### Jenkins Configuration

Create 6 Jenkins tasks, and each task calls the next task in turn: (runs on the `${host-jenkins}` machine).

```
1. sysbench_install: Pull the latest code, package the Proxy compression package
```

The following tasks are run on a separate Sysbench pressure generating machine via Jenkins slave: (runs on the `{host-sysbench}` machine)

```
2. sysbench_sharding:
   a. Sharding scenarios for remote deployment of various versions of Proxy
   b. Execute Sysbench command to pressure test Proxy
   c. Execute Sysbench command to pressure test MySQL Server
   d. Save Sysbench stress test results
   e. Use drawing scripts to generate performance curves and tables (see Appendix 2
for drawing scripts)
3. sysbench_master_slave:
   a. Read and write separation scenarios for remote deployment of various versions
of Proxy
   b. Execute Sysbench command to pressure test Proxy
   c. Execute Sysbench command to pressure test MySQL Server
   d. Save Sysbench stress test results
   e. Use drawing scripts to generate performance curves and tables
4. sysbench_sharding_master_slave:
   a. Remote deployment of sharding + read-write splitting scenarios of various
versions of Proxy
   b. Execute Sysbench command to pressure test Proxy
   c. Execute Sysbench command to pressure test MySQL Server
   d. Save Sysbench stress test results
   e. Use drawing scripts to generate performance curves and tables
5. sysbench_encrypt:
   a. Encryption scenarios for remote deployment of various versions of Proxy
   b. Execute Sysbench command to pressure test Proxy
   c. Execute Sysbench command to pressure test MySQL Server
   d. Save Sysbench stress test results
   e. Use drawing scripts to generate performance curves and tables
6. sysbench_result_aggregation:
   a. Re-execute the drawing script for the pressure test results of all tasks
      python3 plot_graph.py sharding
      python3 plot_graph.py ms
      python3 plot_graph.py sharding_ms
      python3 plot_graph.py encrypt
   b. Use Jenkins "Publish HTML reports" plugin to integrate all images into one
HTML page
```

#### Testing Process

Take sysbench sharding as an example (other scenarios are similar)

##### Enter the Sysbench pressure test result directory

```
cd /home/jenkins/sysbench_res/sharding
```

##### Create the folder for this build

```
mkdir $BUILD_NUMBER
```

##### Take the last 14 builds and save them in a hidden file

```
ls -v | tail -n14 > .build_number.txt
```

##### Deployment and stress testing

Step 1: Execute remote deployment script to deploy Proxy to `{host-proxy}`

./deploy_sharding.sh

```
#!/bin/sh

rm -fr apache-shardingsphere-*-shardingsphere-proxy-bin
tar zxvf apache-shardingsphere-*-shardingsphere-proxy-bin.tar.gz

sh stop_proxy.sh

cp -f prepared_conf/mysql-connector-java-5.1.47.jar apache-shardingsphere-*-
shardingsphere-proxy-bin/lib
cp -f prepared_conf/start.sh apache-shardingsphere-*-shardingsphere-proxy-bin/bin
cp -f prepared_conf/config-sharding.yaml prepared_conf/server.yaml apache-
shardingsphere-*-shardingsphere-proxy-bin/conf

./apache-shardingsphere-*-shardingsphere-proxy-bin/bin/start.sh

sleep 30
```

Step 2: Execute the sysbench script

```
# master

cd /home/jenkins/sysbench_res/sharding
```

```
cd $BUILD_NUMBER

sysbench oltp_read_only --mysql-host=${host-proxy} --mysql-port=3307 --mysql-
user=root --mysql-password='root' --mysql-db=sbtest --tables=10 --table-
size=1000000 --report-interval=10 --time=3600 --threads=10 --max-requests=0 --
percentile=99 --mysql-ignore-errors="all" --rand-type=uniform --range_selects=off -
-auto_inc=off cleanup
sysbench oltp_read_only --mysql-host=${host-proxy} --mysql-port=3307 --mysql-
user=root --mysql-password='root' --mysql-db=sbtest --tables=10 --table-
size=1000000 --report-interval=10 --time=3600 --threads=10 --max-requests=0 --
percentile=99 --mysql-ignore-errors="all" --rand-type=uniform --range_selects=off -
-auto_inc=off prepare

sysbench oltp_read_only        --mysql-host=${host-proxy} --mysql-port=3307 --
mysql-user=root --mysql-password='root' --mysql-db=sbtest --tables=10 --table-
size=1000000 --report-interval=30  --time=180 --threads=256 --max-requests=0 --
percentile=99  --mysql-ignore-errors="all" --range_selects=off --rand-type=uniform
--auto_inc=off run
sysbench oltp_read_only        --mysql-host=${host-proxy} --mysql-port=3307 --
mysql-user=root --mysql-password='root' --mysql-db=sbtest --tables=10 --table-
size=1000000 --report-interval=30  --time=180 --threads=256 --max-requests=0 --
percentile=99  --mysql-ignore-errors="all" --range_selects=off --rand-type=uniform
--auto_inc=off run | tee oltp_read_only.master.txt
sysbench oltp_point_select     --mysql-host=${host-proxy} --mysql-port=3307 --
mysql-user=root --mysql-password='root' --mysql-db=sbtest --tables=10 --table-
size=1000000 --report-interval=30  --time=180 --threads=256 --max-requests=0 --
percentile=99  --mysql-ignore-errors="all" --range_selects=off --rand-type=uniform
--auto_inc=off run | tee oltp_point_select.master.txt
sysbench oltp_read_write       --mysql-host=${host-proxy} --mysql-port=3307 --
mysql-user=root --mysql-password='root' --mysql-db=sbtest --tables=10 --table-
size=1000000 --report-interval=30  --time=180 --threads=256 --max-requests=0 --
percentile=99  --mysql-ignore-errors="all" --range_selects=off --rand-type=uniform
--auto_inc=off run | tee oltp_read_write.master.txt
sysbench oltp_write_only       --mysql-host=${host-proxy} --mysql-port=3307 --
mysql-user=root --mysql-password='root' --mysql-db=sbtest --tables=10 --table-
size=1000000 --report-interval=30  --time=180 --threads=256 --max-requests=0 --
percentile=99  --mysql-ignore-errors="all" --range_selects=off --rand-type=uniform
--auto_inc=off run | tee oltp_write_only.master.txt
sysbench oltp_update_index     --mysql-host=${host-proxy} --mysql-port=3307 --
mysql-user=root --mysql-password='root' --mysql-db=sbtest --tables=10 --table-
size=1000000 --report-interval=30  --time=180 --threads=256 --max-requests=0 --
percentile=99  --mysql-ignore-errors="all" --range_selects=off --rand-type=uniform
--auto_inc=off run | tee oltp_update_index.master.txt
sysbench oltp_update_non_index --mysql-host=${host-proxy} --mysql-port=3307 --
mysql-user=root --mysql-password='root' --mysql-db=sbtest --tables=10 --table-
size=1000000 --report-interval=30  --time=180 --threads=256 --max-requests=0 --
percentile=99  --mysql-ignore-errors="all" --range_selects=off --rand-type=uniform
--auto_inc=off run | tee oltp_update_non_index.master.txt
```

```
sysbench oltp_delete           --mysql-host=${host-proxy} --mysql-port=3307 --
mysql-user=root --mysql-password='root' --mysql-db=sbtest --tables=10 --table-
size=1000000 --report-interval=30  --time=180 --threads=256 --max-requests=0 --
percentile=99  --mysql-ignore-errors="all" --range_selects=off --rand-type=uniform
--auto_inc=off run | tee oltp_delete.master.txt

sysbench oltp_read_only --mysql-host=${host-proxy} --mysql-port=3307 --mysql-
user=root --mysql-password='root' --mysql-db=sbtest --tables=10 --table-
size=1000000 --report-interval=10 --time=3600 --threads=10 --max-requests=0 --
percentile=99 --mysql-ignore-errors="all" --rand-type=uniform --range_selects=off -
-auto_inc=off cleanup
```

4.1.1, 3.0.0, three scenarios of direct connection to MySQL, repeat steps 1 and 2 above.

#### Execute stop proxy script

./stop_proxy.sh

```
#!/bin/sh

./3.0.0_sharding-proxy/bin/stop.sh
./4.1.1_apache-shardingsphere-4.1.1-sharding-proxy-bin/bin/stop.sh
./apache-shardingsphere-*-shardingsphere-proxy-bin/bin/stop.sh
```

#### Generate pressure test curve picture

```
# Generate graph

cd /home/jenkins/sysbench_res/
python3 plot_graph.py sharding
```

#### Use Jenkins Publish HTML reports plugin to publish pictures to the page

```
HTML directory to archive: /home/jenkins/sysbench_res/graph/
Index page[s]: 01_sharding.html
Report title: HTML Report
```

#### sysbench test case describe

##### oltp_point_select

```
Prepare Statement (ID = 1): SELECT c FROM sbtest1 WHERE id=?
Execute Statement: ID = 1
```

##### oltp_read_only

```
Prepare Statement (ID = 1): 'COMMIT'
Prepare Statement (ID = 2): SELECT c FROM sbtest1 WHERE id=?

Statement: 'BEGIN'
Execute Statement: ID = 2
Execute Statement: ID = 2
Execute Statement: ID = 2
Execute Statement: ID = 2
Execute Statement: ID = 2
Execute Statement: ID = 2
Execute Statement: ID = 2
Execute Statement: ID = 2
Execute Statement: ID = 2
Execute Statement: ID = 2
Execute Statement: ID = 1
```

##### oltp_write_only

```
Prepare Statement (ID = 1): 'COMMIT'
Prepare Statement (ID = 2): UPDATE sbtest1 SET k=k+1 WHERE id=?
Prepare Statement (ID = 3): UPDATE sbtest6 SET c=? WHERE id=?
Prepare Statement (ID = 4): DELETE FROM sbtest1 WHERE id=?
Prepare Statement (ID = 5): INSERT INTO sbtest1 (id, k, c, pad) VALUES (?, ?, ?, ?)

Statement: 'BEGIN'
Execute Statement: ID = 2
Execute Statement: ID = 3
Execute Statement: ID = 4
Execute Statement: ID = 5
Execute Statement: ID = 1
```

#### oltp_read_write

```
Prepare Statement (ID = 1): 'COMMIT'
Prepare Statement (ID = 2): SELECT c FROM sbtest1 WHERE id=?
Prepare Statement (ID = 3): UPDATE sbtest3 SET k=k+1 WHERE id=?
Prepare Statement (ID = 4): UPDATE sbtest10 SET c=? WHERE id=?
Prepare Statement (ID = 5): DELETE FROM sbtest8 WHERE id=?
Prepare Statement (ID = 6): INSERT INTO sbtest8 (id, k, c, pad) VALUES (?, ?, ?, ?)

Statement: 'BEGIN'
Execute Statement: ID = 2
Execute Statement: ID = 2
Execute Statement: ID = 2
Execute Statement: ID = 2
Execute Statement: ID = 2
Execute Statement: ID = 2
Execute Statement: ID = 2
Execute Statement: ID = 2
Execute Statement: ID = 2
Execute Statement: ID = 2
Execute Statement: ID = 3
Execute Statement: ID = 4
Execute Statement: ID = 5
Execute Statement: ID = 6
Execute Statement: ID = 1
```

#### oltp_update_index

```
Prepare Statement (ID = 1): UPDATE sbtest1 SET k=k+1 WHERE id=?

Execute Statement: ID = 1
```

#### oltp_update_non_index

```
Prepare Statement (ID = 1): UPDATE sbtest1 SET c=? WHERE id=?

Execute Statement: ID = 1
```

#### oltp_delete

```
Prepare Statement (ID = 1): DELETE FROM sbtest1 WHERE id=?

Execute Statement: ID = 1
```

### Appendix 1

#### Master branch version

server.yaml

```
users:
  - root@%:root
  - sharding@:sharding

props:
  max-connections-size-per-query: 10
  executor-size: 128  # Infinite by default.
  proxy-frontend-flush-threshold: 128  # The default value is 128.
    # LOCAL: Proxy will run with LOCAL transaction.
    # XA: Proxy will run with XA transaction.
    # BASE: Proxy will run with B.A.S.E transaction.
  proxy-transaction-type: LOCAL
  proxy-opentracing-enabled: false
  proxy-hint-enabled: false
  sql-show: false
  check-table-metadata-enabled: false
  lock-wait-timeout-milliseconds: 50000 # The maximum time to wait for a lock
```

config-sharding.yaml

```
schemaName: sbtest

dataSources:
  ds_0:
    url: jdbc:mysql://${host-mysql-1}:3306/sbtest?serverTimezone=UTC&useSSL=false
    username: root
    password:
    connectionTimeoutMilliseconds: 30000
    idleTimeoutMilliseconds: 60000
    maxLifetimeMilliseconds: 1800000
    maxPoolSize: 256
    minPoolSize: 256
    maintenanceIntervalMilliseconds: 30000
  ds_1:
    url: jdbc:mysql://${host-mysql-2}:3306/sbtest?serverTimezone=UTC&useSSL=false
```

```yaml
    username: root
    password:
    connectionTimeoutMilliseconds: 30000
    idleTimeoutMilliseconds: 60000
    maxLifetimeMilliseconds: 1800000
    maxPoolSize: 256
    minPoolSize: 256
    maintenanceIntervalMilliseconds: 30000

rules:
- !SHARDING
  tables:
    sbtest1:
      actualDataNodes: ds_${0..1}.sbtest1_${0..99}
      tableStrategy:
        standard:
          shardingColumn: id
          shardingAlgorithmName: table_inline_1
      keyGenerateStrategy:
        column: id
        keyGeneratorName: snowflake
    sbtest2:
      actualDataNodes: ds_${0..1}.sbtest2_${0..99}
      tableStrategy:
        standard:
          shardingColumn: id
          shardingAlgorithmName: table_inline_2
      keyGenerateStrategy:
        column: id
        keyGeneratorName: snowflake
    sbtest3:
      actualDataNodes: ds_${0..1}.sbtest3_${0..99}
      tableStrategy:
        standard:
          shardingColumn: id
          shardingAlgorithmName: table_inline_3
      keyGenerateStrategy:
        column: id
        keyGeneratorName: snowflake
    sbtest4:
      actualDataNodes: ds_${0..1}.sbtest4_${0..99}
      tableStrategy:
        standard:
          shardingColumn: id
          shardingAlgorithmName: table_inline_4
      keyGenerateStrategy:
        column: id
        keyGeneratorName: snowflake
```

```
    sbtest5:
      actualDataNodes: ds_${0..1}.sbtest5_${0..99}
      tableStrategy:
        standard:
          shardingColumn: id
          shardingAlgorithmName: table_inline_5
      keyGenerateStrategy:
        column: id
        keyGeneratorName: snowflake
    sbtest6:
      actualDataNodes: ds_${0..1}.sbtest6_${0..99}
      tableStrategy:
        standard:
          shardingColumn: id
          shardingAlgorithmName: table_inline_6
      keyGenerateStrategy:
        column: id
        keyGeneratorName: snowflake
    sbtest7:
      actualDataNodes: ds_${0..1}.sbtest7_${0..99}
      tableStrategy:
        standard:
          shardingColumn: id
          shardingAlgorithmName: table_inline_7
      keyGenerateStrategy:
        column: id
        keyGeneratorName: snowflake
    sbtest8:
      actualDataNodes: ds_${0..1}.sbtest8_${0..99}
      tableStrategy:
        standard:
          shardingColumn: id
          shardingAlgorithmName: table_inline_8
      keyGenerateStrategy:
        column: id
        keyGeneratorName: snowflake
    sbtest9:
      actualDataNodes: ds_${0..1}.sbtest9_${0..99}
      tableStrategy:
        standard:
          shardingColumn: id
          shardingAlgorithmName: table_inline_9
      keyGenerateStrategy:
        column: id
        keyGeneratorName: snowflake
    sbtest10:
      actualDataNodes: ds_${0..1}.sbtest10_${0..99}
      tableStrategy:
```

```yaml
        standard:
          shardingColumn: id
          shardingAlgorithmName: table_inline_10
      keyGenerateStrategy:
        column: id
        keyGeneratorName: snowflake

  defaultDatabaseStrategy:
    standard:
      shardingColumn: id
      shardingAlgorithmName: database_inline

  shardingAlgorithms:
    database_inline:
      type: INLINE
      props:
        algorithm-expression: ds_${id % 2}
    table_inline_1:
      type: INLINE
      props:
        algorithm-expression: sbtest1_${id % 100}
    table_inline_2:
      type: INLINE
      props:
        algorithm-expression: sbtest2_${id % 100}
    table_inline_3:
      type: INLINE
      props:
        algorithm-expression: sbtest3_${id % 100}
    table_inline_4:
      type: INLINE
      props:
        algorithm-expression: sbtest4_${id % 100}
    table_inline_5:
      type: INLINE
      props:
        algorithm-expression: sbtest5_${id % 100}
    table_inline_6:
      type: INLINE
      props:
        algorithm-expression: sbtest6_${id % 100}
    table_inline_7:
      type: INLINE
      props:
        algorithm-expression: sbtest7_${id % 100}
    table_inline_8:
      type: INLINE
      props:
```

```
        algorithm-expression: sbtest8_${id % 100}
    table_inline_9:
      type: INLINE
      props:
        algorithm-expression: sbtest9_${id % 100}
    table_inline_10:
      type: INLINE
      props:
        algorithm-expression: sbtest10_${id % 100}
  keyGenerators:
    snowflake:
      type: SNOWFLAKE
      props:
        worker-id: 123
```

config-replica-query.yaml

```
schemaName: sbtest

dataSources:
  ds_0:
    url: jdbc:mysql://${host-mysql-1}:3306/sbtest?serverTimezone=UTC&useSSL=false
    username: root
    password:
    connectionTimeoutMilliseconds: 30000
    idleTimeoutMilliseconds: 60000
    maxLifetimeMilliseconds: 1800000
    maxPoolSize: 128
    minPoolSize: 128
    maintenanceIntervalMilliseconds: 30000

rules:
- !READWRITE_SPLITTING
  dataSources:
    pr_ds:
      primaryDataSourceName: ds_0
      replicaDataSourceNames:
        - ds_0
        - ds_0
```

config-sharding-replica-query.yaml

```
schemaName: sbtest

dataSources:
  primary_ds_0:
    url: jdbc:mysql://${host-mysql-1}:3306/sbtest?serverTimezone=UTC&useSSL=false
    username: root
```

```yaml
    password:
    connectionTimeoutMilliseconds: 30000
    idleTimeoutMilliseconds: 60000
    maxLifetimeMilliseconds: 1800000
    maxPoolSize: 256
    minPoolSize: 256
    maintenanceIntervalMilliseconds: 30000
  primary_ds_1:
    url: jdbc:mysql://${host-mysql-2}:3306/sbtest?serverTimezone=UTC&useSSL=false
    username: root
    password:
    connectionTimeoutMilliseconds: 30000
    idleTimeoutMilliseconds: 60000
    maxLifetimeMilliseconds: 1800000
    maxPoolSize: 256
    minPoolSize: 256
    maintenanceIntervalMilliseconds: 30000

rules:
- !SHARDING
  tables:
    sbtest1:
      actualDataNodes: ds_${0..1}.sbtest1_${0..99}
      tableStrategy:
        standard:
          shardingColumn: id
          shardingAlgorithmName: table_inline_1
      keyGenerateStrategy:
        column: id
        keyGeneratorName: snowflake
    sbtest2:
      actualDataNodes: ds_${0..1}.sbtest2_${0..99}
      tableStrategy:
        standard:
          shardingColumn: id
          shardingAlgorithmName: table_inline_2
      keyGenerateStrategy:
        column: id
        keyGeneratorName: snowflake
    sbtest3:
      actualDataNodes: ds_${0..1}.sbtest3_${0..99}
      tableStrategy:
        standard:
          shardingColumn: id
          shardingAlgorithmName: table_inline_3
      keyGenerateStrategy:
        column: id
        keyGeneratorName: snowflake
```

```
    sbtest4:
      actualDataNodes: ds_${0..1}.sbtest4_${0..99}
      tableStrategy:
        standard:
          shardingColumn: id
          shardingAlgorithmName: table_inline_4
      keyGenerateStrategy:
        column: id
        keyGeneratorName: snowflake
    sbtest5:
      actualDataNodes: ds_${0..1}.sbtest5_${0..99}
      tableStrategy:
        standard:
          shardingColumn: id
          shardingAlgorithmName: table_inline_5
      keyGenerateStrategy:
        column: id
        keyGeneratorName: snowflake
    sbtest6:
      actualDataNodes: ds_${0..1}.sbtest6_${0..99}
      tableStrategy:
        standard:
          shardingColumn: id
          shardingAlgorithmName: table_inline_6
      keyGenerateStrategy:
        column: id
        keyGeneratorName: snowflake
    sbtest7:
      actualDataNodes: ds_${0..1}.sbtest7_${0..99}
      tableStrategy:
        standard:
          shardingColumn: id
          shardingAlgorithmName: table_inline_7
      keyGenerateStrategy:
        column: id
        keyGeneratorName: snowflake
    sbtest8:
      actualDataNodes: ds_${0..1}.sbtest8_${0..99}
      tableStrategy:
        standard:
          shardingColumn: id
          shardingAlgorithmName: table_inline_8
      keyGenerateStrategy:
        column: id
        keyGeneratorName: snowflake
    sbtest9:
      actualDataNodes: ds_${0..1}.sbtest9_${0..99}
      tableStrategy:
```

```
      standard:
        shardingColumn: id
        shardingAlgorithmName: table_inline_9
    keyGenerateStrategy:
      column: id
      keyGeneratorName: snowflake
  sbtest10:
    actualDataNodes: ds_${0..1}.sbtest10_${0..99}
    tableStrategy:
      standard:
        shardingColumn: id
        shardingAlgorithmName: table_inline_10
    keyGenerateStrategy:
      column: id
      keyGeneratorName: snowflake

defaultDatabaseStrategy:
  standard:
    shardingColumn: id
    shardingAlgorithmName: database_inline

shardingAlgorithms:
  database_inline:
    type: INLINE
    props:
      algorithm-expression: ds_${id % 2}
  table_inline_1:
    type: INLINE
    props:
      algorithm-expression: sbtest1_${id % 100}
  table_inline_2:
    type: INLINE
    props:
      algorithm-expression: sbtest2_${id % 100}
  table_inline_3:
    type: INLINE
    props:
      algorithm-expression: sbtest3_${id % 100}
  table_inline_4:
    type: INLINE
    props:
      algorithm-expression: sbtest4_${id % 100}
  table_inline_5:
    type: INLINE
    props:
      algorithm-expression: sbtest5_${id % 100}
  table_inline_6:
    type: INLINE
```

```
      props:
        algorithm-expression: sbtest6_${id % 100}
    table_inline_7:
      type: INLINE
      props:
        algorithm-expression: sbtest7_${id % 100}
    table_inline_8:
      type: INLINE
      props:
        algorithm-expression: sbtest8_${id % 100}
    table_inline_9:
      type: INLINE
      props:
        algorithm-expression: sbtest9_${id % 100}
    table_inline_10:
      type: INLINE
      props:
        algorithm-expression: sbtest10_${id % 100}
  keyGenerators:
    snowflake:
      type: SNOWFLAKE
      props:
        worker-id: 123

- !READWRITE_SPLITTING
  dataSources:
    ds_0:
      primaryDataSourceName: primary_ds_0
      replicaDataSourceNames:
        - primary_ds_0
        - primary_ds_0
    ds_1:
      name: ds_1
      primaryDataSourceName: primary_ds_1
      replicaDataSourceNames:
        - primary_ds_1
        - primary_ds_1
```

config-encrypt.yaml

```
schemaName: sbtest

dataSources:
  ds_0:
    url: jdbc:mysql://${host-mysql-1}:3306/sbtest?serverTimezone=UTC&useSSL=false
    username: root
    password:
    connectionTimeoutMilliseconds: 30000
```

```yaml
    idleTimeoutMilliseconds: 60000
    maxLifetimeMilliseconds: 1800000
    maxPoolSize: 256
    minPoolSize: 256
    maintenanceIntervalMilliseconds: 30000

rules:
- !ENCRYPT
  encryptors:
    md5_encryptor:
      type: MD5
  tables:
    sbtest1:
      columns:
        pad:
          cipherColumn: pad
          encryptorName: md5_encryptor
    sbtest2:
      columns:
        pad:
          cipherColumn: pad
          encryptorName: md5_encryptor
    sbtest3:
      columns:
        pad:
          cipherColumn: pad
          encryptorName: md5_encryptor
    sbtest4:
      columns:
        pad:
          cipherColumn: pad
          encryptorName: md5_encryptor
    sbtest5:
      columns:
        pad:
          cipherColumn: pad
          encryptorName: md5_encryptor
    sbtest6:
      columns:
        pad:
          cipherColumn: pad
          encryptorName: md5_encryptor
    sbtest7:
      columns:
        pad:
          cipherColumn: pad
          encryptorName: md5_encryptor
    sbtest8:
```

```
      columns:
        pad:
          cipherColumn: pad
          encryptorName: md5_encryptor
    sbtest9:
      columns:
        pad:
          cipherColumn: pad
          encryptorName: md5_encryptor
    sbtest10:
      columns:
        pad:
          cipherColumn: pad
          encryptorName: md5_encryptor
```

### 4.1.1 version

server.yaml

```
authentication:
  users:
    root:
      password: root
    sharding:
      password: sharding
      authorizedSchemas: sharding_db

props:
  max.connections.size.per.query: 10
  acceptor.size: 256  # The default value is available processors count * 2.
  executor.size: 128  # Infinite by default.
  proxy.frontend.flush.threshold: 128  # The default value is 128.
    # LOCAL: Proxy will run with LOCAL transaction.
    # XA: Proxy will run with XA transaction.
    # BASE: Proxy will run with B.A.S.E transaction.
  proxy.transaction.type: LOCAL
  proxy.opentracing.enabled: false
  proxy.hint.enabled: false
  query.with.cipher.column: true
  sql.show: false
  allow.range.query.with.inline.sharding: false
```

config-sharding.yaml

```
schemaName: sbtest

dataSources:
```

```
  ds_0:
    url: jdbc:mysql://${host-mysql-1}:3306/sbtest?serverTimezone=UTC&useSSL=false
    username: root
    password:
    connectionTimeoutMilliseconds: 30000
    idleTimeoutMilliseconds: 60000
    maxLifetimeMilliseconds: 1800000
    maxPoolSize: 256
  ds_1:
    url: jdbc:mysql://${host-mysql-2}:3306/sbtest?serverTimezone=UTC&useSSL=false
    username: root
    password:
    connectionTimeoutMilliseconds: 30000
    idleTimeoutMilliseconds: 60000
    maxLifetimeMilliseconds: 1800000
    maxPoolSize: 256

shardingRule:
  tables:
    sbtest1:
      actualDataNodes: ds_${0..1}.sbtest1_${0..99}
      tableStrategy:
        inline:
          shardingColumn: id
          algorithmExpression: sbtest1_${id % 100}
      keyGenerator:
        type: SNOWFLAKE
        column: id
    sbtest2:
      actualDataNodes: ds_${0..1}.sbtest2_${0..99}
      tableStrategy:
        inline:
          shardingColumn: id
          algorithmExpression: sbtest2_${id % 100}
      keyGenerator:
        type: SNOWFLAKE
        column: id
    sbtest3:
      actualDataNodes: ds_${0..1}.sbtest3_${0..99}
      tableStrategy:
        inline:
          shardingColumn: id
          algorithmExpression: sbtest3_${id % 100}
      keyGenerator:
        type: SNOWFLAKE
        column: id
    sbtest4:
      actualDataNodes: ds_${0..1}.sbtest4_${0..99}
```

```
    tableStrategy:
      inline:
        shardingColumn: id
        algorithmExpression: sbtest4_${id % 100}
    keyGenerator:
      type: SNOWFLAKE
      column: id
  sbtest5:
    actualDataNodes: ds_${0..1}.sbtest5_${0..99}
    tableStrategy:
      inline:
        shardingColumn: id
        algorithmExpression: sbtest5_${id % 100}
    keyGenerator:
      type: SNOWFLAKE
      column: id
  sbtest6:
    actualDataNodes: ds_${0..1}.sbtest6_${0..99}
    tableStrategy:
      inline:
        shardingColumn: id
        algorithmExpression: sbtest6_${id % 100}
    keyGenerator:
      type: SNOWFLAKE
      column: id
  sbtest7:
    actualDataNodes: ds_${0..1}.sbtest7_${0..99}
    tableStrategy:
      inline:
        shardingColumn: id
        algorithmExpression: sbtest7_${id % 100}
    keyGenerator:
      type: SNOWFLAKE
      column: id
  sbtest8:
    actualDataNodes: ds_${0..1}.sbtest8_${0..99}
    tableStrategy:
      inline:
        shardingColumn: id
        algorithmExpression: sbtest8_${id % 100}
    keyGenerator:
      type: SNOWFLAKE
      column: id
  sbtest9:
    actualDataNodes: ds_${0..1}.sbtest9_${0..99}
    tableStrategy:
      inline:
        shardingColumn: id
```

```
          algorithmExpression: sbtest9_${id % 100}
      keyGenerator:
        type: SNOWFLAKE
        column: id
    sbtest10:
      actualDataNodes: ds_${0..1}.sbtest10_${0..99}
      tableStrategy:
        inline:
          shardingColumn: id
          algorithmExpression: sbtest10_${id % 100}
      keyGenerator:
        type: SNOWFLAKE
        column: id

  defaultDatabaseStrategy:
    inline:
      shardingColumn: id
      algorithmExpression: ds_${id % 2}
```

config-master_slave.yaml

```
schemaName: sbtest

dataSources:
  ds_0:
    url: jdbc:mysql://${host-mysql-1}:3306/sbtest?serverTimezone=UTC&useSSL=false
    username: root
    password:
    connectionTimeoutMilliseconds: 30000
    idleTimeoutMilliseconds: 60000
    maxLifetimeMilliseconds: 1800000
    maxPoolSize: 256

masterSlaveRule:
  name: ms_ds
  masterDataSourceName: ds_0
  slaveDataSourceNames:
    - ds_0
    - ds_0
```

config-sharding-master_slave.yaml

```
schemaName: sbtest

dataSources:
  primary_ds_0:
    url: jdbc:mysql://${host-mysql-1}:3306/sbtest?serverTimezone=UTC&useSSL=false
    username: root
```

```
    password:
    connectionTimeoutMilliseconds: 30000
    idleTimeoutMilliseconds: 60000
    maxLifetimeMilliseconds: 1800000
    maxPoolSize: 256
  primary_ds_1:
    url: jdbc:mysql://${host-mysql-2}:3306/sbtest?serverTimezone=UTC&useSSL=false
    username: root
    password:
    connectionTimeoutMilliseconds: 30000
    idleTimeoutMilliseconds: 60000
    maxLifetimeMilliseconds: 1800000
    maxPoolSize: 256

shardingRule:
  tables:
    sbtest1:
      actualDataNodes: ds_${0..1}.sbtest1_${0..99}
      tableStrategy:
        inline:
          shardingColumn: id
          algorithmExpression: sbtest1_${id % 100}
      keyGenerator:
        type: SNOWFLAKE
        column: id
    sbtest2:
      actualDataNodes: ds_${0..1}.sbtest2_${0..99}
      tableStrategy:
        inline:
          shardingColumn: id
          algorithmExpression: sbtest2_${id % 100}
      keyGenerator:
        type: SNOWFLAKE
        column: id
    sbtest3:
      actualDataNodes: ds_${0..1}.sbtest3_${0..99}
      tableStrategy:
        inline:
          shardingColumn: id
          algorithmExpression: sbtest3_${id % 100}
      keyGenerator:
        type: SNOWFLAKE
        column: id
    sbtest4:
      actualDataNodes: ds_${0..1}.sbtest4_${0..99}
      tableStrategy:
        inline:
          shardingColumn: id
```

```
        algorithmExpression: sbtest4_${id % 100}
    keyGenerator:
      type: SNOWFLAKE
      column: id
  sbtest5:
    actualDataNodes: ds_${0..1}.sbtest5_${0..99}
    tableStrategy:
      inline:
        shardingColumn: id
        algorithmExpression: sbtest5_${id % 100}
    keyGenerator:
      type: SNOWFLAKE
      column: id
  sbtest6:
    actualDataNodes: ds_${0..1}.sbtest6_${0..99}
    tableStrategy:
      inline:
        shardingColumn: id
        algorithmExpression: sbtest6_${id % 100}
    keyGenerator:
      type: SNOWFLAKE
      column: id
  sbtest7:
    actualDataNodes: ds_${0..1}.sbtest7_${0..99}
    tableStrategy:
      inline:
        shardingColumn: id
        algorithmExpression: sbtest7_${id % 100}
    keyGenerator:
      type: SNOWFLAKE
      column: id
  sbtest8:
    actualDataNodes: ds_${0..1}.sbtest8_${0..99}
    tableStrategy:
      inline:
        shardingColumn: id
        algorithmExpression: sbtest8_${id % 100}
    keyGenerator:
      type: SNOWFLAKE
      column: id
  sbtest9:
    actualDataNodes: ds_${0..1}.sbtest9_${0..99}
    tableStrategy:
      inline:
        shardingColumn: id
        algorithmExpression: sbtest9_${id % 100}
    keyGenerator:
      type: SNOWFLAKE
```

```
        column: id
    sbtest10:
      actualDataNodes: ds_${0..1}.sbtest10_${0..99}
      tableStrategy:
        inline:
          shardingColumn: id
          algorithmExpression: sbtest10_${id % 100}
      keyGenerator:
        type: SNOWFLAKE
        column: id

  defaultDatabaseStrategy:
    inline:
      shardingColumn: id
      algorithmExpression: ds_${id % 2}

  masterSlaveRules:
    ds_0:
      masterDataSourceName: primary_ds_0
      slaveDataSourceNames: [primary_ds_0, primary_ds_0]
      loadBalanceAlgorithmType: ROUND_ROBIN
    ds_1:
      masterDataSourceName: primary_ds_1
      slaveDataSourceNames: [primary_ds_1, primary_ds_1]
      loadBalanceAlgorithmType: ROUND_ROBIN
```

config-encrypt.yaml

```
schemaName: sbtest

dataSources:
  ds_0:
    url: jdbc:mysql://${host-mysql-1}:3306/sbtest?serverTimezone=UTC&useSSL=false
    username: root
    password:
    connectionTimeoutMilliseconds: 30000
    idleTimeoutMilliseconds: 60000
    maxLifetimeMilliseconds: 1800000
    maxPoolSize: 256

encryptRule:
  encryptors:
    encryptor_md5:
      type: md5
  tables:
    sbtest1:
      columns:
        pad:
```

```
          cipherColumn: pad
          encryptor: encryptor_md5
    sbtest2:
      columns:
        pad:
          cipherColumn: pad
          encryptor: encryptor_md5
    sbtest3:
      columns:
        pad:
          cipherColumn: pad
          encryptor: encryptor_md5
    sbtest4:
      columns:
        pad:
          cipherColumn: pad
          encryptor: encryptor_md5
    sbtest5:
      columns:
        pad:
          cipherColumn: pad
          encryptor: encryptor_md5
    sbtest6:
      columns:
        pad:
          cipherColumn: pad
          encryptor: encryptor_md5
    sbtest7:
      columns:
        pad:
          cipherColumn: pad
          encryptor: encryptor_md5
    sbtest8:
      columns:
        pad:
          cipherColumn: pad
          encryptor: encryptor_md5
    sbtest9:
      columns:
        pad:
          cipherColumn: pad
          encryptor: encryptor_md5
    sbtest10:
      columns:
        pad:
          cipherColumn: pad
          encryptor: encryptor_md5
```

### 3.0.0 version

server.yaml

```yaml
authentication:
  username: root
  password: root

props:
  max.connections.size.per.query: 10
  acceptor.size: 256  # The default value is available processors count * 2.
  executor.size: 128  # Infinite by default.
  proxy.frontend.flush.threshold: 128  # The default value is 128.
    # LOCAL: Proxy will run with LOCAL transaction.
    # XA: Proxy will run with XA transaction.
    # BASE: Proxy will run with B.A.S.E transaction.
  proxy.transaction.type: LOCAL
  proxy.opentracing.enabled: false
  sql.show: false
```

config-sharding.yaml

```yaml
schemaName: sbtest

dataSources:
  ds_0:
    url: jdbc:mysql://${host-mysql-1}:3306/sbtest?serverTimezone=UTC&useSSL=false
    username: root
    password:
    autoCommit: true
    connectionTimeout: 30000
    idleTimeout: 60000
    maxLifetime: 1800000
    maximumPoolSize: 256
  ds_1:
    url: jdbc:mysql://${host-mysql-2}:3306/sbtest?serverTimezone=UTC&useSSL=false
    username: root
    password:
    autoCommit: true
    connectionTimeout: 30000
    idleTimeout: 60000
    maxLifetime: 1800000
    maximumPoolSize: 256

shardingRule:
  tables:
    sbtest1:
      actualDataNodes: ds_${0..1}.sbtest1_${0..99}
      tableStrategy:
```

```
      inline:
        shardingColumn: id
        algorithmExpression: sbtest1_${id % 100}
  sbtest2:
    actualDataNodes: ds_${0..1}.sbtest2_${0..99}
    tableStrategy:
      inline:
        shardingColumn: id
        algorithmExpression: sbtest2_${id % 100}
  sbtest3:
    actualDataNodes: ds_${0..1}.sbtest3_${0..99}
    tableStrategy:
      inline:
        shardingColumn: id
        algorithmExpression: sbtest3_${id % 100}
  sbtest4:
    actualDataNodes: ds_${0..1}.sbtest4_${0..99}
    tableStrategy:
      inline:
        shardingColumn: id
        algorithmExpression: sbtest4_${id % 100}
  sbtest5:
    actualDataNodes: ds_${0..1}.sbtest5_${0..99}
    tableStrategy:
      inline:
        shardingColumn: id
        algorithmExpression: sbtest5_${id % 100}
  sbtest6:
    actualDataNodes: ds_${0..1}.sbtest6_${0..99}
    tableStrategy:
      inline:
        shardingColumn: id
        algorithmExpression: sbtest6_${id % 100}
  sbtest7:
    actualDataNodes: ds_${0..1}.sbtest7_${0..99}
    tableStrategy:
      inline:
        shardingColumn: id
        algorithmExpression: sbtest7_${id % 100}
  sbtest8:
    actualDataNodes: ds_${0..1}.sbtest8_${0..99}
    tableStrategy:
      inline:
        shardingColumn: id
        algorithmExpression: sbtest8_${id % 100}
  sbtest9:
    actualDataNodes: ds_${0..1}.sbtest9_${0..99}
    tableStrategy:
```

```
        inline:
          shardingColumn: id
          algorithmExpression: sbtest9_${id % 100}
    sbtest10:
      actualDataNodes: ds_${0..1}.sbtest10_${0..99}
      tableStrategy:
        inline:
          shardingColumn: id
          algorithmExpression: sbtest10_${id % 100}

  defaultDatabaseStrategy:
    inline:
      shardingColumn: id
      algorithmExpression: ds_${id % 2}
```

config-master_slave.yaml

```
schemaName: sbtest

dataSources:
  ds_0:
    url: jdbc:mysql://${host-mysql-1}:3306/sbtest?serverTimezone=UTC&useSSL=false
    username: root
    password:
    autoCommit: true
    connectionTimeout: 30000
    idleTimeout: 60000
    maxLifetime: 1800000
    maximumPoolSize: 256

masterSlaveRule:
  name: ms_ds
  masterDataSourceName: ds_0
  slaveDataSourceNames:
    - ds_0
    - ds_0
```

config-sharding-master_slave.yaml

```
schemaName: sbtest

dataSources:
  primary_ds_0:
    url: jdbc:mysql://${host-mysql-1}:3306/sbtest?serverTimezone=UTC&useSSL=false
    username: root
    password:
    autoCommit: true
    connectionTimeout: 30000
```

```
    idleTimeout: 60000
    maxLifetime: 1800000
    maximumPoolSize: 256
  primary_ds_1:
    url: jdbc:mysql://${host-mysql-2}:3306/sbtest?serverTimezone=UTC&useSSL=false
    username: root
    password:
    autoCommit: true
    connectionTimeout: 30000
    idleTimeout: 60000
    maxLifetime: 1800000
    maximumPoolSize: 256

shardingRule:
  tables:
    sbtest1:
      actualDataNodes: ds_${0..1}.sbtest1_${0..99}
      tableStrategy:
        inline:
          shardingColumn: id
          algorithmExpression: sbtest1_${id % 100}
    sbtest2:
      actualDataNodes: ds_${0..1}.sbtest2_${0..99}
      tableStrategy:
        inline:
          shardingColumn: id
          algorithmExpression: sbtest2_${id % 100}
    sbtest3:
      actualDataNodes: ds_${0..1}.sbtest3_${0..99}
      tableStrategy:
        inline:
          shardingColumn: id
          algorithmExpression: sbtest3_${id % 100}
    sbtest4:
      actualDataNodes: ds_${0..1}.sbtest4_${0..99}
      tableStrategy:
        inline:
          shardingColumn: id
          algorithmExpression: sbtest4_${id % 100}
    sbtest5:
      actualDataNodes: ds_${0..1}.sbtest5_${0..99}
      tableStrategy:
        inline:
          shardingColumn: id
          algorithmExpression: sbtest5_${id % 100}
    sbtest6:
      actualDataNodes: ds_${0..1}.sbtest6_${0..99}
      tableStrategy:
```

```
      inline:
        shardingColumn: id
        algorithmExpression: sbtest6_${id % 100}
  sbtest7:
    actualDataNodes: ds_${0..1}.sbtest7_${0..99}
    tableStrategy:
      inline:
        shardingColumn: id
        algorithmExpression: sbtest7_${id % 100}
  sbtest8:
    actualDataNodes: ds_${0..1}.sbtest8_${0..99}
    tableStrategy:
      inline:
        shardingColumn: id
        algorithmExpression: sbtest8_${id % 100}
  sbtest9:
    actualDataNodes: ds_${0..1}.sbtest9_${0..99}
    tableStrategy:
      inline:
        shardingColumn: id
        algorithmExpression: sbtest9_${id % 100}
  sbtest10:
    actualDataNodes: ds_${0..1}.sbtest10_${0..99}
    tableStrategy:
      inline:
        shardingColumn: id
        algorithmExpression: sbtest10_${id % 100}

defaultDatabaseStrategy:
  inline:
    shardingColumn: id
    algorithmExpression: ds_${id % 2}

masterSlaveRules:
  ds_0:
    masterDataSourceName: primary_ds_0
    slaveDataSourceNames: [primary_ds_0, primary_ds_0]
    loadBalanceAlgorithmType: ROUND_ROBIN
  ds_1:
    masterDataSourceName: primary_ds_1
    slaveDataSourceNames: [primary_ds_1, primary_ds_1]
    loadBalanceAlgorithmType: ROUND_ROBIN
```

config-encrypt.yaml

```
Unsupported
```

### Appendix 2

plot_graph.py

```
import sys
import matplotlib.pyplot as plt
import numpy as np


def generate_graph(path, case_name):
    dataset = {
        'build_num': [],
        'master_version': [],
        'master_xa': [],
        '4.1.1_version': [],
        '3.0.0_version': [],
        'mysql_server': []
    }
    with open(path + '/.build_number.txt') as builds:
        for line in builds:
            dataset['build_num'].append(int(line))
    generate_data(path, case_name, dataset)
    print(dataset)
    fig, ax = plt.subplots()
    ax.grid(True)
    plt.title(case_name)

    data = [dataset['master_version'][-7:], dataset['master_xa'][-7:], dataset['4.
1.1_version'][-7:], dataset['3.0.0_version'][-7:], dataset['mysql_server'][-7:]]
    columns = dataset['build_num'][-7:]
    rows = ['master', 'xa', '4.1.1', '3.0.0', 'mysql']
    rcolors = plt.cm.BuPu(np.full(len(rows), 0.1))
    ccolors = plt.cm.BuPu(np.full(len(columns), 0.1))
    the_table = plt.table(cellText=data, rowLabels=rows, colLabels=columns,
rowColours=rcolors, colColours=ccolors,
                          loc='bottom', bbox=[0.0, -0.50, 1, .28])
    plt.subplots_adjust(left=0.15, bottom=0.3, right=0.98)

    plt.xticks(range(14))
    ax.set_xticklabels(dataset['build_num'])
    plt.plot(dataset['master_version'], 'o-', color='magenta', label='master_
version')
    plt.plot(dataset['master_xa'], 'o-', color='darkviolet', label='master_xa')
    plt.plot(dataset['4.1.1_version'], 'r--', color='blue', label='4.1.1_version')
    plt.plot(dataset['3.0.0_version'], 'r--', color='orange', label='3.0.0_version
')
    plt.plot(dataset['mysql_server'], 'r--', color='lime', label='mysql_server')
    plt.xlim()
    plt.legend()
```

```
    plt.xlabel('build_num')
    plt.ylabel('transactions per second')
    plt.savefig('graph/' + path + '/' + case_name)
    plt.show()


def generate_data(path, case_name, dataset):
    for build in dataset['build_num']:
        fill_dataset(build, case_name, dataset, path, 'master_version', '.master.
txt')
        fill_dataset(build, case_name, dataset, path, 'master_xa', '.xa.txt')
        fill_dataset(build, case_name, dataset, path, '4.1.1_version', '.4_1_1.txt
')
        fill_dataset(build, case_name, dataset, path, '3.0.0_version', '.3_0_0.txt
')
        fill_dataset(build, case_name, dataset, path, 'mysql_server', '.mysql.txt')


def fill_dataset(build, case_name, dataset, path, version, suffix):
    try:
        with open(path + '/' + str(build) + '/' + case_name + suffix) as version_
master:
            value = 0
            for line in version_master:
                if 'transactions:' in line:
                    items = line.split('(')
                    value = float(items[1][:-10])
            dataset[version].append(value)
    except FileNotFoundError:
        dataset[version].append(0)


if __name__ == '__main__':
    path = sys.argv[1]
    generate_graph(path, 'oltp_point_select')
    generate_graph(path, 'oltp_read_only')
    generate_graph(path, 'oltp_write_only')
    generate_graph(path, 'oltp_read_write')
    generate_graph(path, 'oltp_update_index')
    generate_graph(path, 'oltp_update_non_index')
    generate_graph(path, 'oltp_delete')
```

# 4 User Manual

This chapter describes how to use projects of Apache ShardingSphere: ShardingSphere-JDBC, ShardingSphere-Proxy and ShardingSphere-Sidecar. Except main projects, this chapter also describe how to use derivative projects of Apache ShardingSphere: ShardingSphere-Scaling and ShardingSphere-UI.

## 4.1 ShardingSphere-JDBC

### 4.1.1 Introduction

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, Druid, HikariCP.
- Support any kind of JDBC standard database: MySQL, Oracle, SQLServer, PostgreSQL and any SQL92 followed databases.

### 4.1.2 Comparison

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

ShardingSphere-JDBC is suitable for java application.

### 4.1.3 Usage

This chapter will introduce the use of ShardingSphere-JDBC. Please refer to Example for more details.

#### Data Sharding

Data sharding is the basic capability of Apache ShardingSphere. This section uses data sharding as an example. The usage of functions such as replica query, data encryption, shadow database is completely consistent with data sharding, as long as the corresponding rules are configured. Multiple rules can be appended.

Please refer to Configuration Manual for more details.

#### Use Java API

#### Import Maven Dependency

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

### Configure Rule

ShardingSphere-JDBC Java API consists of data sources, rules and properties configuration. The following example is the configuration of 2 databases and 2 tables, whose databases take module and split according to `order_id`, tables take module and split according to `order_id` .

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

```
// Configure actual data sources
Map<String, DataSource> dataSourceMap = new HashMap<>();

// Configure the first data source
HikariDataSource dataSource1 = new HikariDataSource();
dataSource1.setDriverClassName("com.mysql.jdbc.Driver");
dataSource1.setJdbcUrl("jdbc:mysql://localhost:3306/ds0");
dataSource1.setUsername("root");
dataSource1.setPassword("");
dataSourceMap.put("ds0", dataSource1);

// Configure the second data source
HikariDataSource dataSource2 = new HikariDataSource();
dataSource2.setDriverClassName("com.mysql.jdbc.Driver");
dataSource2.setJdbcUrl("jdbc:mysql://localhost:3306/ds1");
dataSource2.setUsername("root");
dataSource2.setPassword("");
dataSourceMap.put("ds1", dataSource2);

// Configure order table rule
ShardingTableRuleConfiguration orderTableRuleConfig = new
ShardingTableRuleConfiguration("t_order", "ds${0..1}.t_order${0..1}");

// Configure database sharding strategy
orderTableRuleConfig.setDatabaseShardingStrategy(new
StandardShardingStrategyConfiguration("user_id", "dbShardingAlgorithm"));

// Configure table sharding strategy
orderTableRuleConfig.setTableShardingStrategy(new
StandardShardingStrategyConfiguration("order_id", "tableShardingAlgorithm"));

// Omit t_order_item table rule configuration ...
// ...

// Configure sharding rule
ShardingRuleConfiguration shardingRuleConfig = new ShardingRuleConfiguration();
shardingRuleConfig.getTables().add(orderTableRuleConfig);

// Configure database sharding algorithm
Properties dbShardingAlgorithmrProps = new Properties();
```

```
dbShardingAlgorithmrProps.setProperty("algorithm-expression", "ds${user_id % 2}");
shardingRuleConfig.getShardingAlgorithms().put("dbShardingAlgorithm", new
ShardingSphereAlgorithmConfiguration("INLINE", dbShardingAlgorithmrProps));

// Configure table sharding algorithm
Properties tableShardingAlgorithmrProps = new Properties();
tableShardingAlgorithmrProps.setProperty("algorithm-expression", "t_order${order_id
% 2}");
shardingRuleConfig.getShardingAlgorithms().put("tableShardingAlgorithm", new
ShardingSphereAlgorithmConfiguration("INLINE", tableShardingAlgorithmrProps));

// Create ShardingSphereDataSource
DataSource dataSource = ShardingSphereDataSourceFactory.
createDataSource(dataSourceMap, Collections.singleton(shardingRuleConfig), new
Properties());
```

#### Use ShardingSphereDataSource

The ShardingSphereDataSource created by ShardingSphereDataSourceFactory implements the standard JDBC DataSource interface. Developer can choose to use native JDBC or ORM frameworks such as JPA or MyBatis through the DataSource.

Take native JDBC usage as an example:

```
DataSource dataSource = ShardingSphereDataSourceFactory.
createDataSource(dataSourceMap, Collections.singleton(shardingRuleConfig), new
Properties());
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 = ps.executeQuery()) {
        while(rs.next()) {
            // ...
        }
    }
}
```

### Use YAML

#### Import Maven Dependency

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

#### Configure Rule

ShardingSphere-JDBC YAML file consists of data sources, rules and properties configuration. The following example is the configuration of 2 databases and 2 tables, whose databases take module and split according to `order_id`, tables take module and split according to `order_id`.

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

```yaml
# Configure actual data sources
dataSources:
  # Configure the first data source
  ds0: !!com.zaxxer.hikari.HikariDataSource
    driverClassName: com.mysql.jdbc.Driver
    jdbcUrl: jdbc:mysql://localhost:3306/ds0
    username: root
    password:
  # Configure the second data source
  ds1: !!com.zaxxer.hikari.HikariDataSource
    driverClassName: com.mysql.jdbc.Driver
    jdbcUrl: jdbc:mysql://localhost:3306/ds1
    username: root
    password:

rules:
# Configure sharding rule
- !SHARDING
  tables:
    # Configure t_order table rule
    t_order:
      actualDataNodes: ds${0..1}.t_order${0..1}
      # Configure database sharding strategy
      databaseStrategy:
        standard:
          shardingColumn: user_id
          shardingAlgorithmName: database_inline
      # Configure table sharding strategy
```

```yaml
      tableStrategy:
        standard:
          shardingColumn: order_id
          shardingAlgorithmName: table_inline
    t_order_item:
    # Omit t_order_item table rule configuration ...
    # ...

  # Configure sharding algorithms
  shardingAlgorithms:
    database_inline:
      type: INLINE
      props:
        algorithm-expression: ds${user_id % 2}
    table_inline:
      type: INLINE
      props:
        algorithm-expression: t_order_${order_id % 2}
```

```java
// Create ShardingSphereDataSource
DataSource dataSource = YamlShardingSphereDataSourceFactory.
createDataSource(yamlFile);
```

#### Use ShardingSphereDataSource

The ShardingSphereDataSource created by YamlShardingSphereDataSourceFactory implements the standard JDBC DataSource interface. Developer can choose to use native JDBC or ORM frameworks such as JPA or MyBatis through the DataSource.

Take native JDBC usage as an example:

```java
DataSource dataSource = YamlShardingSphereDataSourceFactory.
createDataSource(yamlFile);
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()) {
            // ...
        }
    }
}
```

### Use Spring Boot Starter

#### Import Maven Dependency

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

#### Configure Rule

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

```
# Configure actual data sources
spring.shardingsphere.datasource.names=ds0,ds1

# Configure the first data source
spring.shardingsphere.datasource.ds0.type=com.zaxxer.hikari.HikariDataSource
spring.shardingsphere.datasource.ds0.driver-class-name=com.mysql.jdbc.Driver
spring.shardingsphere.datasource.ds0.jdbc-url=jdbc:mysql://localhost:3306/ds0
spring.shardingsphere.datasource.ds0.username=root
spring.shardingsphere.datasource.ds0.password=

# Configure the second 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 t_order table rule
spring.shardingsphere.rules.sharding.tables.t_order.actual-data-nodes=ds$->{0..1}.
t_order$->{0..1}

# Configure database sharding strategy
spring.shardingsphere.rules.sharding.tables.t_order.database-strategy.standard.
sharding-column=user_id
spring.shardingsphere.rules.sharding.tables.t_order.database-strategy.standard.
sharding-algorithm-name=database_inline

# Configure table sharding strategy
spring.shardingsphere.rules.sharding.tables.t_order.table-strategy.standard.
sharding-column=order_id
spring.shardingsphere.rules.sharding.tables.t_order.table-strategy.standard.
sharding-algorithm-name=table_inline
```

```
# Omit t_order_item table rule configuration ...
# ...

# Configure sharding algorithm
spring.shardingsphere.rules.sharding.sharding-algorithms.database_inline.
type=INLINE
spring.shardingsphere.rules.sharding.sharding-algorithms.database_inline.props.
algorithm-expression=ds_${user_id % 2}
spring.shardingsphere.rules.sharding.sharding-algorithms.table_inline.type=INLINE
spring.shardingsphere.rules.sharding.sharding-algorithms.table_inline.props.
algorithm-expression=t_order_${order_id % 2}
```

#### 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 datasource. For example:

```
# Configure actual data sources
spring.shardingsphere.datasource.names=ds0,ds1

# Configure the first data source
spring.shardingsphere.datasource.ds0.jndi-name=java:comp/env/jdbc/ds0
# Configure the second data source
spring.shardingsphere.datasource.ds1.jndi-name=java:comp/env/jdbc/ds1

# Omit rule configurations ...
# ...
```

#### Use ShardingSphereDataSource in Spring

ShardingSphereDataSource can be used directly by injection; or configure ShardingSphereDataSource in ORM frameworks such as JPA or MyBatis.

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

### Use Spring Namespace

#### Import Maven Dependency

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

#### Configure Rule

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

```
<?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.apache.org/schema/shardingsphere/sharding
"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
                        http://www.springframework.org/schema/beans/spring-beans.
xsd
                        http://shardingsphere.apache.org/schema/shardingsphere/
sharding
                        http://shardingsphere.apache.org/schema/shardingsphere/
sharding/sharding.xsd
                        ">
    <!-- Configure actual data sources -->
    <!-- Configure the first data source -->
    <bean id="ds0" class="com.zaxxer.hikari.HikariDataSource" destroy-method="close
">
        <property name="driverClassName" value="com.mysql.jdbc.Driver" />
        <property name="jdbcUrl" value="jdbc:mysql://localhost:3306/ds0" />
        <property name="username" value="root" />
        <property name="password" value="" />
    </bean>
    <!-- Configure the second data source -->
    <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>

    <!-- Configure database sharding strategy -->
```

```xml
    <sharding:sharding-algorithm id="dbShardingAlgorithm" type="INLINE">
            <props>
                <prop key="algorithm-expression">ds$->{user_id % 2}</prop>
            </props>
    </sharding:sharding-algorithm>
    <sharding:standard-strategy id="dbStrategy" sharding-column="user_id"
algorithm-ref="dbShardingAlgorithm" />

    <!-- Configure table sharding strategy -->
    <sharding:sharding-algorithm id="tableShardingAlgorithm" type="INLINE">
            <props>
                <prop key="algorithm-expression">t_order$->{order_id % 2}</prop>
            </props>
    </sharding:sharding-algorithm>
    <sharding:standard-strategy id="tableStrategy" sharding-column="user_id"
algorithm-ref="tableShardingAlgorithm" />

    <!-- Configure distributed key-generate strategy -->
    <sharding:key-generate-algorithm id="snowflakeAlgorithm" type="SNOWFLAKE">
              <props>
                  <prop key="worker-id">123</prop>
              </props>
    </sharding:key-generate-algorithm>
    <sharding:key-generate-strategy id="orderKeyGenerator" column="order_id"
algorithm-ref="snowflakeAlgorithm" />

    <!-- Configure sharding rule -->
    <sharding:rule id="shardingRule">
            <sharding:table-rules>
                <sharding:table-rule logic-table="t_order" actual-data-nodes="ds${0.
.1}.t_order_${0..1}" database-strategy-ref="dbStrategy" table-strategy-ref=
"tableStrategy" key-generate-strategy-ref="orderKeyGenerator" />
            </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>

    <!-- Configure ShardingSphereDataSource -->
    <shardingsphere:data-source id="shardingDataSource" data-source-names="ds0,ds1"
rule-refs="shardingRule">
            <props>
                <prop key="sql-show">false</prop>
            </props>
    </shardingsphere:data-source>
```

```
</beans>
```

#### Use ShardingSphereDataSource in Spring

ShardingSphereDataSource can be used directly by injection; or configure ShardingSphereDataSource in ORM frameworks such as JPA or MyBatis.

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

### 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:

```
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. After that, SQL parse and rewrite phase will be skipped and the overall enforcement efficiency can be enhanced.

#### Clean Hint Values

Sharding values are saved in `ThreadLocal`, so it is necessary to use `hintManager.close()` to clean `ThreadLocal`.

**``HintManager`` has implemented ``AutoCloseable``. We recommend to close it automatically with ``try with resource``.**

#### Codes:

```
// Sharding database and table with HintManager
String sql = "SELECT * FROM t_order";
try (HintManager hintManager = HintManager.getInstance();
     Connection conn = dataSource.getConnection();
     PreparedStatement preparedStatement = conn.prepareStatement(sql)) {
    hintManager.addDatabaseShardingValue("t_order", 1);
    hintManager.addTableShardingValue("t_order", 2);
    try (ResultSet rs = preparedStatement.executeQuery()) {
        while (rs.next()) {
            // ...
```

```
        }
    }
}

// Sharding database and one database route with HintManager
String sql = "SELECT * FROM t_order";
try (HintManager hintManager = HintManager.getInstance();
     Connection conn = dataSource.getConnection();
     PreparedStatement preparedStatement = conn.prepareStatement(sql)) {
    hintManager.setDatabaseShardingValue(3);
    try (ResultSet rs = preparedStatement.executeQuery()) {
        while (rs.next()) {
            // ...
        }
    }
}
```

### Primary Route with Hint

#### Get HintManager

Be the same as sharding based on hint.

#### Configure Primary Database Route

- Use `hintManager.setPrimaryRouteOnly` to configure primary database route.

#### Clean Hint Value

Be the same as data sharding based on hint.

#### Codes:

```
String sql = "SELECT * FROM t_order";
try (
        HintManager hintManager = HintManager.getInstance();
        Connection conn = dataSource.getConnection();
        PreparedStatement preparedStatement = conn.prepareStatement(sql)) {
    hintManager.setPrimaryRouteOnly();
    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

```
<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

```
@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
@ShardingTransactionType(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 @ShardingTransactionType annotation to inject the transaction
type before connection created -->
<sharding:tx-type-annotation-driven />
```

#### Use Distributed Transaction

```
@Transactional
@ShardingTransactionType(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

```
<propeties>
    <btm.version>2.1.3</btm.version>
</propeties>

<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:

```
props:
  xa-transaction-manager-type: Bitronix
```

SpringBoot:

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

Spring Namespace:

```
<shardingsphere:data-source id="xxx" data-source-names="xxx" rule-refs="xxx">
    <props>
        <prop key="xa-transaction-manager-type">Bitronix</prop>
    </props>
</shardingsphere:data-source>
```

### Narayana Transaction

#### Import Maven Dependency

```
<propeties>
    <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>
</propeties>

<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:

```
props:
  xa-transaction-manager-type: Narayana
```

SpringBoot:

```
spring:
  shardingsphere:
```

```
    props:
      xa-transaction-manager-type: Narayana
```

Spring Namespace:

```
<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).

```
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.

#### Governance

Using governance requires designating a registry center in which all the configurations are saved. Users can either use local configurations to cover config center configurations or read configurations from config center.

#### Use Java API

#### Import Maven Dependency

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

<!-- import if using ZooKeeper -->
<dependency>
    <groupId>org.apache.shardingsphere</groupId>
    <artifactId>shardingsphere-governance-repository-zookeeper-curator</artifactId>
    <version>${shardingsphere.version}</version>
</dependency>

<!-- import if using Etcd -->
<dependency>
    <groupId>org.apache.shardingsphere</groupId>
    <artifactId>shardingsphere-governance-repository-etcd</artifactId>
    <version>${shardingsphere.version}</version>
</dependency>
```

#### Configure Rule

Using ZooKeeper as config center and registry center for example.

```
// Omit configure data sources and rule configurations
// ...

// Configure registry center
RegistryCenterConfiguration registryCenterConfig = new RegistryCenterConfiguration(
"Zookeeper", "localhost:2181", new Properties());

// Configure governance
        GovernanceConfiguration governanceConfiguration = new
GovernanceConfiguration("governance-sharding-data-source", registryCenterConfig,
true);

// Create GovernanceShardingSphereDataSource
DataSource dataSource = GovernanceShardingSphereDataSourceFactory.
createDataSource(governanceConfiguration);
```

#### Use GovernanceShardingSphereDataSource

The GovernanceShardingSphereDataSource created by GovernanceShardingSphereDataSourceFactory implements the standard JDBC DataSource interface. Developer can choose to use native JDBC or ORM frameworks such as JPA or MyBatis through the DataSource.

Take native JDBC usage as an example:

```
DataSource dataSource = GovernanceShardingSphereDataSourceFactory.
createDataSource(governanceConfiguration);
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()) {
            // ...
        }
    }
}
```

### Use YAML

#### Import Maven Dependency

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

<!-- import if using ZooKeeper -->
<dependency>
    <groupId>org.apache.shardingsphere</groupId>
    <artifactId>shardingsphere-governance-repository-zookeeper-curator</artifactId>
    <version>${shardingsphere.version}</version>
</dependency>

<!-- import if using Etcd -->
<dependency>
    <groupId>org.apache.shardingsphere</groupId>
    <artifactId>shardingsphere-governance-repository-etcd</artifactId>
    <version>${shardingsphere.version}</version>
</dependency>
```

#### Configure Rule

Using ZooKeeper as config center and registry center for example.

```yaml
governance:
  name: governance_ds
  registryCenter:
      type: Zookeeper
      serverLists: localhost:2181
  overwrite: true
```

```java
// Create GovernanceShardingSphereDataSource
DataSource dataSource = YamlGovernanceShardingSphereDataSourceFactory.
createDataSource(yamlFile);
```

#### Use GovernanceShardingSphereDataSource

The GovernanceShardingSphereDataSource created by YamlGovernanceShardingSphereDataSource-Factory implements the standard JDBC DataSource interface. Developer can choose to use native JDBC or ORM frameworks such as JPA or MyBatis through the DataSource.

Take native JDBC usage as an example:

```
DataSource dataSource = YamlGovernanceShardingSphereDataSourceFactory.
createDataSource(yamlFile);
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()) {
            // ...
        }
    }
}
```

#### Use Spring Boot Starter

#### Import Maven Dependency

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

<!-- import if using ZooKeeper -->
<dependency>
    <groupId>org.apache.shardingsphere</groupId>
    <artifactId>shardingsphere-governance-repository-zookeeper-curator</artifactId>
    <version>${shardingsphere.version}</version>
</dependency>

<!-- import if using Etcd -->
<dependency>
    <groupId>org.apache.shardingsphere</groupId>
    <artifactId>shardingsphere-governance-repository-etcd</artifactId>
    <version>${shardingsphere.version}</version>
</dependency>
```

#### Configure Rule

```
spring.shardingsphere.governance.name=governance-spring-boot-shardingsphere-test
spring.shardingsphere.governance.registry-center.type=Zookeeper
spring.shardingsphere.governance.registry-center.server-lists=localhost:2181
spring.shardingsphere.governance.overwrite=true
```

#### Use GovernanceShardingSphereDataSource in Spring

GovernanceShardingSphereDataSource can be used directly by injection; or configure Governance-ShardingSphereDataSource in ORM frameworks such as JPA or MyBatis.

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

#### Use Spring Namespace

#### Import Maven Dependency

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

<!-- import if using ZooKeeper -->
<dependency>
    <groupId>org.apache.shardingsphere</groupId>
    <artifactId>shardingsphere-governance-repository-zookeeper-curator</artifactId>
    <version>${shardingsphere.version}</version>
</dependency>

<!-- import if using Etcd -->
<dependency>
    <groupId>org.apache.shardingsphere</groupId>
    <artifactId>shardingsphere-governance-repository-etcd</artifactId>
    <version>${shardingsphere.version}</version>
</dependency>
```

#### Configure Rule

```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:governance="http://shardingsphere.apache.org/schema/shardingsphere/governance"
       xmlns="http://www.springframework.org/schema/beans"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
                           http://www.springframework.org/schema/beans/spring-beans.xsd
                           http://shardingsphere.apache.org/schema/shardingsphere/governance
                           http://shardingsphere.apache.org/schema/shardingsphere/governance/governance.xsd">
     <util:properties id="instance-properties">
         <prop key="max-retries">3</prop>
         <prop key="operation-timeout-milliseconds">3000</prop>
     </util:properties>
     <governance:reg-center id="regCenter" type="Zookeeper" server-lists="localhost:2181" />
     <governance:data-source id="shardingDatabasesTablesDataSource" data-source-names="demo_ds_0, demo_ds_1" reg-center-ref="regCenter" config-center-ref="configCenter" rule-refs="shardingRule" overwrite="true" />
     <governance:data-source id="encryptDataSource" data-source-names="demo_ds" reg-center-ref="regCenter" config-center-ref="configCenter" rule-refs="encryptRule" overwrite="true" />
</beans>
```

#### Use GovernanceShardingSphereDataSource in Spring

GovernanceShardingSphereDataSource can be used directly by injection; or configure Governance-ShardingSphereDataSource in ORM frameworks such as JPA or MyBatis.

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

### 4.1.4 Configuration Manual

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, replica query, 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.

### Java API

#### Introduction

Java API is the foundation of all 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

#### Create Simple DataSource

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

```
// Build data source map
Map<String, DataSource> dataSourceMap = // ...

// Build rule configurations
Collection<RuleConfiguration> configurations = // ...

// Build properties
Properties props = // ...

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

#### Create Governance DataSource

The GovernanceShardingSphereDataSource created by GovernanceShardingSphereDataSourceFactory implements the standard JDBC DataSource interface.

```
// Build data source map
Map<String, DataSource> dataSourceMap = // ...

// Build rule configurations
Collection<RuleConfiguration> configurations = // ...

// Build properties
Properties props = // ...

// Build governance configuration
GovernanceConfiguration governanceConfig = // ...

DataSource dataSource = GovernanceShardingSphereDataSourceFactory.
createDataSource(dataSourceMap, configurations, props, governanceConfig);
```

#### Use DataSource

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

Take native JDBC usage as an example:

```
DataSource dataSource = // Use Apache ShardingSphere factory to create DataSource
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()) {
            // ...
        }
    }
}
```

### 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 <EncryptColumn-RuleConfiguration> | 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.RegistryCenterConfiguration

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 |

### 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);
```

### Change History

#### 5.0.0-alpha

##### Replica Query

##### Root Configuration

Class name: ReplicaQueryRuleConfiguration

Attributes:

| • Name* | DataType | Description |
|---|---|---|
| d ataSo urces (+) | Col lection<ReplicaQueryData SourceRuleConfiguration> | Data sources of primary and replicas |
| loa dBala ncers (*) | Map<String, ShardingSpher eAlgorithmConfiguration> | Load balance algorithm name and configurations of replica data sources |

##### Replica Query Data Source Configuration

Class name: ReplicaQueryDataSourceRuleConfiguration

Attributes:

| Name | D ataType | Description | Default Value |
|---|---|---|---|
| name | String | Replica query data source name | • |
| primaryDat aSource-Name | String | Primary sources source name | • |
| replicaData Source-Names (+) | Co llection <String> | Replica sources source name list | • |
| loadB alancerName (?) | String | Load balance algorithm name of replica sources | Round robin load balance algorithm |

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

### ShardingSphere-4.x

#### 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 |

### ShardingSphere-3.x

#### 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 |

#### 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.

###### Support item

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 item

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.

##### Support item

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 item

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);

// Continue to create ShardingDataSource through ShardingSlaveDataSourceFactory
```

### YAML Configuration

#### Introduction

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

#### Create Simple DataSource

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

```
// Indicate YAML file path
File yamlFile = // ...

DataSource dataSource = YamlShardingSphereDataSourceFactory.
createDataSource(yamlFile);
```

#### Create Governance DataSource

The GovernanceShardingSphereDataSource created by YamlGovernanceShardingSphereDataSourceFactory implements the standard JDBC DataSource interface.

```
// Indicate YAML file path
File yamlFile = // ...

DataSource dataSource = YamlGovernanceShardingSphereDataSourceFactory.
createDataSource(yamlFile);
```

#### Use DataSource

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

Take native JDBC usage as an example:

```
DataSource dataSource = // Use Apache ShardingSphere factory to create DataSource
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()) {
            // ...
        }
```

```
    }
}
```

### YAML Configuration Item

#### Data Source Configuration

It is divided into single data source configuration and multi data source configuration. Single data source configuration used for data encryption rules; and multi data source configuration used for fragmentation, replica query and other rules. If features such as encryption and sharding are used in combination, a multi data source configuration should be used.

#### Single Data Source Configuration

#### Configuration Example

```yaml
dataSource: !!org.apache.commons.dbcp2.BasicDataSource
  driverClassName: com.mysql.jdbc.Driver
  url: jdbc:mysql://127.0.0.1:3306/ds_name
  username: root
  password: root
```

#### Configuration Item Explanation

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

#### Multi Data Source Configuration

#### Configuration Example

```yaml
dataSources:
  ds_0: !!org.apache.commons.dbcp2.BasicDataSource
    driverClassName: com.mysql.jdbc.Driver
    url: jdbc:mysql://127.0.0.1:3306/ds_0
    username: sa
    password:
```

```
  ds_1: !!org.apache.commons.dbcp2.BasicDataSource
    driverClassName: com.mysql.jdbc.Driver
    url: jdbc:mysql://127.0.0.1:3306/ds_1
    username: sa
    password:
```

#### 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
    url: # Database URL
    username: # Database username
    password: # Database password
    # ... Other properties for data source pool
```

#### Rule Configuration

Begin to configure with the rule alias to configure multiple rules.

#### Configuration Example

```
rules:
-! XXX_RULE_0
  xxx
-! XXX_RULE_1
  xxx
```

#### Configuration Item Explanation

```
rules:
-! XXX_RULE # Rule alias, `-` means can configure multi rules
  # ... Specific rule configurations
```

Please refer to specific rule configuration for more details.

#### Properties Configuration

#### Configuration Example

```
props:
  xxx: xxx
```

#### Configuration Item Explanation

```
props:
  xxx: xxx # Properties key and value
```

Please refer to specific rule configuration for more details.

#### 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

#### 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:
  # ...
```

#### Readwrite-splitting

##### Configuration Item Explanation

```yaml
dataSource: # Omit the data source configuration, please refer to the usage

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
        # ...

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

##### Management

```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
```

### 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:
  - !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
```

#### Change History

##### 5.0.0-alpha

##### Replica Query

##### Configuration Item Explanation

```
dataSource: # 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
        # ...
```

```yaml
props:
  # ...
```

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

#### ShardingSphere-4.x

#### Readwrite-splitting

#### Configuration Item Explanation

```yaml
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:

```java
DataSource dataSource = YamlMasterSlaveDataSourceFactory.
createDataSource(yamlFile);
```

### ShardingSphere-3.x

#### Readwrite-splitting

##### 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 `MasterSlaveDataSourceFactory` factory class:

```
DataSource dataSource = MasterSlaveDataSourceFactory.createDataSource(yamlFile);
```

### 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.

#### Support item

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 item

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.

#### Configuration Item Explanation

```
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);
```

### Spring Boot Starter Configuration

#### Introduction

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

#### Data Source Configuration

```
spring.shardingsphere.datasource.names= # Data source name, multiple data sources are separated by commas
spring.shardingsphere.datasource.common.type= # Database connection pool type name
spring.shardingsphere.datasource.common.driver-class-name= # Database driver class name

spring.shardingsphere.datasource.<datasource-name>.url= # Database URL connection
spring.shardingsphere.datasource.<datasource-name>.username= # Database username
spring.shardingsphere.datasource.<datasource-name>.password= # Database password
spring.shardingsphere.datasource.<datasource-name>.xxx= # Other properties of database connection pool
```

#### Rule Configuration

```
spring.shardingsphere.rules.<rule-type>.xxx= # rule configurations
  # ... Specific rule configurations
```

Please refer to specific rule configuration for more details.

#### Properties Configuration

```
spring.shardingsphere.props.xxx.xxx= # Properties key and value
```

Please refer to Properties Configuration for more details about type of algorithm.

### 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 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.

### 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

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.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
```

#### Change History

#### 5.0.0-alpha

#### Replica Query

#### Configuration Item Explanation

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

spring.shardingsphere.rules.replica-query.data-sources.<replica-query-data-source-name>.primary-data-source-name= # Primary data source name
spring.shardingsphere.rules.replica-query.data-sources.<replica-query-data-source-name>.replica-data-source-names= # Replica data source names, multiple data source names separated with comma
spring.shardingsphere.rules.replica-query.data-sources.<replica-query-data-source-name>.load-balancer-name= # Load balance algorithm name

# Load balance algorithm configuration
spring.shardingsphere.rules.replica-query.load-balancers.<load-balance-algorithm-name>.type= # Load balance algorithm type
spring.shardingsphere.rules.replica-query.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.

#### 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。

### 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

##### 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 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
```

#### 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

#### 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
```

#### 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
```

#### 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
```

#### Spring Namespace Configuration

##### Introduction

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

##### Spring Namespace Configuration Item

##### Configuration 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
                           ">
```

```
    <bean id="ds0" class="org.apache.commons.dbcp2.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.dbcp2.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>

    <!-- Rule configurations, please refer to specific rule configuration for more
details. -->
    <!-- ... -->

    <shardingsphere:data-source id="shardingDataSource" data-source-names="ds0,ds1"
rule-refs="..." >
        <props>
            <prop key="xxx.xxx">${xxx.xxx}</prop>
        </props>
    </shardingsphere:data-source>
</beans>
```

#### Configuration Item Explanation

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

<shardingsphere:data-source />

| *Name* | *Type* | *Description* |
|---|---|---|
| id | Attribute | Spring Bean Id |
| data-source-names | Attribute | Data source name, multiple data source names are separated by commas |
| rule-refs | Attribute | Rule name, multiple rule names are separated by commas |
| props (?) | Tag | Properties configuration, Please refer to Properties Configuration for more details |

### 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 |

<sharding:table-rule />

| *Name* | *Type* | *Description* |
|---|---|---|
| logic-table | Attribute | Logic table name |
| actual-data-nodes | Attribute | 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 | Attribute | Data source names for auto sharding table |
| database-strategy-ref | Attribute | Database strategy name for standard sharding table |
| table-strategy-ref | Attribute | Table strategy name for standard sharding table |
| sharding-strategy-ref | Attribute | sharding strategy name for auto sharding table |
| key-generate-strategy-ref | Attribute | 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* | • <br>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.

### 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 | A t t r i b u t e | Spring Bean Id | |
| que ryWithCip herCol- umn (?) | A t t r i b u t e | Whether query with cipher column for data encrypt. User you can use plaintext to query if have | true |
| table (+) | T a g | Encrypt table configuration | |

<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 |

#### Governance

##### Configuration Item Explanation

##### Management

```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:governance="http://shardingsphere.apache.org/schema/shardingsphere/
governance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
                           http://www.springframework.org/schema/beans/spring-
beans.xsd
                           http://shardingsphere.apache.org/schema/shardingsphere/
governance
                           http://shardingsphere.apache.org/schema/shardingsphere/
governance/governance.xsd
">
    <governance:reg-center id="regCenter" type="ZooKeeper" server-lists=
"localhost:2181" />
    <governance:data-source id="shardingDatabasesTablesDataSource" data-source-
names="demo_ds_0, demo_ds_1" reg-center-ref="regCenter" config-center-ref=
```

```
"configCenter" rule-refs="shardingRule" overwrite="true" />
</beans>
```

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

| *Name* | *Type* | *Description* |
|---|---|---|
| id | Attribute | Registry center name |
| type | Attribute | Registry center type. Example: ZooKeeper, etcd |
| server-lists | Attribute | The list of servers that connect to registry center, including IP and port number; use commas to separate |
| props (?) | Attribute | Properties for center instance config, such as options of zookeeper |

#### Mixed Rules

##### Configuration Item Explanation

```
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:tx="http://www.springframework.org/schema/tx"
       xmlns:shardingsphere="http://shardingsphere.apache.org/schema/
shardingsphere/datasource"
       xmlns: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://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.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.alibaba.druid.pool.DruidDataSource" 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.alibaba.druid.pool.DruidDataSource" init-
method="init" destroy-method="close">
        <!-- ...Omit specific configuration. -->
    </bean>

    <bean id="read_ds0_1" class="com.alibaba.druid.pool.DruidDataSource" init-
method="init" destroy-method="close">
        <!-- ...Omit specific configuration. -->
    </bean>

    <bean id="write_ds1" class="com.alibaba.druid.pool.DruidDataSource" init-
method="init" destroy-method="close">
        <!-- ...Omit specific configuration. -->
    </bean>

    <bean id="read_ds1_0" class="com.alibaba.druid.pool.DruidDataSource" init-
method="init" destroy-method="close">
        <!-- ...Omit specific configuration. -->
    </bean>

    <bean id="read_ds1_1" class="com.alibaba.druid.pool.DruidDataSource" 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>
```

## Change History

### 5.0.0-alpha

#### Replica Query

#### Configuration Item Explanation

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

<replica-query:rule />

| *Name* | *Type* | *Description* |
|---|---|---|
| id | Attribute | Spring Bean Id |
| data-source-rule (+) | Tag | Replica query data source rule configuration |

<replica-query:data-source-rule />

| *Name* | *Ty pe* | *Description* |
|---|---|---|
| id | Att rib ute | Primary-replica data source rule name |
| primary-da ta-source-name | Att rib ute | Primary data source name |
| replica-dat a-source-names | Att rib ute | Replica data source names, multiple data source names separated with comma |
| load-balance - algorithm-ref | Att rib ute | Load balance algorithm name |

<replica-query: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.

#### 4.x

#### 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* | *Type* | *Explanation* |
|---|---|---|
| id | Attribute | Spring Bean Id |
| type | Attribute | Type of load balance algorithm, 'RANDOM' 或 'ROUND_ROBIN', support custom extension |
| props-ref (?) | Attribute | Properties of load balance algorithm |

### 3.x

#### 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>
```

#### 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>
```

#### 1.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:rdb="http://www.dangdang.com/schema/ddframe/rdb"
    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.dangdang.com/schema/ddframe/rdb
                        http://www.dangdang.com/schema/ddframe/rdb/rdb.xsd
                        ">
    <!-- 配置真实数据源 -->
    <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>

    <!-- 定义读写分离数据源，读写分离数据源实现了 DataSource 接口，可直接当做数据源处理 -->
    <rdb:master-slave-data-source id="dbtbl_0" master-data-source-ref="dbtbl_0_
master" slave-data-sources-ref="dbtbl_0_slave_0, dbtbl_0_slave_1" strategy-type=
"ROUND_ROBIN" />
    <rdb:master-slave-data-source id="dbtbl_1" master-data-source-ref="dbtbl_1_
master" slave-data-sources-ref="dbtbl_1_slave_0, dbtbl_1_slave_1" strategy-type=
"ROUND_ROBIN" />

    <!-- 通过 rdb:strategy 和 rdb:data-source 继续构建分片数据源 -->
    <rdb:strategy id="databaseStrategy" sharding-columns="user_id" algorithm-
expression="dbtbl_${user_id.longValue() % 2}"/>
    <rdb:strategy id="orderTableStrategy" sharding-columns="order_id" algorithm-
expression="t_order_${order_id.longValue() % 4}"/>

    <rdb:data-source id="shardingDataSource">
        <rdb:sharding-rule data-sources="dbtbl_0, dbtbl_1">
            <rdb:table-rules>
                <rdb:table-rule logic-table="t_order" actual-tables="t_order_${0..
```

```
3}" database-strategy="databaseStrategy" table-strategy="orderTableStrategy"/>
            </rdb:table-rules>
        </rdb:sharding-rule>
    </rdb:data-source>
</beans>
```

#### Built-in Algorithm

##### Introduction

Apache ShardingSphere allows developers to implement algorithms via SPI; At the same time, Apache ShardingSphere also provides a couple of built-in algorithms for simplify developers.

##### Usage

The built-in 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 built-in algorithms of Apache ShardingSphere according to its functions for developers' reference.

##### 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 at-tern | String | Suffix pattern of sharding data sources or tables, must can be transformed to Java LocalDateTime, must be consistent with date-time-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* | • <br> D a t a T y p e * | *Description* | *D efa ult Val ue* |
|---|---|---|---|
| sh arding-columns (?) | S t r i n g | sharing column names | • |
| algori thm-expression | S t r i n g | Inline expression sharding algorithm | • |
| allow-rang e-query-with-i nline-sharding (?) | b o o l e a n | 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 |

### Properties Configuration

#### Introduction

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 |
| 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 |
| xa- tran sact ion- mana ger- type (?) | String | XA Transaction manager type. Include: Atomikos, Narayana and Bitronix. | Atomikos |

### 4.1.5 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.

## 4.2 ShardingSphere-Proxy

### 4.2.1 Introduction

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.

- 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.

### 4.2.2 Comparison

| | *Shardi ngSphere-JDBC* | *Shardin gSphere-Proxy* | *ShardingS phere-Sidecar* |
|---|---|---|---|
| Database | Any | `MySQ L/PostgreSQL` | MySQL |
| Connections Count Cost | High | `Low` | High |
| Supported Languages | Java Only | `Any` | Any |
| Performance | Low loss | `Relative ly high loss` | Low loss |
| De centralization | Yes | `No` | Yes |
| Static Entry | No | `Yes` | No |

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

### 4.2.3 Usage

This chapter will introduce the use of ShardingSphere-Proxy. Please refer to Example for more details.

#### Proxy Startup

#### Startup Steps

1. Download the latest version of ShardingSphere-Proxy.
2. If users use docker, they can implement `docker pull shardingsphere/shardingsphere-proxy` to get the clone. Please refer to Docker Clone for more details.
3. After the decompression, revise `conf/server.yaml` and documents begin with `config-` prefix, `conf/config-xxx.yaml` for example, to configure sharding rules and replica query rules. Please refer to Configuration Manual for the configuration method.

4. 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 PostgreSQL

1. Use any PostgreSQL client end 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 client end to connect, such as `mysql -u root -h 127.0.0.1 -P 3307`.

#### Using user-defined sharding algorithm

When developer need to use user-defined sharding algorithm, it can not configure via inline expression in YAML file simply, should use the way below to configure sharding algorithm.

1. Implement `ShardingAlgorithm` interface.
2. Package Java file to jar.
3. Copy jar to ShardingSphere-Proxy's `conf/lib-ext` 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 source, with each yaml configuration document named by `config-` prefix as a logic data source.

#### Governance

ShardingSphere-Proxy use SPI to support Governance, realize the unified management of configurations and metadata, as well as instance disabling and replica disabling.

#### Zookeeper

ShardingSphere-Proxy has provided the solution of Zookeeper in default, which implements the functions of config center, registry center. Configuration Rules consistent with ShardingSphere-JDBC YAML.

#### Other Third Party Components

Refer to Supported Third Party Components for details.

1. Use SPI methods in logic coding and put the generated jar package to the lib folder of ShardingSphere-Proxy.
2. Follow Configuration Rules to configure and use it.

#### Distributed Transaction

ShardingSphere-Proxy supports LOCAL, XA, BASE transactions, LOCAL transaction is default value, it is original transaction of relational database.

#### XA transaction

Default XA transaction manager of ShardingSphere is Atomikos. Users can customize Atomikos configuration items through adding `jta.properties` in conf catalog of ShardingSphere-Proxy. Please refer to Official Documents of Atomikos for detailed configurations.

- Use Narayana XA Transaction Manager。

1. Copy the jar file required by Narayana to `conf/lib`. The reference package is as follows:

```
<propeties>
    <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>
</propeties>
<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>
```

2. Configure `xa-transaction-manager-type` in `conf/server.yaml`:

```
props:
  xa-transaction-manager-type: Narayana
```

3. Add `jbossts-properties.xml` to customize Narayana configuration. The order of path loading is`user.dir (pwd) > user.home > java.home > classpath`. Please refer to Narayana official documentation for more details.

- Use Bitronix XA Transaction Manager。

1. Copy the jar file required by Bitronix to `conf/lib`. The reference package is as follows:

```
<propeties>
    <btm.version>2.1.3</btm.version>
</propeties>

<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>
```

2. Configure `xa-transaction-manager-type` in `conf/server.yaml`:

```
props:
  xa-transaction-manager-type: Bitronix
```

3. Please refer to Bitronix official documentation for more details.

#### BASE Transaction

Since we have not packed the BASE implementation jar into ShardingSphere-Proxy, you should copy relevant jar which implement `ShardingTransactionManager` SPI to `conf/lib`, then switch the transaction type to `BASE`.

### 4.2.4 Configuration Manual

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 only provided YAML configuration. By configuration, application developers can flexibly use data sharding, replica query, data encryption, shadow database or the combination of them.

Rule configuration keeps consist with YAML configuration of ShardingSphere-JDBC.

#### Data Source Configuration

#### Configuration Item Explanation

```
schemaName: # Logic schema name

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
    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

rules: # Keep consist with ShardingSphere-JDBC configuration
  # ...
```

For more data source configuration parameters, see HikariCP.

#### 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.

```
rules:
  - !AUTHORITY
    users:
      - root@localhost:root # <username>@<hostname>:<password>
      - sharding@:sharding
    provider:
      type: NATIVE
```

If the hostname is % or empty, it means no restrict to the user's host.

The type of the provider must be explicitly specified. Refer to 5.11 Proxy for more implementations.

#### Properties Configuration

#### Introduction

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

### Configuration Item Explanation

| *Name* | • 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 |
| sql-simple (?) | boolean | Whether show SQL details in simple style. | false |
| executor-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 |
| max-connections-size-per-query (?) | int | Max opened connection size for each query. | 1 |
| check-table-metadata-enabled (?) | boolean | Whether validate table meta data consistency when application startup or updated. | false |
| proxy-frontend-flush-threshold (?) | int | Flush threshold for every records from databases for ShardingSphere-Proxy. | 128 |
| proxy-transaction-type (?) | String | Default transaction type of ShardingSphere-Proxy. Include: LOCAL, XA and BASE. | LOCAL |
| proxy-opentracing-enabled (?) | boolean | Whether enable opentracing for ShardingSphere-Pr... | false |

#### YAML Syntax

`!!` means instantiation of that class

`!` means self-defined alias

`-` means one or multiple can be included

`[]` means array, can substitutable with `-` each other

### 4.2.5 Docker Clone

#### Pull Official Docker Clone

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

#### Build Docker Clone 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 clone, 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 designated PostgreSQL IP in `/${your_work_dir}/conf/config-xxx.yaml` configuration is accessible to Docker container.

Question 3: How to start ShardingProxy whose backend databases are MySQL.

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

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

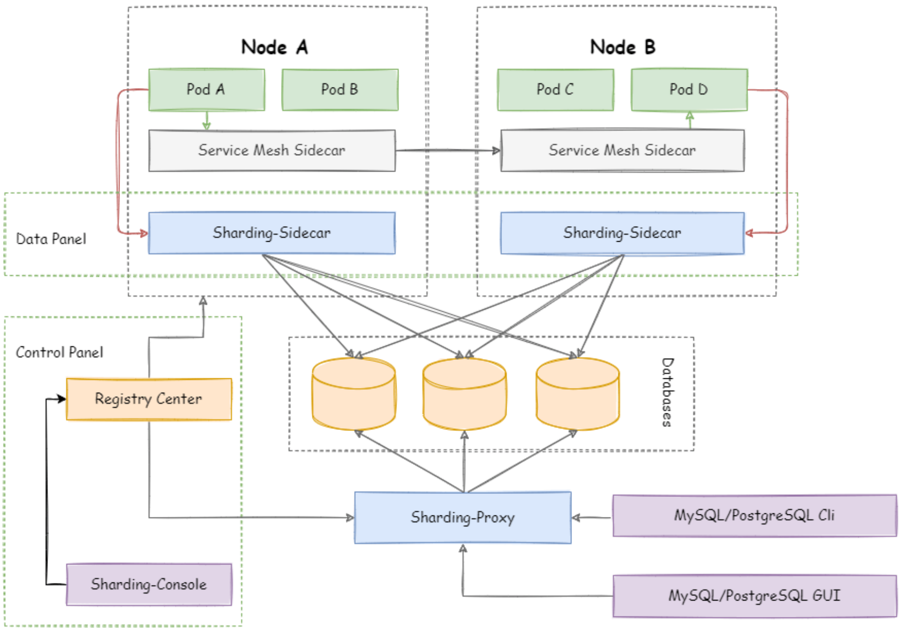
Answer: Volume the directory where `shardingsphere-strategy.jar` stores to /opt/shardingsphere-proxy/ext-lib.

## 4.3 ShardingSphere-Sidecar

### 4.3.1 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`.




### 4.3.2 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.

## 4.4 ShardingSphere-Scaling

### 4.4.1 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 **Alpha** version.

### 4.4.2 Build

#### Build&Deployment

1. Execute the following command to compile and generate the ShardingSphere-Scaling binary package:

```
git clone https://github.com/apache/shardingsphere.git;
cd shardingsphere;
mvn clean install -Prelease;
```

The binary package's directory is:`/shardingsphere-distribution/shardingsphere-scaling-distribution/target/apache-shardingsphere-${latest.release.version}-shardingsphere-scaling-bin.tar.gz`。

2. Unzip the distribution package, modify the configuration file `conf/server.yaml`, we should ensure the port does not conflict with others, and modify the resume from break-point(optional) server lists:

```
port: 8888
blockQueueSize: 10000
workerThread: 30

resumeBreakPoint:
  name: scalingJob
  registryCenter:
    type: ZooKeeper
    serverLists: localhost:2181
    props:
      retryIntervalMilliseconds: 10000
```

3. Start up ShardingSphere-Scaling:

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

4. See the log file `logs/stdout.log`, ensure startup successfully.
5. Ensure startup successfully by `curl`.

```
curl -X GET http://localhost:8888/scaling/job/list
```

response:

```
{"success":true,"errorCode":0,"errorMsg":null,"model":[]}
```

#### Shutdown ShardingSphere-Scaling

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

#### Configuration

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

| Name | Description | De fault value |
|---|---|---|
| port | Listening port of HTTP server | 8888 |
| bloc kQueu eS-ize | Queue size of data transmission channel | 10000 |
| wo rkerT hread | Worker thread pool size, the number of migration task threads allowed to run concurrently | 30 |
| r esume Break Point | Resume from break-point service | |

### 4.4.3 Manual

#### Manual

#### Environment

JAVA, JDK 1.8+.

The migration scene we support:

| Source | Target | Support |
|---|---|---|
| MySQL(5.1.15 ~ 5.7.x) | ShardingSphere-Proxy | Yes |
| PostgreSQL(9.4 ~ ) | ShardingSphere-Proxy | Yes |

**Attention**:

If the backend database is MySQL, please download mysql-connector-java-5.1.47.jar and put it into `${shardingsphere-scaling}\lib directory`.

### Privileges

MySQL need to open `binlog`, and `binlog format` should be Row model. Privileges of users scaling used should include Replication privileges.

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

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

PostgreSQL need to support and open test_decoding feature.

### API

ShardingSphere-Scaling provides a simple HTTP API

### Start scaling job

Interface description: POST /scaling/job/start

Body:

| Parameter | Describe |
|---|---|
| ruleConfig.source | source data source configuration |
| ruleConfig.target | target data source configuration |
| jobConfiguration.concurrency | sync task proposed concurrency |

Data source configuration:

| Parameter | Describe |
|---|---|
| type | data source type(available parameters:shardingSphereJdbc,jdbc) |
| parameter | data source parameter |

*** Notice ***

Currently source type must shardingSphereJdbc

Example:

```
curl -X POST \
  http://localhost:8888/scaling/job/start \
  -H 'content-type: application/json' \
  -d '{
        "ruleConfig": {
          "source": {
            "type": "shardingSphereJdbc",
            "parameter": "
                dataSources:
                  ds_0:
                    dataSourceClassName: com.zaxxer.hikari.HikariDataSource
                    jdbcUrl: jdbc:mysql://127.0.0.1:3306/scaling_0?useSSL=false
                    username: scaling
                    password: scaling
                  ds_1:
                    dataSourceClassName: com.zaxxer.hikari.HikariDataSource
                    jdbcUrl: jdbc:mysql://127.0.0.1:3306/scaling_1?useSSL=false
                    username: scaling
                    password: scaling
                rules:
                - !SHARDING
                  tables:
                    t_order:
                      actualDataNodes: ds_$->{0..1}.t_order_$->{0..1}
                      databaseStrategy:
                        standard:
                          shardingColumn: order_id
                          shardingAlgorithmName: t_order_db_algorith
                      logicTable: t_order
                      tableStrategy:
                        standard:
                          shardingColumn: user_id
                          shardingAlgorithmName: t_order_tbl_algorith
                  shardingAlgorithms:
                    t_order_db_algorith:
                      type: INLINE
                      props:
                        algorithm-expression: ds_$->{order_id % 2}
                    t_order_tbl_algorith:
                      type: INLINE
                      props:
                        algorithm-expression: t_order_$->{user_id % 2}
                "
          },
          "target": {
              "type": "jdbc",
              "parameter": "
                username: root
```

```
                password: root
                jdbcUrl: jdbc:mysql://127.0.0.1:3307/sharding_db?
serverTimezone=UTC&useSSL=false
                "
          }
        },
        "jobConfiguration":{
          "concurrency":"3"
        }
      }'
```

Response:

```
{
   "success": true,
   "errorCode": 0,
   "errorMsg": null,
   "model": null
}
```

#### Get scaling progress

Interface description: GET /scaling/job/progress/{jobId}

Example:

```
curl -X GET \
  http://localhost:8888/scaling/job/progress/1
```

Response:

```
{
   "success": true,
   "errorCode": 0,
   "errorMsg": null,
   "model": {
        "id": 1,
        "jobName": "Local Sharding Scaling Job",
        "status": "RUNNING/STOPPED"
        "syncTaskProgress": [{
            "id": "127.0.0.1-3306-test",
            "status": "PREPARING/MIGRATE_HISTORY_DATA/SYNCHRONIZE_REALTIME_DATA/
STOPPING/STOPPED",
            "historySyncTaskProgress": [{
                "id": "history-test-t1#0",
                "estimatedRows": 41147,
                "syncedRows": 41147
            }, {
```

```
                "id": "history-test-t1#1",
                "estimatedRows": 42917,
                "syncedRows": 42917
            }, {
                "id": "history-test-t1#2",
                "estimatedRows": 43543,
                "syncedRows": 43543
            }, {
                "id": "history-test-t2#0",
                "estimatedRows": 39679,
                "syncedRows": 39679
            }, {
                "id": "history-test-t2#1",
                "estimatedRows": 41483,
                "syncedRows": 41483
            }, {
                "id": "history-test-t2#2",
                "estimatedRows": 42107,
                "syncedRows": 42107
            }],
            "realTimeSyncTaskProgress": {
                "id": "realtime-test",
                "delayMillisecond": 1576563771372,
                "position": {
                    "filename": "ON.000007",
                    "position": 177532875,
                    "serverId": 0
                }
            }
        }]
    }
}
```

#### List scaling jobs

Interface description: GET /scaling/job/list

Example:

```
curl -X GET \
  http://localhost:8888/scaling/job/list
```

Response:

```
{
  "success": true,
  "errorCode": 0,
  "model": [
```

```
    {
      "jobId": 1,
      "jobName": "Local Sharding Scaling Job",
      "status": "RUNNING"
    }
  ]
}
```

#### Stop scaling job

Interface description: GET /scaling/job/stop

Body:

| Parameter | Describe |
|---|---|
| jobId | job id |

Example:

```
curl -X GET \
  http://localhost:8888/scaling/job/stop/1
```

Response:

```
{
   "success": true,
   "errorCode": 0,
   "errorMsg": null,
   "model": null
}
```

#### Operate through the UI interface

We provide user interface in ShardingSphere-UI, so all the operations related can be implemented with a click of the UI interface. For more information, please refer to the ShardingSphere-UI module.

## 4.5 ShardingSphere-UI

### 4.5.1 Introduction

ShardingSphere-UI is a simple and useful web administration console for ShardingSphere. It is designed to help users more easily use ShardingSphere, and currently provides ability of registry center management, dynamic configuration management, database governance, etc.

The frontend of project uses vue as javascript framework and the backend is a standard spring boot project. You can deploy it with maven, and also run locally by separating the frontend and backend.

## 4.5.2 Manual

### Navigation

This chapter will introduce the use of ShardingSphere-UI.

### Build

### Binary Run

1. `git clone https://github.com/apache/shardingsphere-ui.git`;
2. Run `mvn clean install -Prelease`;
3. Get the package in `/shardingsphere-ui/shardingsphere-ui-distribution/target/apache-shardingsphere-${latest.release.version}-shardingsphere-ui-bin.tar.gz`;
4. After the decompression, run `bin/start.sh`;
5. visit `http://localhost:8088/`.

### Source Code Debug

ShardingSphere-UI use frontend and backend separately mode.

### backend

1. Main class is `org.apache.shardingsphere.ui.Bootstrap`;
2. visit `http://localhost:8088/`.

### frontend

1. `cd shardingsphere-ui-frontend/`;
2. run `npm install`;
3. run `npm run dev`;
4. visit `http://localhost:8080/`.

#### Configuration

Configuration file of ShardingSphere-UI is conf/application.properties in distribution package. It is constituted by two parts.

1. Listening port;
2. authentication.

```
server.port=8088

user.admin.username=admin
user.admin.password=admin
```

#### Notices

1. If you run the frontend project locally after a build with maven, you may fail to run it due to inconsistent version of node. You can clean up `node_modules/` directory and run it again. The error log is:

```
ERROR  Failed to compile with 17 errors
error  in ./src/views/orchestration/module/instance.vue?vue&type=style&index=0&
id=9e59b740&lang=scss&scoped=true&
Module build failed (from ./node_modules/sass-loader/dist/cjs.js):
Error: Missing binding /shardingsphere/shardingsphere-ui/shardingsphere-ui-
frontend/node_modules/node-sass/vendor/darwin-x64-57/binding.node
Node Sass could not find a binding for your current environment: OS X 64-bit with
Node.js 8.x
Found bindings for the following environments:
  - OS X 64-bit with Node.js 6.x
This usually happens because your environment has changed since running `npm
install`.
Run `npm rebuild node-sass` to download the binding for your current environment.
```

#### Registry Center

#### Registry Center Configuration

The registry center needs to be added and activated first. Multiple registries can be added, but only one is active, and the following runtime status operate on the currently active registry. Zookeeper and etcd support is provided now, and the support for other registries will be added later.

- Click the + button to add a new registry center.
- Use additional config center to manage configuration with other config centers.
- Registry center can be edited, activated, and deleted.

#### Rule Config

##### Rule Config

- After added and activated a registry center, the configuration of all data sources in the current active registry center can be obtained, including data sharding, replica query, properties, and so on.
- The configuration can be modified by the YAML format.
- Click the + button to add a new data source and sharding rule.

#### Runtime Status

##### Runtime Status

- After added and activated a registry center, all running instances of the current registry center will be obtained.
- Users can disable or enable the instance by operate button.
- Users can disable or enable the access to replica database.

# 5 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.

## 5.1 SQL Parser

### 5.1.1 SQLParserFacade

| *SPI Name* | *Description* |
|---|---|
| SQLParserFacade | 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 |

### 5.1.2 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 |

## 5.2 Configuration

### 5.2.1 ShardingSphereRuleBuilder

| *SPI Name* | *Description* |
|---|---|
| ShardingSphe reRuleBuilder | Used to convert user configurations to rule objects |

| *I mplementation Class* | *Description* |
|---|---|
| Shardi ngRuleBuilder | Used to convert user sharding configurations to sharding rule objects |
| ReplicaQue ryRule-Builder | Used to convert user replica query configurations to replica query rule objects |
| Encry ptRuleBuilder | Used to convert user encryption configurations to encryption rule objects |
| Shad owRuleBuilder | Used to convert user shadow database configurations to shadow database rule objects |

### 5.2.2 YamlRuleConfigurationSwapper

| *SPI Name* | *Description* |
|---|---|
| YamlRuleC onfigurationSwapper | Used to convert YAML configuration to standard user configuration |

| *Implementation Class* | *Description* |
|---|---|
| ShardingRuleConfigurationYamlSwapper | Used to convert YAML sharding configuration to standard sharding configuration |
| Re plicaQueryRuleConfigurationYamlSwapper | Used to convert YAML replica query configuration to standard replica query configuration |
| EncryptRuleConfigurationYamlSwapper | Used to convert YAML encryption configuration to standard encryption configuration |
| ShadowRuleConfigurationYamlSwapper | Used to convert YAML shadow database configuration to standard shadow database configuration |

### 5.2.3 ShardingSphereYamlConstruct

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

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

## 5.3 Kernel

### 5.3.1 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 |

### 5.3.2 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 |

### 5.3.3 SQLRouter

| *SPI Name* | *Description* |
|---|---|
| SQLRouter | Used to process routing results |

| *Implementation Class* | *Description* |
|---|---|
| ShardingSQLRouter | Used to process sharding routing results |
| ReplicaQuerySQLRouter | Used to process replica query routing results |
| ShadowRouteSQLRouter | Used to process shadow database routing results |

### 5.3.4 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 |

### 5.3.5 SQLExecutionHook

| *SPI Name* | *Description* |
|---|---|
| SQLExecutionHook | Hook of SQL execution |

| *Implementation Class* | *Description* |
|---|---|
| TransactionalSQLExecutionHook | Transaction hook of SQL execution |
| OpenTracingSQLExecutionHook | Open tracing hook of SQL execution |

### 5.3.6 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 |

### 5.3.7 StoragePrivilegeLoader

| *SPI Name* | *Description* |
|---|---|
| StoragePrivilegeLoader | Use SQL dialect to load privilege metadata |

| *Implementation Class* | *Description* |
|---|---|
| MySQLPrivilegeLoader | Use MySQL dialect to load privilege metadata |
| O raclePrivilegeLoader | Use Oracle dialect to load privilege metadata |
| Postg reSQLPrivilegeLoader | Use PostgreSQL dialect to load privilege metadata |

## 5.4 Data Sharding

### 5.4.1 ShardingAlgorithm

| *SPI Name* | *Description* |
|---|---|
| ShardingAlgorithm | Sharding algorithm |

| *Implementation Class* | *Description* |
|---|---|
| InlineShardingAlgorithm | Inline sharding algorithm |
| ModShardingAlgorithm | Modulo sharding algorithm |
| HashModShardingAlgorithm | Hash modulo sharding algorithm |
| FixedIntervalShardingAlgorithm | Fixed interval sharding algorithm |
| M utableIntervalShardingAlgorithm | Mutable interval sharding algorithm |
| Vo lumeBasedRangeShardingAlgorithm | Volume based range sharding algorithm |
| Boun daryBasedRangeShardingAlgorithm | Boundary based range sharding algorithm |
| ClassBasedShardingAlgorithm | Class based sharding algorithm |

### 5.4.2 KeyGenerateAlgorithm

| *SPI Name* | *Description* |
|---|---|
| KeyGenerateAlgorithm | Key generate algorithm |

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

### 5.4.3 TimeService

| *SPI Name* | *Description* |
|---|---|
| TimeService | Use current time for routing |

| *Implementation Class* | *Description* |
|---|---|
| DefaultTimeService | Get the current time from the application system for routing |
| Database TimeServiceDelegate | Get the current time from the database for routing |

### 5.4.4 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 |

## 5.5 Readwrite-splitting

### 5.5.1 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 |

## 5.6 Data Encryption

### 5.6.1 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 |

### 5.6.2 QueryAssistedEncryptAlgorithm

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

| *Implementation Class* | *Description* |
|---|---|
| None | |

## 5.7 SQL Audit

### 5.7.1 SQLAuditor

| *SPI Name* | *Description* |
|---|---|
| SQLAuditor | SQL auditor |

| *Implementation Class* | *Description* |
|---|---|
| None | |

## 5.8 Distributed Transaction

### 5.8.1 ShardingTransactionManager

| *SPI Name* | *Description* |
|---|---|
| ShardingTransactionManager | Distributed transaction manager |

| *Implementation Class* | *Description* |
|---|---|
| XAShardingTransactionManager | XA distributed transaction manager |
| SeataATShardingTransactionManager | Seata distributed transaction manager |

### 5.8.2 XATransactionManager

| *SPI Name* | *Description* |
|---|---|
| XATransactionManager | XA distributed transaction manager |

| *Implementation Class* | *Description* |
|---|---|
| Atom ikosTransactionManager | XA distributed transaction manager based on Atomikos |
| Naraya naXATransactionManager | XA distributed transaction manager based on Narayana |
| Bitron ixXATransactionManager | XA distributed transaction manager based on Bitronix |

### 5.8.3 XADataSourceDefinition

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

| *Implementation Class* | *Description* |
|---|---|
| MySQLXAD ataSourceDefinition | Auto convert Non XA MySQL data source to XA MySQL data source |
| MariaDBXAD ataSourceDefinition | Auto convert Non XA MariaDB data source to XA MariaDB data source |
| PostgreSQLXAD ataSourceDefinition | 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 ataSourceDefinition | 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 |

### 5.8.4 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 |

## 5.9 Distributed Governance

### 5.9.1 RegistryCenterRepository

| *SPI Name* | *Description* |
|---|---|
| RegistryCenterRepository | Registry center repository |

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

### 5.9.2 GovernanceWatcher

| *SPI Name* | *Description* |
|---|---|
| GovernanceWatcher | Governance watcher |

| *Implementation Class* | *Description* |
|---|---|
| TerminalStateChangedWatcher | Terminal state changed watcher |
| DataSourceStateChangedWatcher | Data source state changed watcher |
| LockChangedWatcher | Lock changed watcher |
| PropertiesChangedWatcher | Properties changed watcher |
| PrivilegeNodeChangedWatcher | Privilege changed watcher |
| GlobalRuleChangedWatcher | Global rule changed watcher |

## 5.10 Scaling

### 5.10.1 ScalingEntry

| *SPI Name* | *Description* |
|---|---|
| ScalingEntry | Entry of scaling |

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

## 5.11 Proxy

### 5.11.1 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 |

### 5.11.2 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 |

### 5.11.3 AuthorityProvideAlgorithm

| *SPI Name* | *Description* |
|---|---|
| AuthorityProvideAlgorithm | User authority loading logic |

| *Implementation Class* | *Type* | *Description* |
|---|---|---|
| NativeAuthorityProviderAlgorithm | NATIVE | Persist user authority defined in server.yaml into the backend database. An admin user will be created if not existed. |
| AllPrivilegesPermittedAuthorityProviderAlgorithm | ALL_PRIVILEGES_PERMITTED | All privileges granted to user by default (No authentication). Will not interact with the actual database. |

# 6 Downloads

## 6.1 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.

### 6.1.1 Apache ShardingSphere - Version: 5.0.0-alpha ( Release Date: Nov 10, 2020 )

- Source Codes: [ SRC ] [ ASC ] [ SHA512 ]
- ShardingSphere-JDBC Binary Distribution: [ TAR ] [ ASC ] [ SHA512 ]
- ShardingSphere-Proxy Binary Distribution: [ TAR ] [ ASC ] [ SHA512 ]
- ShardingSphere-Scaling Binary Distribution: [ TAR ] [ ASC ] [ SHA512 ]

### 6.1.2 ShardingSphere UI - Version: 5.0.0-alpha ( Release Date: Nov 22, 2020 )

- Source Codes: [ SRC ] [ ASC ] [ SHA512 ]
- ShardingSphere-UI Binary Distribution: [ TAR ] [ ASC ] [ SHA512 ]

## 6.2 All Releases

Find all releases in the Archive repository. Find all incubator releases in the Archive incubator repository.

## 6.3 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
```

# 7 FAQ

## 7.1 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.

## 7.2 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`.

## 7.3 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.

## 7.4 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.

## 7.5 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).

## 7.6 If sharding database is partial, should tables without sharding database & table be configured in sharding rules?

Answer:

Yes. ShardingSphere merges multiple data sources to a united logic data source. Therefore, for the part without sharding database or table, ShardingSphere can not decide which data source to route to without sharding rules. However, ShardingSphere has provided two options to simplify configurations.

Option 1: configure default-data-source. All the tables in default data sources need not to be configured in sharding rules. ShardingSphere will route the table to the default data source when it cannot find sharding data source.

Option 2: isolate data sources without sharding database & table from ShardingSphere; use multiple data sources to process sharding situations or non-sharding situations.

## 7.7 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.

## 7.8 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.

## 7.9 In SQLSever and PostgreSQL, why does the aggregation column without alias throw exception?

Answer:

SQLServer and PostgreSQL will rename aggregation columns acquired without alias, such as the following SQL:

```
SELECT SUM(num), SUM(num2) FROM tablexxx;
```

Columns acquired by SQLServer are empty string and (2); columns acquired by PostgreSQL are empty sum and sum(2). It will cause error because ShardingSphere is unable to find the corresponding column.

The right SQL should be written as:

```
SELECT SUM(num) AS sum_num, SUM(num2) AS sum_num2 FROM tablexxx;
```

## 7.10 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;
}
```

## 7.11 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.

## 7.12 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

## 7.13 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

## 7.14 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
```

## 7.15 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.

## 7.16 Why does my custom distributed primary key do not work after implementing `ShardingKeyGenerator` 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.

## 7.17 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.

## 7.18 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 from `the default dataSource`.
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).

## 7.19 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`).

## 7.20 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.

## 7.21 How to add a new logic schema dynamically when use ShardingSphere-Proxy?

Answer:

1. Before version 4.1.0, sharing-proxy can't support adding a new logic schema dynamically, for example, when a proxy starting with two logic schemas, it always hold the two schemas and will be notified about the table/rule changed events in the two schemas.
2. Since version 4.1.0, sharing-proxy support adding a new logic schema dynamically via ShardingSphere-UI or zookeeper, and it's a plan to support removing a exist logic schema dynamically in runtime.

## 7.22 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. For example, we recommend Navicat with version 11.1.13(not 12.x), and turn on "introspect using jdbc metadata" (or it will get all real tables info from informations_schema) in idea or datagrip.

## 7.23 Found a JtaTransactionManager in spring boot project when integrating with ShardingTransaction of XA

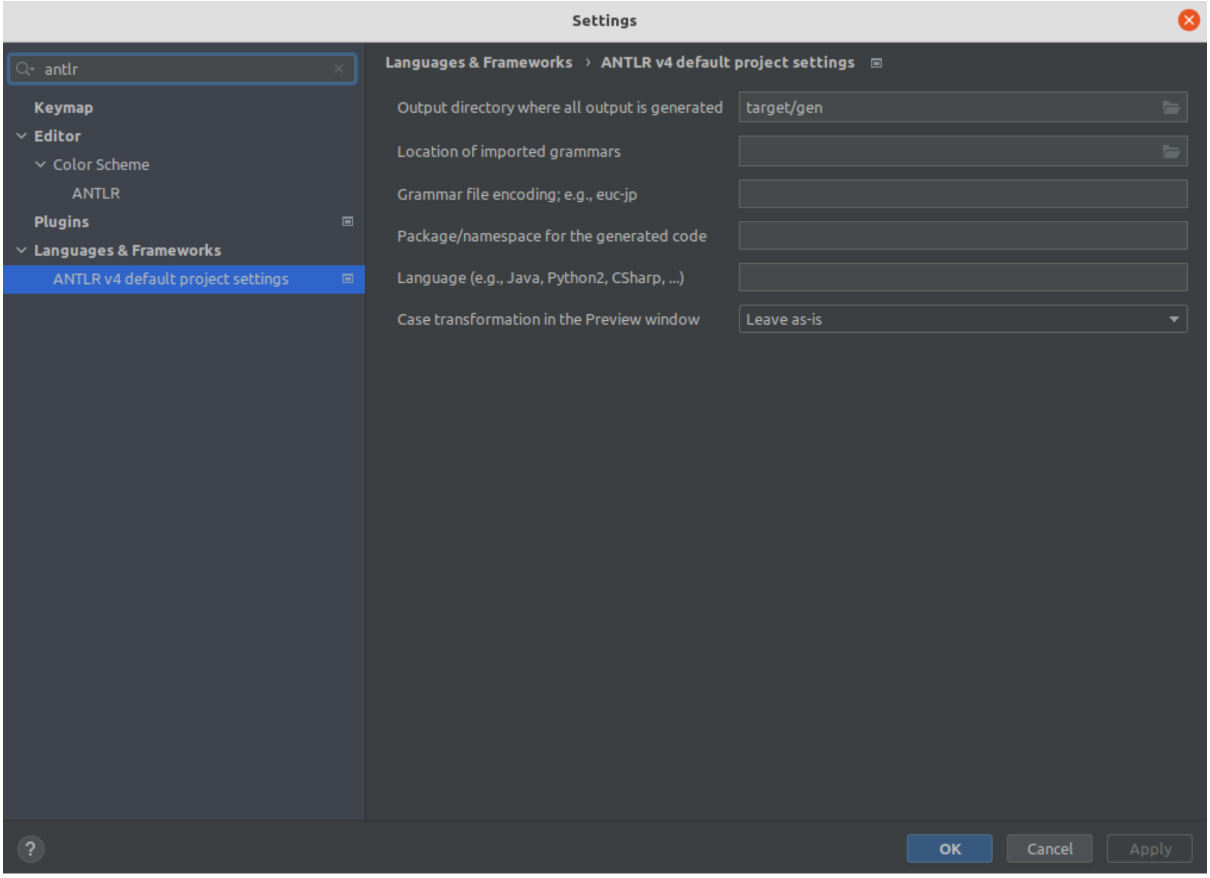
Answer:

1. `shardingsphere-transaction-xa-core` include atomikos, it will trigger auto-configuration mechanism in spring-boot, add `@SpringBootApplication(exclude = JtaAutoConfiguration.class)` will solve it.

## 7.24 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:


Settings
antlr
Keymap
Editor
Color Scheme
ANTLR
Plugins
Languages & Frameworks
ANTLR v4 default project settings
Languages & Frameworks › ANTLR v4 default project settings
Output directory where all output is generated
target/gen
Location of imported grammars
Grammar file encoding; e.g., euc-jp
Package/namespace for the generated code
Language (e.g., Java, Python2, CSharp, ...)
Case transformation in the Preview window
Leave as-is
OK
Cancel
Apply